# TCP/IP for MVS, VM, OS/2 and DOS X Window System Guide

Document Number GG24-3911-01

July 1994

International Technical Support Organization
Raleigh Center

**Take Note!**

Before using this information and the product it supports, be sure to read the general information under "Special Notices" on page xv.

**Second Edition (July 1994)**

This edition applies to:

- Version 2.2.1 of IBM TCP/IP for MVS, Program Number 5735-HAL for use with the MVS Operating System
- Version 2.2 of IBM TCP/IP for VM, Program Number 5735-FAL for use with the VM Operating System
- Version 1.2.5 of IBM AIXwindows Environment/6000, Program Number 5601-257 for use with AIX Version 3.2.5 for the RISC System/6000
- Version 2.0 of IBM TCP/IP for OS/2, Program Number 5622-086 for use with the OS/2 Operating System
- Version 2.1.1 of IBM TCP/IP for DOS, Program Number 5621-219, and Version 3.3 of HCL-eXceed/DOS for use with the DOS Operating System.
- Version 2.1.1 of IBM TCP/IP for DOS, Program Number 5621-219, and Version 3.3.3 of HCL-eXceed/W for use with the DOS Operating System and Microsoft Windows.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

An ITSO Technical Bulletin Evaluation Form for readers′ feedback appears facing Chapter 1. If this form has been removed, comments may be addressed to:

IBM Corporation, International Technical Support Center
Dept. 545, Building 657
P.O. Box 12195
Research Triangle Park, NC 27709

# Abstract

The purpose of this document is to provide information pertinent to the implementation of the X Window Systems for TCP/IP under MVS, VM, AIX/6000, OS/2 and DOS.

The document focuses on how to customize and use the X Window Systems provided by the following products:

- IBM TCP/IP Version 2.2.1 for MVS
- IBM TCP/IP Version 2.2 for VM
- IBM AIXwindows Environment/6000 Version 1.2.5
- IBM TCP/IP Version 2.0 for OS/2
- HCL-eXceed/DOS Version 3.3 with TCP/IP Version 2.1.1 for DOS
- HCL-eXceed/W Version 3.3.3 with TCP/IP Version 2.1.1 for DOS

The intended audience for this document are customers and IBM system engineers who will evaluate and implement X Windows on one or more of the above platforms. It is assumed that the reader has a working knowledge of TCP/IP and each of the operating systems listed above.

(195 pages)

# Contents

# Figures

# Tables

## Special Notices

This publication is intended to help the customer to understand and install the X Window Systems for MVS, VM, AIX, OS/2 and DOS. The information in this publication is not intended as the specification of any programming interfaces that are provided by:

- IBM TCP/IP Version 2.2.1 for MVS
- IBM TCP/IP Version 2.2 for VM
- IBM AIXwindows Environment/6000 Version 1.2.5 for AIX/6000
- IBM TCP/IP Version 2.0 for OS/2
- IBM TCP/IP Version 2.1.1 for DOS

See the PUBLICATIONS section of the IBM Programming Announcement for these products for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 208 Harbor Drive, Stamford, CT 06904 USA.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM (″vendor″) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

The following terms, which are denoted by an asterisk (*) in this publication, are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX
AIXwindows
GDDM
graPHIGS
IBM
InfoExplorer

# Preface

The purpose of this document is to provide information pertinent to the implementation of the X Window System for TCP/IP under MVS, VM, AIX/6000, OS/2 and DOS.

Specifically the document addresses the customization and interoperability of the X Window Systems provided by the following products:

- IBM TCP/IP Version 2.2.1 for MVS running under MVS/ESA
- IBM TCP/IP Version 2.2 for VM running under VM/ESA and VM/SP
- IBM AIXwindows Environment/6000 Version 1.2.5 running under AIX/6000 Version 3.2.5
- IBM TCP/IP Version 2.0 for OS/2 running under OS/2 Version 2.1
- HCL-eXceed/DOS Version 3.3 with TCP/IP Version 2.1.1 for DOS running under DOS Version 6.1
- HCL-eXceed/W Version 3.3.3 with TCP/IP Version 2.1.1 for DOS running under DOS Version 6.1 with Microsoft Windows**Version 3.1

Although this document is not intended as an X Window System application programming guide, the X client application environments are discussed for MVS, VM, AIX/6000 and OS/2. Information is provided that will allow a user to understand what is required to get an X application working on each of these systems.

X Window System interoperability with a non-IBM platform is also discussed and demonstrated.

This document is intended for persons who will:

1. Evaluate the client/server functions of the X Window Systems for the appropriate IBM system platforms described above.
2. Customize and implement the X Window Systems on the system platforms described above.
3. Plan for and develop suitable X Window System client application environments on the system platforms described above.

It is assumed that the reader has a working knowledge of TCP/IP and each of the operating systems listed above.

The document is organized as follows:

- Chapter 1, "Introduction"

  The introduction provides an overview of the X Window System and describes each of the X Window Systems as implemented on the MVS, VM, AIX/6000, OS/2, and DOS platforms.

- Chapter 2, "Installation"

  This provides a description of the steps to install and verify the basic X Window System on each of the MVS, VM, AIX/6000, OS/2 and DOS platforms.

- Chapter 3, "X Client Application Considerations"

  This chapter describes the X client application environment under MVS, VM, AIX/6000 and OS/2. It includes considerations for creating, compiling, link-editing and running X client applications.

- Chapter 4, "Customizing the X Server"

  This chapter describes the steps to customize the X Window Systems on each of the server platforms, AIX/6000, OS/2 and DOS. It includes guidelines for customizing the workstation environment including colors, fonts and keyboard mappings.

- Chapter 5, "Multivendor Interoperability"

  This chapter describes the steps taken to achieve interoperability between the IBM X Window System implementations and a Hewlett-Packard X Window System.

# Related Publications

The following publications are considered particularly suitable for a more detailed discussion of the topics covered in this document.

## Prerequisite Publications

- *TCP/IP for MVS: Planning and Customization*, SC31-6085
- *TCP/IP for MVS: User's Guide*, SC31-6088
- *TCP/IP for MVS: Programmer's Reference*, SC31-6087
- *TCP/IP for VM: Planning and Customization*, SC31-6082
- *TCP/IP for VM: User's Guide*, SC31-6081
- *TCP/IP for VM: Programmer's Reference*, SC31-6084
- *TCP/IP for OS/2: Installation and Administration*, SC31-6075
- *TCP/IP for OS/2: User's Guide*, SC31-6076
- *TCP/IP for OS/2: Command Reference*, SX75-0070
- *TCP/IP for OS/2: Programmer's Reference*, SC31-6077
- *TCP/IP for OS/2: X Window System Client Guide*, SC31-7087
- *TCP/IP for OS/2: X Window System Server Guide*, SC31-7070
- *TCP/IP for DOS: Installation and Administration Guide*, SC31-7047
- *TCP/IP for DOS: User's Guide*, SC31-7045
- *TCP/IP for DOS: Command Reference*, SX75-0083
- *AIX for RISC System/6000 Installation Guide*, SC23-2341
- *HCL-eXceed/W for Microsoft Windows User's Guide*, by Hummingbird Communications Ltd., Manual Release 3.3.3
- *HCL-eXceed/DOS User's Guide*, by Hummingbird Communications Ltd., Manual Release 3.3

## Additional Publications

- *AIX Hypertext Information Base Library*, SC23-2163
- *X Window System User's Guide*, by Valerie Quercia and Tim O'Reilly, Published by O'Reilly & Associates, Inc., ISBN 1-56592-015-5
- *A Beginner's Guide to the X Window System*, by Hewlett-Packard Company, HP Part Number 98594-90001
- *Configuring the X Window System on Series 300*, by Hewlett-Packard Company, HP Part Number 98594-90025
- *The Joy of X*, by Niall Mansfield, Published by Addison-Wesley Co., ISBN 0-201-56512-9

## International Technical Support Center Publications

- *IBM TCP/IP for MVS Installation and Interoperability*, GG24-3687
- *IBM TCP/IP for VM Installation and Interoperability*, GG24-3624
- *IBM TCP/IP for OS/2 Installation and Interoperability*, GG24-3531
- *AIXwindows Environment Version 1.2 2D Base Features*, GG24-3813
- *AIXwindows Programming Guide*, GG24-3382

A complete list of International Technical Support Center publications, with a brief description of each, may be found in *Bibliography of International Technical Support Centers Technical Bulletins,* GG24-3070.

To get listings of ITSO technical bulletins (redbooks) online, VNET users may type:

```
TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
```

**How to Order ITSO Technical Bulletins (Redbooks)**

IBM employees in the USA may order ITSO books and CD-ROMs using PUBORDER. Customers in the USA may order by calling 1-800-879-2755 or by faxing 1-800-284-4721. Visa and Master Cards are accepted. Outside the USA, customers should contact their local IBM office.

Customers may order hardcopy redbooks individually or in customized sets, called GBOFs, which relate to specific functions of interest. IBM employees and customers may also order redbooks in online format on CD-ROM collections, which contain the redbooks for multiple products.

# Acknowledgements

The advisor for the Second Edition of this document was:

Eamon Murphy
International Technical Support Organization, Raleigh Center

The author of the Second Edition was:

Andrea Paravan
IBM Germany

The advisor for the First Edition was:

Philippe Beaupied
International Technical Support Organization, Raleigh Center

The authors of the First Edition were:

Paul Brisk
IBM Australia

Stephan Mueller
IBM Switzerland

This publication is the result of a residency conducted at the International Technical Support Organization, Raleigh Center.

Thanks to the following people for the invaluable advice and guidance provided in the production of this document:

John Doyle
Rodney Maxwell
William T. Nelson
Allen Springer
Barbara Walters
TCP/IP Development, Research Triangle Park, Raleigh

Rob MacGregor
Carla Sadtler
International Technical Support Organization, Raleigh Center

Mark Kressin
International Technical Support Organization, Austin Center

Scott Vetter
Michael Schwartz
International Technical Support Organization, Poughkeepsie Center

Elbert Hu

TCP/IP Development, Thomas Watson Research Center, Yorktown Heights

# Chapter 1. Introduction

The X Window System** is a portable network-based graphical windowing system that was developed at the Massachusetts Institute of Technology (MIT) in 1984. Often known simply as X, it has developed through several releases since its inception, the latest being X Version 11 Release 6. It is now accepted as the default industry standard for a windowing system and is implemented by a number of platform vendors including IBM*, DEC**, Hewlett-Packard**, Sun**, and AT&T**.

The reasons for the wide acceptance of X by the industry and the important role it is playing in the development of applications include:

- X is a network-transparent Graphical User Interface (GUI). Network transparency means that X applications can run on one host and direct their output to a display either connected to the same host, or to another host. To the user, an X workstation looks like it is connected to many different hosts at the same time.
- X applications are independent of vendor and device. Since the application only needs to communicate with the X interface, it does not need to know the details of any particular workstation display′s hardware or operating system. As long as an X application is able to establish a connection to the workstation, it can use all the capabilities of the base windowing system on that workstation. The workstation′s hardware and system software is hidden by the X protocol, which means an application running on one vendor′s platform can use another vendor′s workstation that supports X.
- As a GUI, X provides some advantages to the end user:
  - It is easy to learn
  - It is easy to use
  - It operates intuitively

## 1.1 A Brief History

The birth of the X Window System grew out of two separate projects that were being conducted at MIT in the late 1970s.

- The first was the *Argus Project* underway at the MIT Laboratory for Computer Science. Its aim was to develop a programming language that could be used in a distributed network environment. It was soon realized that one of the tools that was needed for Argus was a terminal that could display multiple windows. In the Argus environment there could be many different processes, each running on a different node in the network. The debugging process would be greatly simplified if each process could be displayed on a separate window on the one display.
- The second project was *Project Athena*, which was jointly sponsored by IBM and DEC. The aim of Project Athena was to build a computing environment composed of heterogeneous platforms, each with a consistent user interface.

It was not long before members from both project teams were discussing the applicability of available windowing software. After some initial investigation, it was discovered that some work had been done at Stanford University to develop the prototype of a window system that was simply called W.

Two of the team members, Bob Scheifler of the Argus project and Jim Gettys of Project Athena, took the W system and developed it into the X Window System. Which of course is how X derived its name. X comes after W!

The MIT X Consortium was formed in January 1988 and operated within the MIT Laboratory for Computer Science. In September 1993 the X Consortium moved out of MIT and became an independent, not-for-profit membership corporation. The continuing purpose of the X Consortium is to foster the development, evolution and maintenance of a comprehensive set of vendor-neutral, system architecture-neutral, network-transparent windowing and user interface standards.

Today the X Window System is owned by the X Consortium Inc., which has as its members those industry groups that have an interest in influencing the development of X. It is chiefly hardware and software vendors that make up the X Consortium.

## 1.2 X Concepts

The following are some concepts relevant to the X Window System.

### 1.2.1 Client/Server

The implementation of X consists of two components and this provides the independence of the applications from the devices and the hardware. They are:

- The X client
- The X server

The X server is a program that runs on a workstation that controls a display, a keyboard and a mouse. The application is called the X client. This model is probably not what you would first expect when you consider other well-known client/server relationships such as Telnet or FTP. However, it does make sense when you understand the relationship between the X application and the workstation. Remember that every time the application wants to use the screen for output, or the keyboard for input, it has to ask the X server.

The X client can make requests of any X server on the network for display, mouse, or keyboard services. The host on which the application runs need only operate with an X Window System that supports the X client service. Similarly, the workstation need only provide the X server support. It is not necessary for every platform participating in an X Window network to provide both server and client support.

Figure 1 on page 3 illustrates the components of an X Window System.

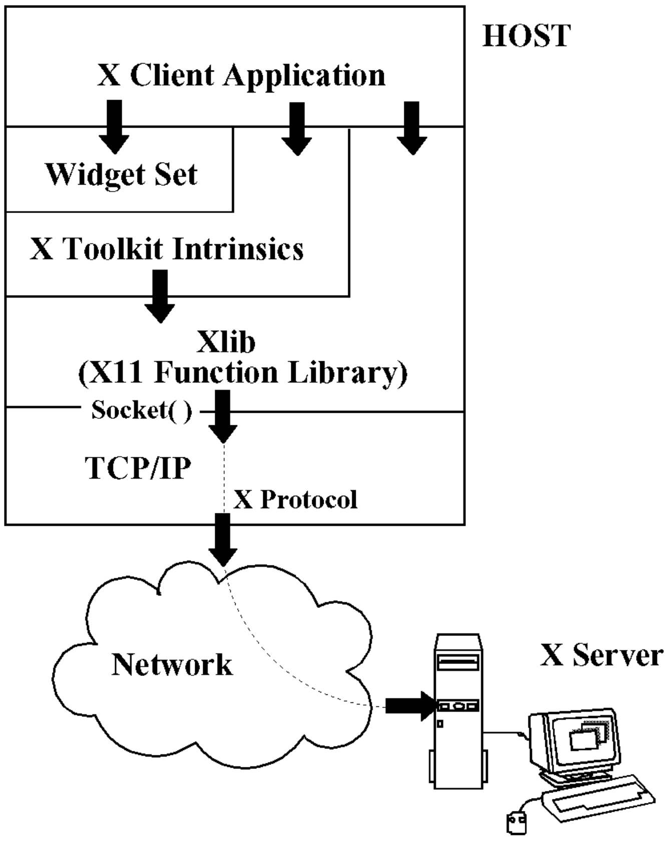


*Figure 1. The Components of an X Window System*

## 1.2.2 Client

The client is the actual application and is designed to employ a graphical user interface to display its output. It is usually written by the end user but may be included as part of the X Window System software.

In order to support an X client application a platform needs to provide the appropriate function libraries that allow the application to communicate across the network to the X server. When the application is written using the C programming language, the function library is called *Xlib*. It provides nearly 300 functions that can be called by the application to perform tasks that can include:

- Create, move, scale, stack and delete windows
- Draw lines rectangles, arcs and polygons
- Employ fonts, color maps, graphic images and cursors
- Perform a wide variety of other functions

A graphic display is not required on the client host because the X client can request this of the X server. In addition, the client application does not need to be concerned with the devices provided by the server. For example, since the X server handles the screen, the client does not need to know about the specifications of the output screen. The same application can send output to either a high resolution color screen or to a small black and white screen.

#### 1.2.2.1 X Toolkit

In theory `Xlib` is the only library that is required to run an X application. It provides the base or primitive blocks required to create the X client application. However, it is quite a complex task and may require many lines of code to produce even the simplest of application windows using Xlib alone.

In order to make X client application programming much easier, a set of high-level, pre-programmed functions known as an *X toolkit*, or Xt is provided on most X client platforms. One Xt function usually translates into several Xlib calls and there are hundreds of standard Xt functions that are available. Xt functions are also called *intrinsics*.

#### 1.2.2.2 Widgets

In creating an X client application, the programmer must create and plan to manage user interface objects that the user at the workstation will manipulate to drive the application. These objects are referred to as *widgets*.

It is appropriate to consider widgets as a set of procedures or functions that are at a level higher than those provided with the X toolkit. Widgets are comprised of `Xt` functions. Examples of widgets include:

- Push buttons
- Scroll bars
- Text boxes
- Pull-down menus
- Dialog boxes

Usually an X toolkit will include a few of the more common widgets that might be used in an X client application. However, it is more likely that the application developer will purchase a widget set separately from a vendor. Widget sets are developed by vendors as collections of pre-programmed graphics objects that are common to many X client applications. Examples of available widget sets are:

- The Athena Widget Set from MIT
- The OSF/Motif** Widget Set from the Open Software Foundation (OSF)
- The Open Look** Widget Set from AT&T

Although the widgets in these sets perform essentially the same functions, the difference between the sets is the *look and feel.*

### 1.2.3 Server

The X server is the composite partner of the X client in an X Window network. The server is the program on the workstation that controls the screen, and handles the keyboard and the mouse (or other input devices) for one or more X clients. The X server is responsible for output to the display, the mapping of colors, the loading of fonts and the keyboard mapping.

The X server workstation must provide a bit-mapped graphic display terminal which allows each individual picture element (or *pixel*) on the screen to be accessed and used to display a specific color or shade of grey. The pixels are the elements that are used to construct a graphic image such as a window on the screen.

Sometimes, the results of the application output are displayed on the system where the client is running. In this case the client and the server reside on the one platform. Often, however, you may wish to run the X client application on a remote machine that is better suited to the task, and just display the output at your local workstation. The application on the remote machine would be the client and your local workstation would be the server.

The client identifies the server by using an *X client display variable*. It has the syntax:

```
HOST:SERVER.SCREEN
```

where:

| | |
|---|---|
| *HOST* | specifies the host, or network node on which runs the X server system. For UNIX ** systems, when HOST=`unix`, the output is directed to the local console. |
| *SERVER* | specifies the server or display number (starting with 0) which represents the resources controlled by the one server program. The term server and display can be used interchangeably when discussing X Windows. A display may be composed of multiple screens, but the screens share only one keyboard and pointer. Since most workstations have only one keyboard and one pointer, they are classified as having only one display. If a host has several displays, each is assigned a number (beginning with 0) when the X server for that display is started. |
| *SCREEN* | specifies the screen number, where a screen is a physical terminal. If there is only one physical screen this is omitted. It is assumed to be 0. |

Typically X server programs run on personal computers, high performance UNIX workstations or what are called X Terminals.

#### 1.2.3.1 X Terminals

The popularity of the X Window System and its widespread usage has opened a market for a new kind of terminal - the *X Terminal*. X Terminals are typically low-cost, easy-to-use workstations which operate in a graphical, X Window environment. In particular, the X Terminal executes the X server part of the X Window system. The X client will not normally execute on an X Terminal but will run on the CPU of the host system. An X Terminal usually has the server code downloaded from the host or stored in read-only memory (ROM).

IBM has introduced a family of X Terminals called *Xstations*. The first of these was the IBM Xstation 120, which was introduced in February 1990. Since that time Xstation models 130, 140 and 150 have also been introduced.

The X server code for the IBM Xstations is distributed with IBM AIX Xstation Manager*/6000 Version 1.4. This is the latest code at the time of writing and provides X11R5 support.

For more information about Xstations refer to the appropriate product publications including *The IBM Xstation Handbook*, GG24-3695.

#### 1.2.3.2 Window Manager

The window manager as an X client is an important part of the X Window System. It is the basis for providing a graphical interface to the user that is controlled by a point and click device such as a mouse.

A window manager allows users to manipulate the application windows at the X server display. Functions supported include:

- Move windows
- Scale windows
- Move the input focus from one window to the next
- Alter the stacking of the windows
- Iconization of windows

Examples of the window managers available for use with an X Window System are:

| | |
|---|---|
| **uwm** | The universal window manager |
| **awm** | The Athena window manager |
| **twm** | Tom′s window manager |
| **olwm** | The Open Look window manager |
| **mwm** | The Motif window manager |

It should be noted that under OS/2* on the IBM PS/2* the window manager used with the X Window System is OS/2 Presentation Manager*.

#### 1.2.3.3 Display, Screen, Window, Mouse, Keyboard

| | |
|---|---|
| **Display** | Is associated with an X server program and can have one or more physical screens. A display can only have a single keyboard and a single pointing device. |
| **Screen** | Physical terminal connected to a system. |
| **Window** | A region provided by a display. It represents the connection between a client and a server and is the output area provided by the X server for the application. |
| **Mouse** | An example of a pointer device used for input and managed by the X server. Most X servers assume a three-button mouse, but you can use a two-button mouse as well. The use of the middle button is emulated by pressing both the left and the right mouse. |
| **Keyboard** | Input device managed by the X server. |

### 1.2.4 X Networking

The X server and the X client communicate using the network transparent X protocol. It is through using this protocol that the X application can be independent of the physical components and structure of the network. The application identifies and communicates with the X server using the X client display variable described previously.

Currently X Window System is supported across two network transport mechanisms:

- TCP/IP
- DECnet**

TCP/IP is certainly more common and is better suited to the concept of the X Window System because it is a widely accepted industry standard for interconnecting heterogeneous system platforms.

An X application makes calls to Xlib and it is the functions within Xlib that communicate to the X server using the X protocol. The Xlib functions use the socket interface to access the network.

Information such as keyboard input passes from the server to the client in the form of *events*. An event is a packet of information that tells the client something it needs to act on and occurs as a result of the pointer moving or a key being depressed. When a client program receives a meaningful event, it responds with a request to the server. For instance, the client may request that a window be scaled. The server would respond to the request by updating the appropriate window on the display.

### 1.2.5 Fonts

A font describes the size and shape of text characters. Normally an X server will provide several font options and each font will be referenced by a font file which is a list of indexed character patterns. When the server is requested to draw a character string, it treats each character in the string as an index to the font. For example, the ASCII 0 digit corresponds to the decimal value 48. When the workstation draws the digit 0 it draws whatever character it finds at index position 48 in the font file. Most fonts have a replacement character that the workstation draws by default when the string specifies a character that is not present in the font.

In most instances fonts are used for drawing strings of ordinary text. Almost all such fonts contain the 96 displayable characters of the ASCII set, with a space character for the replacement character. While using a font your application may need information about the sizes of individual letters and the amount of space to leave between successive lines of the text drawn. This information is stored in a font structure. One or more of these files can be found in the font directories on the X server workstation.

With X Version 11 Release 4 there is a new naming convention for fonts. The X Logical Font Description (XLFD) specifies a complete X systematic name for each font. An example of an XLFD defined font name is illustrated in Figure 2 on page 8.

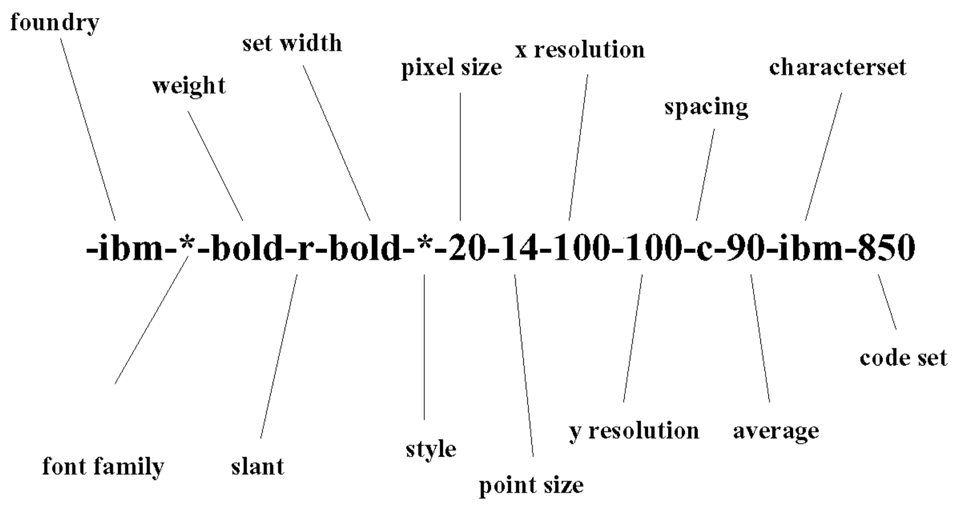


*Figure 2. An Example of the XLFD Font Description Convention*

The meaning for each font item is shown in Table 1.

*Table 1 (Page 1 of 2). XLFD Font Item Descriptions*

| **Font** | **Description** |
|---|---|
| Foundry | Specifies the type foundry or typeface design company (IBM, adobe, etc.). |
| Font family | The name of the type family. |
| Weight | Describes the blackness of the printed font. |
| Slant | Describes the style of the font (*r* for roman, *i* for italic, *o* for oblique). |
| Set width | Describes how tightly packed the characters are (*normal, narrow, expanded*). |
| Style | Identifies additional typographic style information (*serif, sans serif, informal* normally). |
| Pixel size | The font′s nominal height from the top of a capital letter to the bottom of a descender in pixels. |
| Point size | The font′s nominal height. The units are tenths of printerpoints (there are approximately 28 printerpoints per centimeter or 72 per inch). |
| Horizontal resolution | Measured in pixels per inch or dots per inch. |
| Vertical resolution | Measured in pixels per inch or dots per inch. |

*Table 1 (Page 2 of 2). XLFD Font Item Descriptions*

| Font | Description |
|---|---|
| Spacing | Describes whether a font is proportionally spaced or not (*m* for monospace, *p* for proportional, *c* for terminal fonts). |
| Average width | The unweighted arithmetic mean width in units of tenths of pixels. |
| Characterset | The X registered name for the organization that controls the character set. |
| Code set | Specifies which character set is used by the font. Most X11R4 fonts use Latin-1 (*1* for Latin 1, *850* for code page 850). |

When working with an X application, the user can employ the resources provided by the X server. The fonts file is loaded from a file on the workstation providing the display, and not from the host where the application runs. In this way the user is able to choose the font for the application output based on what is available at the X server. Prior to X11R5, all of the standard fonts provided with X were bitmap fonts. Release 5 sees the addition of font scaling and includes some new outline fonts, which are suitable for scaling. An outline font stored in a single file can be scaled to any point size you request (though the scaling requires some system overhead).

The standard Release 5 bitmap fonts should be available on your system in portable compiled font format. The font files have an extension of .pcf to indicate this format.

In addition, prior to Release 5, fonts needed to be available on the local machine or had to be provided over the network via certain protocols. In Release 5, the X Window System provides a font server (fs) from which you can request fonts resident on other machines in the network.

### 1.2.6 Colors

Since an X Window System is device independent, it is possible for the same application to use either a monochrome or a color display for output. This is achieved through the use of *colormaps* at the X server.

The number of colors that a workstation can support depends upon the number of bits in the pixel value. This is often referred to as the depth of a screen. For example, a black and white screen has a single bit pixel that is either 1 or 0. A color display may have 4, 8, 10, 12 or 24-bit pixels, although 4-bit or 8-bit pixel screens are more common. For a color display that supports an 8-bit pixel, there are 256 possible colors that could be selected at one time.

The X client application specifies a pixel value that is sent to the X server. The X server uses a *colormap* to convert the pixel value into a color on the screen. The *colormap*, also called a *color lookup table*, contains an array of *colorcells*, each with a different combination of red, green, and blue (RGB) primary color values. Each one of these combinations represents a different color or shade. The pixel value is used by the X server as an index to a *colorcell* within a *colormap*. With an 8-bit pixel, the color table can have 256 cells that can be accessed.

The X Window System defines six *visual classes*, which describe the color capability of the workstation. Before an X client application can send a pixel value to a workstation, it must understand the visual class supported by the workstation. There are Xlib functions that are called, which allow an application to determine parameters such as the colormap, the pixel value (or depth) and the visual class that are supported by the workstation.

Most workstations support just one visual class, but some allow the use of several visual classes at the same time. For example, a screen displaying multiple windows may have a different visual class for each window. The visual class tells the X client application how the workstation will use the pixel value as an index to the colormap. The visual classes are described in Table 2.

*Table 2. X Window System Visual Classes*

| **Visual Class** | **Description** |
|---|---|
| PseudoColor | In this visual class, each pixel value indexes a red-green-blue colormap. The contents of the colormap may be changed dynamically. Each pixel is treated as a single index into a single colormap array. Each entry in the colormap contains a red-green-blue triplet. |
| DirectColor | In DirectColor, the pixel is decomposed into separate bit fields for red, green, and blue. The primary fields index the colormap separately for each primary. The colormap is treated as three separate arrays containing red, green, and blue values. The red field from the pixel indexes the red colormap, the green field indexes the green colormap, and the blue field indexes the blue colormap. Direct color requires more bits per pixel than PseudoColor, but allows more colors and shades to be displayed simultaneously. |
| GrayScale | GrayScale is like PseudoColor, except the video screen displays black-and-white (or green, or amber). |
| StaticColor | StaticColor is similar to PseudoColor, except that the contents of the colormap are predefined by the workstation software and cannot be changed. |
| TrueColor | TrueColor is similar to DirectColor, except that the contents of a TrueColor map are predefined to be a linear or near-linear ramp, and cannot be changed. This visual class is commonly used in color workstations which lack color translation hardware. |
| StaticGray | StaticGray is similar to GrayScale, except that the contents of StaticGray colormaps cannot be changed. Most black and white displays have a visual class of StaticGray and a depth of queue. |

By using a workstation that supports the PseudoColor visual class it is possible to support up to 16,777,216 colors, of which 256 can be displayed at one time using only an 8-bit pixel. This is because colorcells can be used dynamically and a workstation can load new RGB primary values into colorcells quite rapidly thus providing 256 distinct levels for red, green, and blue primary colors yielding 16,777,216 combinations.

Some workstations have no color translation hardware at all. X supports these workstations with the StaticColor, TrueColor and StaticGray visual classes. As far as an X application is concerned, these workstations do have colormaps, but the colormaps are pre-loaded with red-green-blue values and cannot be changed.

The support provided for the visual classes listed above varies according to which display adapter is being used. For example, a graphics display adapter will provide more comprehensive support than a standard display adapter. For more information you should refer to the adapter′s documentation.

### 1.2.7 Keyboard

The keyboard is managed by the X server. When a key is pressed on the keyboard, the keycode that is associated with the key is translated into a character or a string on the server. This information is then passed to the X client as an event. The advantage of this design is that the client application can be device independent.

In order to understand how the application can be independent of the keyboard it is important to understand the following terms:

| | |
|---|---|
| **Keycode** | This is a number between 8 and 255 (in decimal notation). Each keycode represents a physical key on a keyboard. The keycode is fixed for each key and cannot be changed. |
| **Keysym** | A keysym is a name that represents the label of the key. Keysyms are mapped to keycodes at the server. |
| **Modifier** | A modifier is a logical keyname. A logical keyname represents functions recognized by X programs. A modifier modifies the actions of other keys which means that, when it is pressed in conjunction with another key, the two keys together represent a particular function. The simplest example is the shift modifier. The modifier is also associated to a keysym. |

The server has a file which contains all the keyboard-specific information. This file is read when the X server is started and is the input for the mapping of the keyboard. The keyboard map specifies the keysyms that are mapped to each keycode.

As an example, assume we have a very simple keyboard with two keys, one with the label T, the other with the label Shift. The T key has the keycode 40 and the Shift key has the keycode 50. In the server keyboard map, keycode 40 is associated with the keysym t while keycode 50 is associated with the keysym shift. In addition the keysym shift is associated with the modifier shift. When we press the T key X interprets this as the t character, and when we press Shift with T, X interprets this as the T character.

## 1.3 IBM X Window System Implementations

Below are the IBM X Window System implementations that we will be working with in this publication.

### 1.3.1 MVS

The IBM product TCP/IP Version 2.2.1 for MVS (Program Number 5735-HAL) provides an X Window System client service. This support is automatically shipped with the product. This means that MVS can support an application, known as the X client, that can communicate with a display provided by an X server. The X client application runs in a TSO user address space and communicates with the X server using the X protocol across a TCP/IP network.

The X server provides access to resources such as a screen, a keyboard, a mouse, fonts, and graphics. It accepts requests from the X client application and sends user input back to the X client. Each X client can make requests of multiple X servers and each X server can service multiple X clients. Examples of X server platforms that would interoperate with an MVS X client application might be OS/2, AIX* or DOS.

TCP/IP Version 2.2.1 for MVS provides two methods for supporting X client applications under MVS:

1. X Window System GDDM* support
2. X Window System API for user-written applications

### 1.3.1.1 MVS X Window System GDDM Support

The X Window System GDDM interface supports graphics display output from the IBM Graphical Data Display Manager (GDDM) to be displayed at an X server station.

When the X Window System GDDM interface is activated it translates the data stream created by GDDM into the X protocol and transmits it to the X server for display.

### 1.3.1.2 MVS X Window System API

It is possible to write an X client application and run it under MVS. The MVS X Window System provides an application program interface (API) allowing an application to make calls to the API to create the X protocol to have output displayed on an X server. When writing this application, the user need only be concerned with the X client API and the syntax of the API calls.

Application programs using the X Window System must be written in C and require the following corequisite products:

- IBM C for System/370*, Compiler Licensed Program (Program Number 5688-187)
- IBM C for System/370, Library Licensed Program (Program Number 5688-188)

This environment under MVS is illustrated in Figure 3 on page 13.

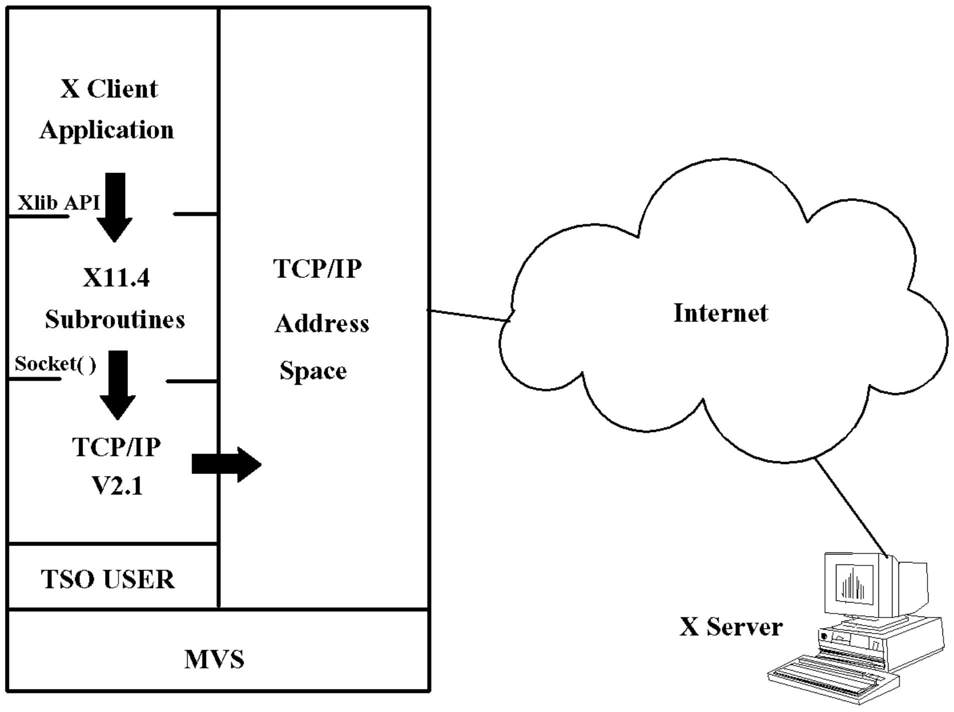


*Figure 3. MVS X Window System API*

The X Window System under MVS is equivalent to the MIT-defined X Window System Version 11 Release 4. It provides:

- SEZAX11L which contains the Version 11 X Window System subroutines:
  - *X Window System subroutines*.
  - *Extension routines* which permit the creation of extensions to the core `Xlib` functions with the same performance characteristics.
  - *MIT extensions to the X Window System*. The extensions provided by the AIX X Window System are not supported by the MVS X Window System API. The following MIT extensions are supported:
    - SHAPE
    - MITMISC
    - MULTIBUF
  - *Associate Table functions* which support the association of user data structures with X Window resources.
  - *MIT X Miscellaneous Utility Routines* that are a common set of functions that have in the past proved useful to application writers.
- SEZAOLDX, which provides Version 10 X Window System-compatible routines.
- XTLIB (the X Toolkit intrinsics library).

- XAWLIB (the Athena Widget set library).
- XMLIB (the OSF/Motif Release 1.1-based widget set library).
- Header files required for compiling X client applications.
- Header files required for compiling X client applications using the OSF/Motif-based widget set.
- Standard MIT X client applications.
- Sample X client applications (XSAMP1, XSAMP2, XSAMP3).

The application calls `XOpenDisplay()` to start communication with an X server on a workstation. This invokes the Xlib code to open a communication path called a socket to the X server and send the appropriate X protocol to initiate the client/server communication.

The X protocol used between the Xlib code and an X server uses an ISO Latin-1 encoding for character strings, while an application running under MVS will use EBCDIC. The Xlib code will automatically translate character strings to and from EBCDIC and ISO Latin-1 as required using internal translate tables.

Under MVS, external names must be eight characters or less. Under the X Window System, the X client API supports names of routines and data sets that often exceed this limit. In order to support the X client API under MVS, all X client application names longer than eight characters are remapped to unique names using the C compiler preprocessor. This name remapping is found in a file called X11GLUE.H, which is automatically included in a user-written X client application when the standard X header file called XLIB.H is included.

#### 1.3.1.3 MVS X Window System Toolkit

IBM provides an X toolkit with the X Window System for MVS. The purpose of the toolkit is to simplify the design and reduce the time to code X client applications by providing a set of common user interface functions often called intrinsics. These functions are composed of many of the functions that are to be found in `Xlib`. By using the toolkit intrinsics, the application programmer will not need to go down to the low level of coding the `Xlib` routines to define the mechanisms for handling the interaction between the X server and the X client.

For a full list of the intrinsics supplied with the X toolkit for MVS please refer to the *TCP/IP Version 2.2.1 for MVS: Programmer's Reference*.

#### 1.3.1.4 MVS X Window System Widget Sets

IBM provides two widget sets with the X Window System for MVS. These are:

- The MIT-developed widget set commonly referred to as the Athena widget set.
- The OSF/Motif widget set.

The widgets in each of these sets are composed of the intrinsics in the X toolkit provided with the MVS X Window System. These widgets provide the programmer with many pre-programmed objects that are commonly used in X client applications. They greatly reduce the amount of time and effort required to create a client application under MVS.

For a full list of the widgets supplied with the X Window System for MVS please refer to the *TCP/IP Version 2 for MVS: Programmer's Reference*.

#### 1.3.1.5 Creating an X Client Application under MVS

In order to write an application that uses the X Window System protocol under MVS, it is necessary to understand the X client API. To help you gain a better understanding of this API, IBM provides the source and executable code of three sample X client application programs, XSAMP1, XSAMP2, and XSAMP3 with TCP/IP for MVS. In addition, IBM also provides some standard X client applications developed by MIT. These provide the best way to experiment with and learn about the X client API under MVS. They are discussed in greater detail in 3.1, “Under MVS” on page 59.

### 1.3.2 VM

The X Window System provided by the IBM product TCP/IP Version 2.2 for VM (Program Number 5735-FAL) is essentially the same as that provided under MVS. It provides an X Window System client service that is automatically shipped with the product. In the same way as MVS, VM can support an application, known as the X client, that can communicate with a display provided by an X server. The X client application runs in a CMS virtual machine and communicates with the X server using the X protocol across a TCP/IP network.

Examples of X server platforms that would interoperate with a VM X client application might be OS/2, AIX or DOS.

As with MVS, TCP/IP Version 2.2 for VM provides two methods for supporting X client applications:

1. X Window System GDDM support
2. X Window System API for user-written applications

#### 1.3.2.1 VM X Window System GDDM Support

The X Window System GDDM interface supports graphics display output from the IBM Graphical Data Display Manager (GDDM) to be displayed at an X server station.

When activated, the X Window System GDDM interface translates the data stream created by GDDM into the X protocol and transmits it to the X server for display.

#### 1.3.2.2 VM X Window System API

It is possible to write an X client application and run it in a VM/CMS environment. The VM X Window System provides an application program interface (API) allowing an application to make calls to the API to create the X protocol to have output displayed on an X server. When writing this application, the user need only be concerned with the X client API and the syntax of the API calls.

Application programs written using the X Window System must be written in C and require the following corequisites:

- IBM C for System/370, Compiler Licensed Program (Program Number 5688-187)
- IBM C for System/370, Library Licensed Program (Program Number 5688-188)

This environment under VM/CMS is illustrated in Figure 4 on page 16.

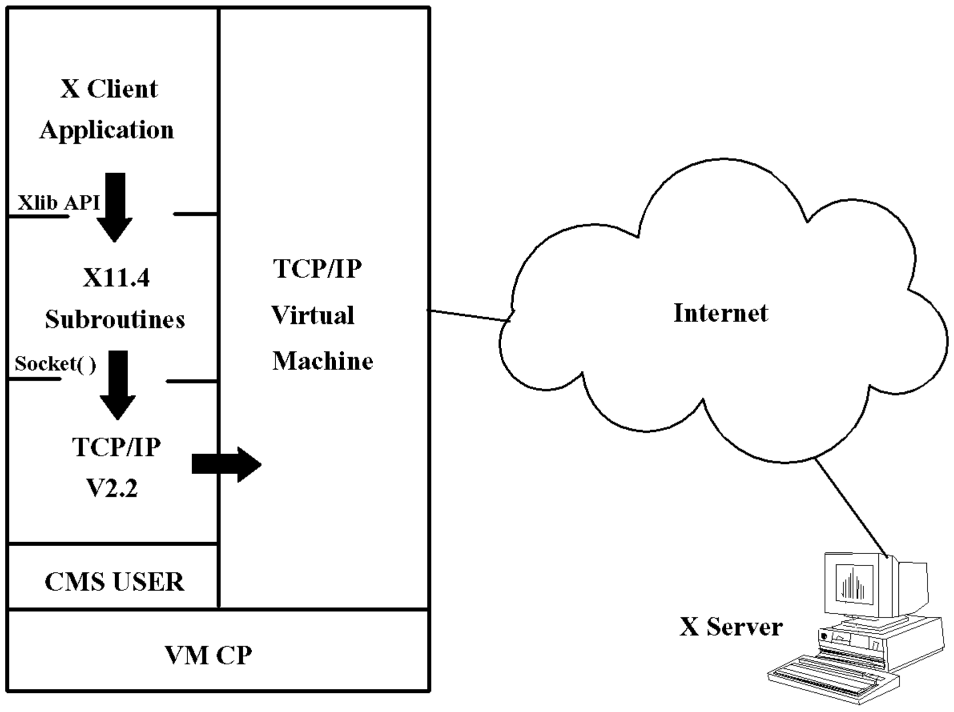


*Figure 4. VM X Window System API*

The X Window System under VM is is equivalent to the MIT-defined X Window System Version 11 Release 4. It provides:

- X11LIB TXTLIB, which contains the Version 11 X Window System subroutines:
  - *X Window System Subroutines*.
  - *Extension Routines*, which permit the creation of extensions to the core Xlib functions with the same performance characteristics.
  - *MIT Extensions to the X Window System*. As with MVS, the AIX extensions are not supported by the X Window System API under VM. The following MIT extensions are supported:
    - SHAPE
    - MITMISC
    - MULTIBUF
  - *Associate Table Functions*, which support the association of user data structures with X Window resources.
  - *Miscellaneous Utility Routines*, which are a common set of functions that have in the past proved useful to application writers.
- OLDXLIB TXTLIB, which provides Version 10 X Window System compatible routines.
- XTLIB TXTLIB (X Toolkit intrinsics library).
- XAWLIB TXTLIB (Athena Widget set library).

- XMLIB TXTLIB (OSF/Motif-based widget set library).
- Header files required for compiling X client applications.
- Header files required for compiling X client applications using the OSF/Motif-based widget set.
- Standard MIT X client applications.
- Sample X client applications.

The application calls XOpenDisplay() to start communication with an X server on a workstation. This invokes the Xlib code to open a communication path called a socket to the X server and send the appropriate X protocol to initiate the client-server communication.

The X protocol used between the Xlib code and an X server uses an ISO Latin-1 encoding for character strings, while an application running under VM/CMS will use EBCDIC. The Xlib code will automatically translate character strings to and from EBCDIC and ISO Latin-1 as required using internal translate tables.

Under VM/CMS, external names must be eight characters or less. Under the X Window System, the X client API supports names of routines and data sets that often exceed this limit. In order to support the X client API under VM/CMS all X client application names longer than eight characters are remapped to unique names using the C compiler preprocessor. This name remapping is found in a file called X11GLUE H, which is automatically included in a user-written X client application when the standard X header file called XLIB H is included.

#### 1.3.2.3 VM X Window System Toolkit

IBM provides an X toolkit with the X Window System for VM. It is very similar in function and purpose to that provided with MVS. Please refer to 1.3.1.3, “MVS X Window System Toolkit” on page 14 for a description.

For a full list of the intrinsics supplied with the X toolkit for VM please refer to the *TCP/IP Version 2 Release 2 for VM: Programmer's Reference*.

#### 1.3.2.4 VM X Window System Widget Sets

As with MVS, IBM provides two widget sets with the X Window System for VM. They are:

- The MIT-developed widget set commonly referred to as the Athena widget set.
- The OSF/Motif widget set.

The widgets in each of these sets are composed of the intrinsics in the X toolkit provided with the VM X Window System. These widgets provide the programmer with many pre-programmed objects that are commonly used in X client applications. They greatly reduce the amount of time and effort required to create a client application under VM.

For a full list of the widgets supplied with the X Window System for VM please refer to the *TCP/IP Version 2 Release 2 for VM: Programmer's Reference*.

#### 1.3.2.5 Creating an X Client Application under VM

In order to write an application that uses the X Window System protocol under VM/CMS, it is necessary to understand the X client API. To help you gain a better understanding of this API, IBM provides the source and object code of three sample X client application programs, XSAMP1, XSAMP2, and XSAMP3 with TCP/IP for VM. In addition, IBM also provides some standard X client applications developed by MIT. These provide the best way to experiment with and learn about the X client API under VM. They are discussed in greater detail in 3.2, "Under VM" on page 80.

### 1.3.3 AIX/6000

The IBM X Window System Implementation for the IBM RISC System/6000* is called IBM AIXwindows Environment/6000* Version 1.2.5. This version was announced in October 1993. The program number is 5601-257. AIXwindows Environment/6000 provides both the client and the server part of the X Window System. Included with AIXwindows Environment/6000 is the X server, programming libraries, and a number of client applications that are supplied as either executable modules or as source code.

#### 1.3.3.1 What is AIXwindows

- IBM′s implementation of X11 release 5 plus:

  AIXwindows consists of several parts: the basic runtime code and utilities are in X11rte.obj, files needed to develop and compile X clients are in X11dev, and there are various font components. Some additional utilities and toys are in X11ext (xdt, for example).

  – Display PostScript

    For each display adapter supported there is a specific driver. All the native display adapters on the RS/6000 support Display PostScript Xstations and PCs that are used as X terminals do not support Display PostScript.

    There are only a few applications that make use of Display PostScript, for example, FrameMaker for imbedded PostScript graphics.

  – Motif window manager and libraries from the OSF

    The Motif window manager and the motif libraries provide a look and feel that is similar to MS-Windows and Presentation Manager. The three dimensional look of the Motif buttons was developed by HP. After the introduction of the pseudo 3D look of Motif, MS-Windows and Presentation Manager quickly changed to provide the same visual feedback.

  – X.desktop** from IXI

    The X.desktop product from IXI is shipped as AIXdesktop with the AIXwindows extensions (X11ext). It provides a simple desktop interface to AIX. It is an add-on product and is not easily integrated. To really use it one needs to heavily customize it.

    Note that a replacement for AIXwindows called Common Desktop Environment will be provided in AIX V4.1. This operating system version was not available at the time of this residency.

  – Additional utilities

AIXwindows comes with some additional utilities that let you edit fonts (fontutil), change application resources interactively (custom), a menu utility for the included demos (demodr), an automatic screen locker (xss) and much more.

– Additional fonts

  In addition to the standard MIT fonts in the subdirectories of /usr/lib/X11/fonts there are a lot of fonts in /usr/lib/X11/fonts itself that are provided as IBM add-ons, including fonts that conform to the ergonomic standard ISO 9241.

  All IBM fonts are supplied in ISO code pages and the PC850 code page. The machine uses PC850 fonts by default. This is fine when one wants to integrate with PCs, but for a UNIX environment one should use the ISO code pages as these are commonly used in the UNIX world.

– AIXwindows provides only 2D support

  In addition to AIXwindows there is an AIXwindows 3D package that provides libraries with advanced graphics features (PEX, graPhigs, GL, SoftgraPhigs, and openGL). The 3D package is not included in AIXwindows and must be purchased separately.

#### 1.3.3.2 The Components of X

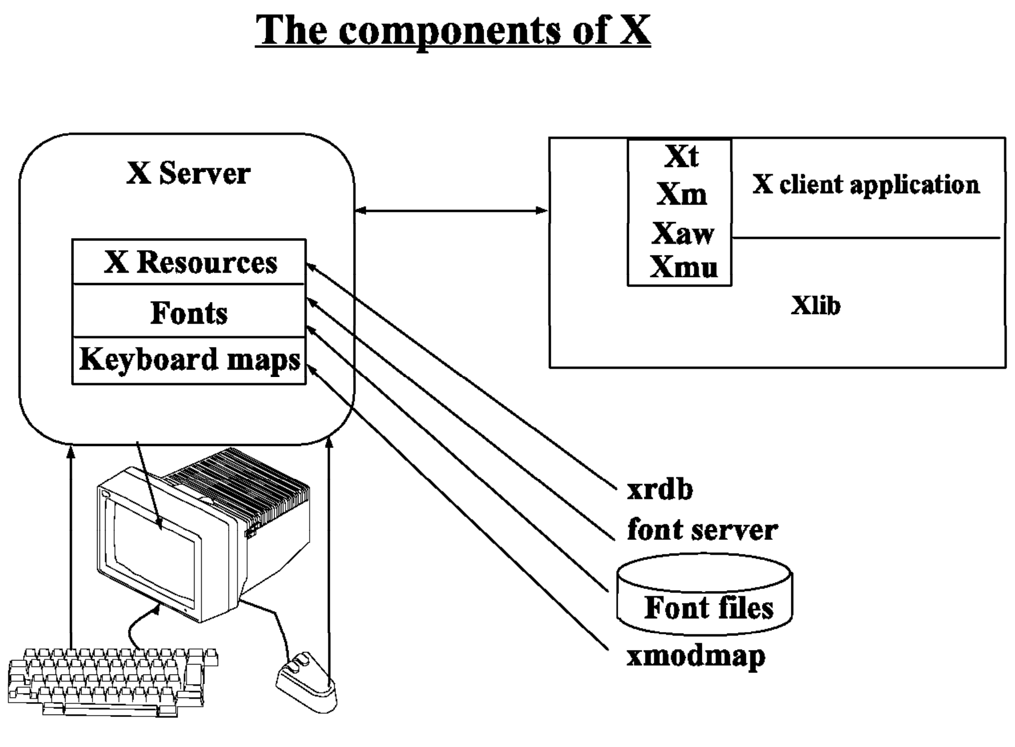


*Figure 5. The Components of X*

The X server receives input events from the keyboard, mouse, tablet or other devices. It processes those events and transfers them to the application via the X protocol. The application gets the events through the X library (Xlib).

An application accesses the X server by using either Xlib directly or by using one of the higher level libraries such as Xt (X toolkit), Xm (Motif), Xaw (Athena widget set) or others.

No matter what high-level libraries are used, all requests to the X server are then sent to the server via low-level primitives of the X library. The server then reacts to the requests and updates the screen.

The server maintains several types of resources internally. For the average end user, the fonts, application resources and keyboard map are the interesting ones. They can be modified by special X applications:

| | |
|---|---|
| setroot | Modifies the properties of the root window, for example, nice backgrounds and fancy cursor shapes |
| xrdb | Modifies the X resource database (application resources) in the server. |
| xset | Modifies properties of the X server such as the font path, screen saver time-out, and bell volume |
| font server | Optionally provides the fonts for the X server |
| xmodmap | Modifies the server′s keyboard map so that ILS keyboards can be supported. |

Display PostScript and other extensions to the X server use extensions of the X protocol to communicate with the server. Use xdpyinfo to find out what extensions are supported by a running X server. The xdpyinfo command will also tell you about other server characteristics. Use xset q to find out even more.

#### 1.3.3.3 AIXwindows Environment/6000 3D Feature

The features listed previously are part of the AIXwindows Environment/6000 and they satisfy the requirements of the user who needs a two dimensional GUI. When there is a requirement for 3D capability you can order the AIXwindows Environment/6000 3D feature.

The use of the 3D feature is not covered in this document. Three dimensional GUI support is a non-standard enhancement.

AIX Windows/3D includes:

- Graphics Library (GL**) API with DBCS capability
- Personal graPHIGS*
- Graphical Kernel System Compatibility Option (GKS-CO)
- PEX, client support (full API) and server support

### 1.3.4 OS/2

TCP/IP Version 2.0 for OS/2 (Program Number 5622-086) provides an X Window System server and an X Window System client. The support provided is equivalent to Version 11 Release 5 (X11R5) of the X Window System server as defined by MIT.

The X server and client code are not shipped automatically with TCP/IP for OS/2 but must be ordered as a separate features.

### 1.3.4.1 X Window System Server

The OS/2 X server enables X client application programs to display output in an OS/2 Presentation Manager window. The X client applications can reside on one or more system platforms that support the X client function. They communicate with the OS/2 X server across a TCP/IP network. Examples of IBM platforms that can support X client applications and can communicate with an OS/2 X server include MVS, VM, OS/2 and AIX.

The OS/2 Presentation Manager is used by the OS/2 X server function as the X Window manager. It is often referred to as the PMX server. This means that all of the keyboard, display, and pointer devices that are currently supported by the OS/2 Presentation Manager are available to the X server window manager.

The OS/2 X Window System server support is composed of the following components:

- *OS/2 X server code (PMX.EXE):* This code uses and accesses the following key files that are found by default in the TCPIP\X11 subdirectory (with the exception of the X0HOSTS file as listed below):
  - The X client host authorization file (X0HOSTS). This is used to identify the X client hosts on the network that are authorized to connect to the OS/2 X server. This file must be created by the X server administrator.

    **Note:** This file must be stored in the TCPIP\ETC subdirectory.
  - The color database (RGB.TXT). This file maps a large number of color names onto some RGB (red-green-blue) values. It is possible to edit this file to either modify the mappings or to create new available colors.

    PMX uses information from the OS/2 Presentation Manager to map the RGB values to indexes in the Presentation Manager color tables.
  - X Font files.
- *X Font Support:* Includes fonts supplied by the IBM AIX X Windows product as well as fonts from the MIT X Consortium X11R5 distribution. It is also possible to import the source of fonts from other systems that are not provided with the OS/2 X Window System and compile them for use by X client applications.
- *National Language Support* for keyboards.

There are a series of programs that are included in X Window System server to assist with administering and customizing PMX. An example is XMODMAP, which supports the displaying and modification of the PMX keyboard modifier map and keysym table.

### 1.3.4.2 X Window System Client and OSF/Motif Kits

In OS/2, the X Window System client support consists of an application program interface (API) that creates the X program. This API lets you create an application that uses the TCP/IP sockets system functions to communicate with an X Window System server. As an application writer, you need to be concerned only with the client API in writing your application.

The communication path from the OS/2 X Window System application to the server involves the client code, the X Window System library, and the TCP/IP library. The application program that you create is the client part of a client-server relationship. The X server provides access to the resources that are shared among many X applications, such as the screen, keyboard, mouse, fonts, and graphics contexts.

***X Window System Client Kit:*** The X Window System client has two purposes:

- To develop X Window applications to run on an OS/2 system.
- To run X Window applications on an OS/2 system.

In the X Window System, the notion of client and server is somewhat reversed. Normally you think of yourself, or the program you are running on your PC, as a client, which accesses shared resources such as disk storage or printers on a server in another location.

However, with the X Window System, the resource that you share is the display, which is attached to your workstation. Therefore, your workstation must function as a server in order to share the display among several X client applications.

TCP/IP Version 2.0 for OS/2 is the first version of this product that includes the X Window System client. In previous versions, you were able to have your TCP/IP for OS/2 workstation function as an X server. This meant that you could execute an X client application on a remote system such as an RS/6000, and the presentation information was displayed on your own OS/2 workstation using X protocol and the X server functions.

The X Window System client provides you with the ability to develop X Window applications for the OS/2 platform and then also run them on the same platform. Your workstation is now equipped to function as both an X Window client and server.

The X Window System client includes two parts. The X Window System client kit contains the executable sample X client applications, and can be installed on either a FAT or an HPFS file system. The X Window System Client programmer′s toolkit contains all the libraries, the header files, and the source files for the sample applications, that you need to create your own X client applications. The programmer′s toolkit must be installed on an HPFS file system.

**Note:** If you want to develop or execute an OSF/Motif application, you need the OSF/Motif kit.

***X Window Structure:*** The X Window System client kit includes the following layers shown in Figure 6 on page 23.

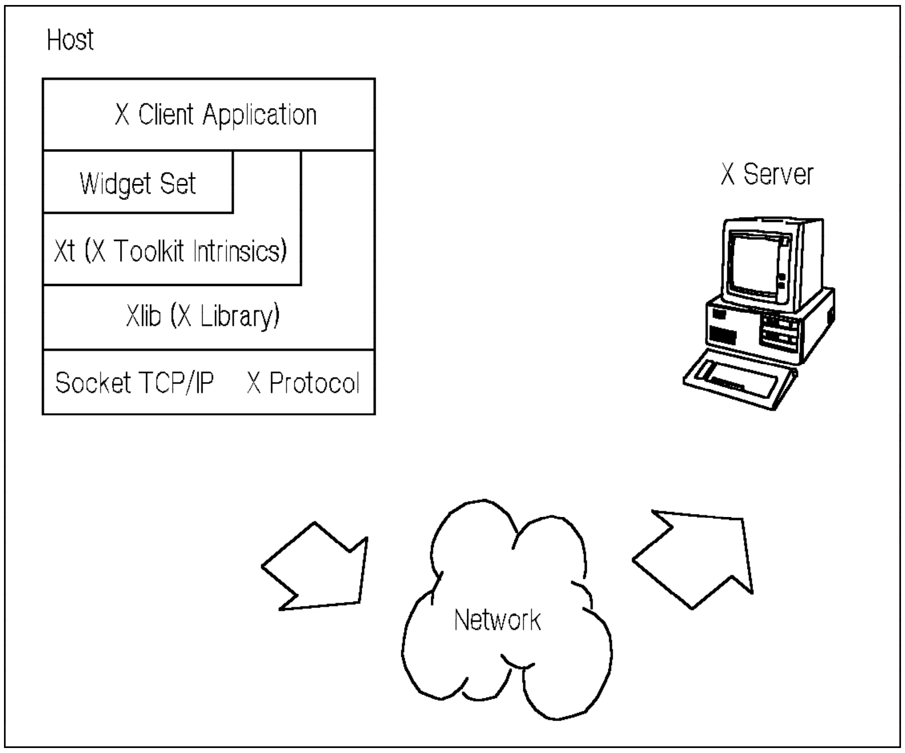


*Figure 6. X Window Application Layers*

*The X Library (Xlib):* The X Window System client contains the X library (Xlib), a set of low-level application functions that provide access to, and control of, the display, its windows, and the input devices. Xlib is the fundamental layer that supports the intrinsics and widgets that are included in the X Window System client.

*The X Toolkit (Xt) Intrinsics Library:* The X Window System client contains the X toolkit library (Xt) intrinsics library, on the top of the X library, which allows you to simplify the design of applications by providing an underlying set of common user interface functions. Xt provides an improved approach to GUI programming. It creates a general mechanism for producing reusable user-interface components, and provides routines for creating and using user-interface components called widgets. For more information on the X toolkit library read the *X Window Client System Guide*.

*The Widget Sets:* The X Window System client provides you with the Athena widget set.

You can obtain the OSF/Motif widget set that is separately available in the TCP/IP for OS/2 OSF/Motif kit.

A widget set is a collection of separate widgets. The widget is the fundamental data type in the X Window System client. Widgets generally provide a user interface component, such as, scroll bar, a text-entry field, or a menu. Widgets are allocated dynamically and contain state information. Each widget belongs to a widget class that is allocated statically and initialized. The widget class contains the operations allowed on widgets of that class.

*OSF/Motif Widget Set:* You can separately order the TCP/IP for OS/2 OSF/Motif kit. It contains the OSF/Motif widget set, which implements user interface components, such as scroll bars, menus, and buttons. You can combine the OSF/Motif widget set with the Xt Intrinsics and Xlib to construct a Motif application. For more information about the OSF/Motif widget set, refer to *X Window System client Guide.*

## 1.3.5 DOS

IBM does not currently provide an X Window System product for DOS. In order to investigate X Windows on the DOS platform we used the product HCL-eXceed** from Hummingbird Communications Ltd. (HCL) of Markham, Ontario, Canada.

HCL has two flavors of HCL-eXceed available:

1. HCL-eXceed/DOS V3.3, which runs under DOS and provides its own X Window manager called hwm. However, you can start your own X Window manager such as the Motif window manager.
2. HCL-eXceed/W V3.3.3, which runs under Microsoft Windows 3 for DOS and uses Microsoft Windows as the X Window manager.

For the purpose of this book we worked mainly with HCL-eXceed/W V3.3.3 and used IBM′s TCP/IP Version 2.1.1 for DOS (Program Number 5621-219) under DOS Version 6.1. We used Microsoft Windows Version 3.1.

HCL-eXceed/W provides an X Window server for the DOS environment. The support provided is equivalent to Version 11 Release 5 (X11R5) of the X Window System server as defined by MIT. It enables X client application programs to display output in a Microsoft Windows for DOS window. The X client applications can reside on one or more system platforms that support the X client function. They communicate with the HCL-eXceed server across a TCP/IP network. Examples of IBM platforms that can support X client applications and can communicate with an HCL-eXceed server include MVS, VM, OS/2 and AIX.

The HCL-eXceed server provides the following features (for further details, please refer to the *HCL-eXeed/W for Microsoft Windows User′s Guide* available with the product):

- Use of Microsoft Windows for DOS as the X Window Manager. This allows the user to take advantage of the functions provided by Microsoft Windows such as cut and paste between windows, and support for either a two- or three-button mouse.
- An interactive configuration utility for customizing such things as server and communication options, colors, fonts and keyboard mapping.
- An automatic client starter that supports the creation of icons for regularly accessed client applications. The user can click on the icon to open a window that will use REXEC or RSH to start up the X client application on a remote host. Of course, the host must support REXEC or RSH.
- A font compiler that allows the user to import fonts that are of the standard .BDF or .PCF format and add them to the library of fonts available to HCL-eXceed (.FON format).

# Chapter 2. Installation

## 2.1 ITSO Network Configuration

For the purpose of this document we worked on a number of X Window System-capable platforms. These systems are installed at the International Technical Support Organization (ITSO) Center at Raleigh. The network configuration with each of the X Window System platforms that we used is illustrated in Figure 7.

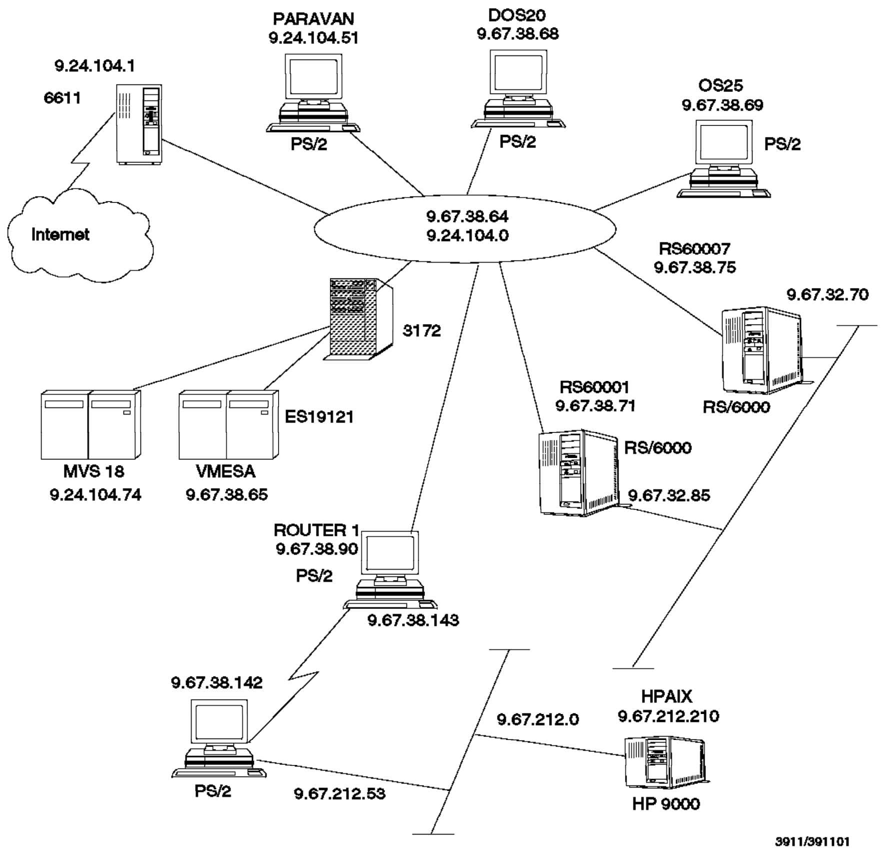


*Figure 7. Component of the Network Configuration at the ITSO Raleigh*

## 2.2 MVS

The following steps are a guide to installing the X Window System under MVS. The description is divided into two parts as follows:

1. Installation Verification for the X Window System API
2. Installing the X Window System GDDM Interface

It is assumed that the base TCP/IP product has been installed on your MVS system.

## 2.2.1 Installation Verification for the MVS X Window System API

The X Window System is installed with the other target libraries when TCP/IP is installed under MVS. All that needs to be done is to verify that the API is ready to accept calls from an X client application.

IBM provides three sample programs with TCP/IP for MVS. They are called XSAMP1, XSAMP2 and XSAMP3. These C programs have already been compiled and link-edited and can be found in *tcpip*.SEZALINK, where *tcpip* is the high-level data set name qualifier under which the TCP/IP libraries are installed.

It may be that the TCP/IP for MVS on your system is accessing a common system LINKLIB. If this is the case, you can check in one of two places to determine which LINKLIB TCP/IP is accessing:

1. Check the STEPLIB DD statement in the TCP/IP startup procedure.
2. Look in SYS1.PARMLIB(LNKLSTxx) for a clue as to which is the authorized LINKLIB.

In order to verify the installation of the API you must do the following:

1. Ensure that the X server has authorized the MVS host as an X client. For specific details on how to do this please refer to 4.1.6, “Controlling X Client Access to AIX/6000” on page 125 when AIX is the X server, 4.2.4, “Controlling X Client Access to OS/2” on page 132 when OS/2 is the X server, and 4.3.1.4, “Controlling X Client Access” on page 153 for a DOS X server.
2. From the TSO command prompt type:

   ```
   XSAMPn -display <X client display variable>
   ```

   where n=2 or 3 and *client display variable* is the variable that the X client must use to access the X server. It has the format host:0.0 where host is the Internet address of the X server and 0.0 represents the target server.server screen. For example, when using our OS/2 machine as the X server, since it had the Internet address 9.24.104.51, we typed the command:

   ```
   XSAMP2 -display 9.24.104.51:0.0
   ```

   Also try this for XSAMP3. Note that in the case of XSAMP1, the display variable is not supported so you will need to preset the variable as described in 3. below. You may then load XSAMP1.

   The displays you will see at an OS/2 X server are illustrated in Figure 8 on page 27 for XSAMP1, Figure 9 on page 27 for XSAMP2, and Figure 10 on page 27 for XSAMP3.

   XSAMP1 opens a display at the X server for 60 seconds and then closes it. The best way to end either XSAMP2 or XSAMP3 is to close the window at the X server.

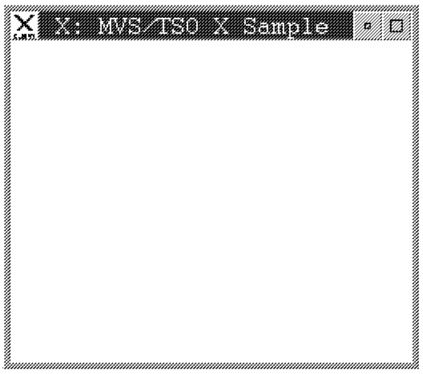


Figure 8. Display at an OS/2 X Server for XSAMP1 Sample X Client Program

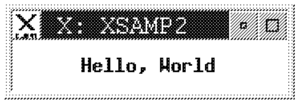


Figure 9. Display at an OS/2 X Server for XSAMP2 Sample X Client Program

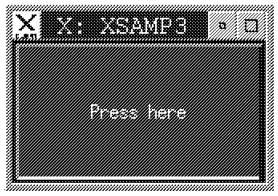


Figure 10. Display at an OS/2 X Server for XSAMP3 Sample X Client Program

3. Once you have verified the API using XSAMP1, XSAMP2 and XSAMP3 you can set up an X client display variable data set. This means that you will not need to type in the X client display variable each time you invoke the X client program. Allocate a data set called *userid*.XWINDOWS.DISPLAY where *userid* is the TSO user ID under which the X client application will be started. This data set should be sequential, fixed block with a record length of 80 bytes. The contents of this data set are illustrated in Figure 11.

```
9.24.104.51:0.0
```

Figure 11. Example of the Contents of userid.XWINDOWS.DISPLAY Data Set

   Look out for the following in *userid*.XWINDOWS.DISPLAY: when you add the entry to this dataset using the ISPF editor, you may end up with numbers in the far right columns of the data set. This will result in the error message `Error: Can't Open Display`. You must correct this by deleting the characters to the right of the X client display variable if your ISPF editor profile is set with *numbers off* or by using the UNNUM command if your ISPF profile is set with *numbers on*.

## 2.2.2 Installing the MVS X Window System GDDM Interface

The X Window System GDDM interface must be installed and activated in order for a GDDM application to send its output to an X server for display using the X protocol. The executable code for the X Window System GDDM interface must be in *tcpip*.SEZALINK, where *tcpip* is the high-level data set name qualifier under which all of the TCP/IP libraries are installed.

As discussed previously, you can check in one of two places to determine which LINKLIB TCP/IP for MVS on your system is accessing:

1. Check the STEPLIB DD statement in the TCP/IP startup procedure.
2. Look in SYS1.PARMLIB(LNKLSTxx) for a clue as to which is the authorized LINKLIB.

### 2.2.2.1 Installation Steps

The following steps provide a guide to installing the X Window System GDDM interface under MVS:

1. Allocate a partitioned data set for your user CLISTs. You could call the data set *userid*.tcpip.CLIST, where *userid* is your TSO user ID. This data set name will be assumed for the following discussion.
2. Copy the members INSTGDXD and GDDMXD from *tcpip*.SEZAINST into *userid*.TCPIP.CLIST. You do not need to modify these CLISTs at all.
3. Ensure that the following members are in *tcpip*.SEZALINK:
   - GDXLIOX0
   - EZAADMLR
   - KEYCODE
   - GXDEMO1
   - GXDEMO2
   - GXDEMO3
   - GXDEMO4
   - GXDEMO4A
   - GXDEMO5
   - GXDEMO6
4. The X Window System GDDM interface is installed by invoking the CLIST INSTGDXD. Enter the following command from the TSO command prompt:

   ```
   EXEC TCPIP(INSTGDXD) 'GDXDLOAD(tcpip.SEZALINK),
   GDXIN(tcpip.SEZACMTX), GDDMLOAD(gddm.GDDMLOAD),
   GDDMLIB(gddm.GDDMLIB)'
   ```

   where:

   | | |
   |---|---|
   | *tcpip*.SEZALINK | is the full name of the SEZALINK that TCP/IP is accessing on your sytem. |
   | *gddm*.GDDMLOAD | is the full name of GDDMLOAD library on your system. |
   | *gddm*.GDDMLIB | is the full name of GDDMLIB library on your system. |

   The CLIST INSTGDXD performs the following:

   a. Link-edits the module EZAADMLR and replaces it back into *tcpip*.SEZALINK.

   b. Link-edits the demonstration program load modules GXDEMO1 through GXDEMO6 and replaces them back into *tcpip.*SEZALINK.

5. Ensure that INSTGDXD runs with a return code of 0.

A point to note is that the blocksize of *gddm*.GDDMLOAD and *gddm*.GDDMLIB needs to be greater than or equal to the blocksize for *tcpip*.LINKLIB. If this is not the case then INSTGDXD will run with a severity code of 4.

#### 2.2.2.2 Installation Verification

Use the following tasks as a guide to testing the installation of the X Window System GDDM interface.

1. Each time you want to activate the X Window System GDDM interface you must establish a STEPLIB that includes the following data sets:

| | |
|---|---|
| *tcpip.*SEZALINK | is the SEZALINK which contains the executable X Window System GDDM interface code. |
| *pli.*SIBMLINK | is the full name of the PLI common library on your system. |
| *c370.*SEDCLINK | is the full name of the C/370 runtime library on your system. |
| *gddm.*gddmload | is the full name of the GDDM load library on your system. |

   If you have those libraries already defined in your link library you don′t need to access them via a steplib.

   We found the easiest way to establish this steplib was to create a CLIST and then execute it each time we wanted to activate the X Window system GDDM interface. We called the CLIST GDDMX and placed it in *userid.*TCPIP.CLIST. An example of this CLIST is shown in Figure 12. You invoke this CLIST from the TSO command by typing:

```
EXEC BIN(GDDMX)
```

```
ISPEXEC LIBDEF ISPLLIB DATASET                   +
               ID(′SYS1.TCPIP.V2R2M1.SEZALINK′   +
                  ′GDDM.V2R3.GDDMLOAD′           +
                  ′PLI.V2R3M0.SIBMLINK′          +
                  ′C370.V2R1M0.SEDCLINK′) UNCOND
ALLOC F(ADMSYMBL) DA(′GDDM.V2R3.GDDMSYM′) SH
```

*Figure 12. Example of the STEPLIB CLIST for the X Window System GDDM Interface*

2. Activate the X Window System GDDM interface by executing the CLIST GDDMXD that you copied in to *userid.*TCPIP.CLIST by typing the following at the TSO command prompt:

```
EXEC TCPIP(GDDMXD)
```

   You will be prompted for a positional parameter to which you need to enter:

```
ON
```

You should see the message `GDDMXD/MVS ACTIVE`.

3. IBM has provided some sample programs that allow you to test the X Window System GDDM interface under MVS.

   In order to run these demonstration programs you need to do the following:

   a. Ensure that the X server has authorized the MVS host as an X client. For specific details on how to do this please refer to 4.1.6, "Controlling X Client Access to AIX/6000" on page 125 for AIX as the X server, 4.2.4, "Controlling X Client Access to OS/2" on page 132 when OS/2 is the X server, and 4.3.1.4, "Controlling X Client Access" on page 153 for a DOS X server.

   b. Set up an X client display variable data set in order to identify the X server target display. This data set is called *userid.*XWINDOWS.DISPLAY. Please refer to 2.2.1, "Installation Verification for the MVS X Window System API" on page 26 for more information on this data set. The contents of this data set are illustrated in Figure 11 on page 27.

   c. In order to interact with the sample programs using the X server keyboard you will need to alter the keyboard mapping for the Enter key. We discovered that the default keyboard mappings provided with the X Window Systems for both OS/2 and AIX had the Enter key set to the keysym name RETURN. You need to remap this key's keysym name to EXECUTE. Please refer to 4.2.2.1, "Remapping the Keyboard under OS/2" on page 128 for a description of how to perform the remapping with OS/2 and 4.1.5, "Remapping the Keyboard Under AIX/6000" on page 123 for AIX.

   d. Execute the sample programs GXDEMO1 through GXDEMO6 from the TSO command prompt. Each program displays a series of frames. You can progress through the frames by pressing the Enter key. The application will close the window and end after the last frame.

      An example of the first frame for GXDEMO1 is illustrated in Figure 13.

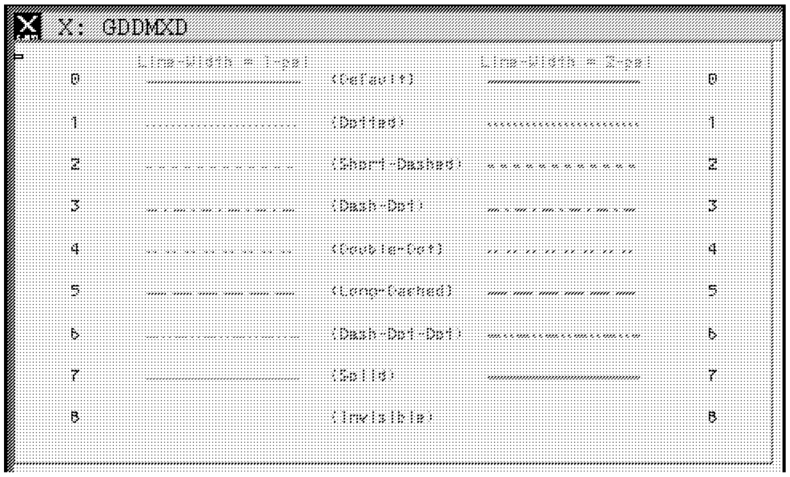


*Figure 13. Display at an OS/2 X Server for the GXDEMO1 First Frame*

At the TSO screen you will get the messages:

```
GDDMXD: File GDXALTCS PSS not found. No alternative character set.
```

This is an informational message and does not indicate an error. GDXALTCS.PSS is an alternate APL2 character set mapping. For more information on this data set please refer to 3.1.3.2, "APL2 Character Set Keyboard for GDDM under MVS" on page 76.

## 2.3 VM

The following is a guide to installing the X Window System under VM. The description is divided into two parts as follows:

1. Installation Verification for the X Window System API
2. Installing the X Window System GDDM Interface

It is assumed that the base TCP/IP product has been installed on your VM system and uses the minidisk addressing convention as described in *TCP/IP Version 2 Release 2 for VM: Planning and Customization.* This convention means that TCPMAINT controls the following minidisks:

- TCPMAINT user A disk (191)
- Client code disk (592)
- Server code disk (591)
- Service (PTF) disk (2C1)
- Source code disk (5C3)
- Samples disk (5C4)

### 2.3.1 Installation Verification for the VM X Window System API

The X Window system code is installed on the TCPMAINT client code disk at address 592. All that needs to be done is to verify that the API is ready to accept calls from an X client application.

IBM provides three sample programs with TCP/IP for VM. They are identical to those provided with MVS and are called XSAMP1, XSAMP2 and XSAMP3. In order to execute them they need to be compiled and link-edited as follows:

1. You need to have access to the following disks:

   - TCPMAINT 592 minidisk for TCP/IP client code
   - TCPMAINT 5C4 minidisk for the sample programs
   - The minidisk that has the C/370 compiler

   If you have the required authority you can access a disk under VM by typing the following command at the CMS prompt:

   ```
   LINK machine xxx yyy RR
   ACC yyy z
   ```

   where *machine* is the virtual machine that owns the minidisk, *xxx* is the address of the minidisk, *yyy* is the virtual address that the minidisk will be known to your virtual machine and *z* is the mode. For the following discussion we have assumed that *xxx* equals *yyy.*

2. On the 5C4 minidisk you will find the following sample source files:

   - XSAMP1 C

- XSAMP2 C
- XSAMP3 C

3. Compile and link-edit XSAMP1 by doing the following:

   a. Set the LOADLIB and TXTLIB search order using the following CMS commands:

   ```
   SET LDRTBLS 25
   GLOBAL LOADLIB EDCLINK
   GLOBAL TXTLIB X11LIB COMMTXT EDCBASE IBMLIB CMSLIB
   ```

   b. Compile XSAMP1 C using the following command:

   ```
   CC XSAMP1 (DEFINE(IBMCPP)
   ```

   This creates XSAMP1 TEXT on your A disk.

   c. Link-edit XSAMP1 using the following command:

   ```
   CMOD XSAMP1
   ```

   This creates XSAMP1 MODULE on your A disk.

4. Ensure that the X server has authorized the VM host as an X client. For specific details on how to do this please refer to 4.1.6, “Controlling X Client Access to AIX/6000” on page 125 for AIX as the X server, 4.2.4, “Controlling X Client Access to OS/2” on page 132 when OS/2 is the X server, and 4.3.1.4, “Controlling X Client Access” on page 153 for a DOS X server.

5. Identify the target X server display using the following CMS command:

   ```
   GLOBALV SELECT CENV SET DISPLAY <X client display variable>
   ```

   where *<X client display variable>* is the variable that the X client must use to access the X server. It has the format *host:0.0* where *host* is the Internet address of the X server and *0.0* represents the *target server.server screen*. For example, when using our OS/2 machine as the X server, since it had the Internet address 9.67.38.89, we typed the command:

   ```
   GLOBALV SELECT CENV SET DISPLAY 9.67.38.89:0.0
   ```

6. Run XSAMP1 by simply typing XSAMP1 at the CMS command line.

   XSAMP1 opens a display at the X server for 60 seconds and then closes it. The display is illustrated in Figure 8 on page 27.

7. The process for compiling and link-editing XSAMP2 is the same as for XSAMP1 except for setting the LOADLIB and TXTLIB search order:

   ```
   SET LDRTBLS 25
   GLOBAL LOADLIB EDCLINK
   GLOBAL TXTLIB XAWLIB XTLIB X11LIB COMMTXT EDCBASE IBMLIB CMSLIB
   ```

This is because XSAMP2 uses some of the intrinsics in the Xt library (XTLIB) and some of the widgets from the Athena widget set library (XAWLIB).

8. When you compile XSAMP2 you will see the following warning:

```
WARNING EDC0244 XSAMP2 C D1:106 External name fallback_resources
has been truncated to FALLBACK.
```

This is normal.

9. When you execute XSAMP2 you will see a window as illustrated in Figure 9 on page 27. The best way to end XSAMP2 is to close down the window at the X server.

10. The process for compiling and link-editing XSAMP3 is the same as for XSAMP1 and XSAMP2 except for setting the LOADLIB and TXTLIB search order:

```
SET LDRTBLS 25
GLOBAL LOADLIB EDCLINK
GLOBAL TXTLIB XMLIB XTLIB X11LIB COMMTXT EDCBASE IBMLIB CMSLIB
```

This is because XSAMP3 uses some of the intrinsics in the Xt library (XTLIB) and some of the widgets from the OSF/Motif widget set library (XMLIB).

11. When you execute XSAMP3 you will see a window as illustrated in Figure 10 on page 27. The best way to end XSAMP3 is to close down the window at the X server.

**Note:**

Ensure that your virtual machine has sufficient virtual storage for the compile and link-edit steps. Otherwise these steps will fail with a severe error:

```
Virtual Storage Exceeded.
```

### 2.3.2 Installing the VM X Window System GDDM Interface

The X Window System GDDM interface must be installed and activated in order for a GDDM application to send its output to an X server for display using the X protocol.

When installing the X Window System GDDM interface, the GDDM shared segment needs to be reinstalled. A shared segment is a mechanism that VM uses to make executable modules available in a single image for use by multiple users. If a shared segment was not used, each user would need a copy of the executable code running within their own virtual machine. Using a segment saves VM resources and greatly improves the performance of applications under VM.

The GDDM shared segment is for GDDM modules and improves the performance of GDDM applications. It needs to be reinstalled with the X Window System GDDM interface to allow the interface modules to access the shared segment.

**Note:** For our VM system we opted to make a copy of the GDDM shared segment for use by the X Window System GDDM interface modules. This is because if we reinstalled the existing named GDDM shared segment we would need to reassemble the bootstrap modules for each of the products that use GDDM.

If you elect to make a new named copy of the GDDM shared segment then you will need to zap the modules for those products that will be used with the new GDDM shared segment for the X Window System GDDM interface. This needs to be done to point the module at the new named GDDM shared segment that must be accessed for the X Window System GDDM interface. Please refer to 3.2.3, “Using GDDM Applications under VM” on page 87 for more information on zapping modules.

#### 2.3.2.1 Installation Steps

The following steps provide a guide to installing the X Window System GDDM interface under VM:

1. Ensure that you have access to the TCPMAINT 592 minidisk on which you will find the following files:

   | | |
   |---|---|
   | **GDDMXD TXTLIB** | is the executable code. |
   | **INSTGDXD EXEC** | is the installation EXEC. |
   | **GDDMXD EXEC** | activates and deactivates the X Window System GDDM interface. |
   | **COMMTXT TXTLIB** | is the common TCP/IP text library. |
   | **X11LIB TXTLIB** | is the X Window function library. |
   | **GDXALTCS PSS** | is an alternative character set for APL2. |
   | **GDXAPLCS SAMPMAP** | is a sample keyboard mapping for APL2. |

2. Ensure that you are linked to the GDDM minidisk and have access to the following files:

   - ADMBLSEG EXEC
   - ADMGLIB TXTLIB
   - ADMNLIB TXTLIB
   - ADMPLIB TXTLIB

3. Ensure that you are linked to the C/370 minidisk and have access to the C/370 runtime library, IBMLIB TXTLIB.

4. Before running the installation EXEC, you need to define the GDDM shared segment that will be available to the X Window System GDDM interface. You need to know the name and the spool location of the existing GDDM shared segment. This will be installation dependent and can be determined by typing the following command from the CMS command line:

   ```
   Q NSS NAME filename MAP
   ```

   where *filename* is the name of the existing saved shared segment. This will give you the beginning and end segment locations for the shared segment.

   For example the segment name that was defined when GDDM was installed for VM on our machine was ADMXA230. We issued the command:

   ```
   Q NSS NAME ADMXA230 MAP
   ```

   which produced the output as illustrated in Figure 14 on page 35.

**Note:** You need to have a CP privilege class E to execute this command.

```
FILE FILENAME FILETYPE MINSIZE  BEGPAG ENDPAG TYPE CL #USERS PARMREGS VMGROUP
0325 ADMXA230 DCSS       N/A    04000  042FF   SR  P  00001   N/A       N/A
0443 ADMXA230 DCSS       N/A    04000  042FF   SR  A  00000   N/A       N/A
Ready; T=0.01/0.01 17:55:13
```

*Figure 14. Example of the Output from the Q NSS NAME Command on VM*

5. Define the new GDDM shared segment by typing the following command:

```
DEFSAG <new segment name> begpag-endpag SR
```

   where:

   - *<new segment name>* is the new name of the shared segment.
   - *begpag* is the segment start location.
   - *endpag* is the segment end location.
   - SR means share/read.

   You must use the *begpag* and *endpag* values that were returned from the Q NSS NAME command. Refer to the example illustrated in Figure 14. On our VM system we typed:

```
DEFSEG GDDMXD 4000-42FF
```

   where GDDMXD is the name we selected for the GDDM shared segment to be reinstalled.

6. The X Window System GDDM interface is actually installed by invoking the exec INSTGDXD. Start this at the CMS command prompt.

7. During the execution of INSTGDXD you will be need to respond to prompts as follows:

   a. Which are the products for which you want saved segments installed? The options are GDDM, PGF, IMD, IVU. You only need to choose GDDM.

   b. What is the name of the saved segment? Enter the name you choose when you defined the saved segment using the DEFSEG command. In our installation we used GDDMXD.

   c. Do you want to include the Image Symbol Editor routines? Respond YES.

   d. Do you want to include the Image Processing routines? Respond YES.

   The INSTGDXD performs the following:

   a. Builds the following load modules and places them on your A disk.

      - GDXLIOX0 MODULE
      - GDXCLOSE MODULE
      - GDXADML1 MODULE
      - KEYCODE MODULE

   b. Builds the demonstration load modules GXDEMO1 through GXDEMO6 and places them on your A disk.

   c. Builds the GDXBLSEG EXEC and invokes it to create the GDDM shared segment.

8. Ensure that INSTGDXD runs with no errors. While it is running INSTGDXD provides you with messages to indicate the step that is currently executing. An example of the last messages you should see for a successful run are illustrated in Figure 15 on page 36.

```
*** Segment name: GDDMXD   Start address: 04000000...

*** Loading TEXT decks....
Saved segment GDDMXD was successfully saved in fileid 0444.
*** GDDM DCSS build complete.

*** ADMBLSEG complete for chosen products
Ready; T=3.00/4.51 17:57:50
```

*Figure 15. Example of a Successful Run for INSTGDXD*

9. You can check that the GDDM shared segment has been installed successfully by issuing the following command:

   ```
   Q NSS NAME <new segment name> MAP
   ```

   where <*new segment name*> is the name of the GDDM shared segment just created by INSTGDXD. Notice that the class of this segment is A, which means available.

10. Copy the modules built by INSTGDXD from your A disk across to a minidisk such as the TCPMAINT 592 disk so that they may be commonly accessed.

### 2.3.2.2 Installation Verification

Use the following steps as a guide to verifying the installation of the X Window System GDDM interface for VM.

1. IBM provides some sample programs that allow you to test the X Window System GDDM interface under VM. Before running these programs you need to do a number of tasks. In fact, each time you want to have a GDDM application display its output through the X Window System GDDM interface to an X server you must do these same tasks as follows:

   a. Ensure you have access to the TCPMAINT 592 minidisk, the GDDM minidisk and the minidisk that holds the C/370 libraries.

   b. For VM/XA* systems only, enter the following command from the CMS command prompt:

      ```
      SET STORECLR ENDCMD
      ```

      This ensures that GETMAIN requests by the X Window System GDDM interface code are processed correctly.

   c. Activate the X Window System GDDM interface by issuing the following command:

      ```
      GDDMXD ON
      ```

      You should see the message `GDDMXD/VM active`.

d. Set the following CMS global variables by issuing these three commands:

```
SET LDRTBLS 25
GLOBAL LOADLIB EDCLINK
GLOBAL TXTLIB ADMNLIB GDDMXD ADMPLIB ADMGLIB X11LIB COMMTXT IBMLIB
EDCBASE CMSLIB
```

e. Identify the target X server display using the following CMS command:

```
GLOBALV SELECT CENV SET DISPLAY <X client display variable>
```

where *<X client display variable>* is the variable that the X client must use to access the X server. It has the format *host:0.0* where *host* is the internet address of the X server and *0.0* represents the *target server.server screen*. For example, when using our OS/2 machine as the X server, since it had the Internet address 9.67.38.89, we typed the command:

```
GLOBALV SELECT CENV SET DISPLAY 9.67.38.89:0.0
```

2. Ensure that the X server has authorized the VM host as an X client. For specific details on how to do this please refer to 4.1.6, "Controlling X Client Access to AIX/6000" on page 125 for AIX as the X server, 4.2.4, "Controlling X Client Access to OS/2" on page 132 when OS/2 is the X server, and 4.3.1.4, "Controlling X Client Access" on page 153 for a DOS X server.

3. In order to interact with the sample programs using the X server keyboard you will need to alter the keyboard mapping for the Enter key. We discovered that the default keyboard mappings provided with the X Window Systems for both OS/2 and AIX had the Enter key set to the keysym name *RETURN*. You need to remap this key′s keysym name to *EXECUTE*. Please refer to 4.2.2.1, "Remapping the Keyboard under OS/2" on page 128 for a description of how to perform the remapping with OS/2 and 4.1.5, "Remapping the Keyboard Under AIX/6000" on page 123 for AIX.

4. Execute the sample programs GXDEMO1, GXDEMO2, GXDEMO3 GXDEMO4, GXDEMO4A, GXDEMO5 and GXDEMO6 from the CMS command prompt. Each program displays a series of frames that you can progress through by pressing the Enter key. The application will close the window at the X server and end after the last frame.

   An example of the first frame for GXDEMO1 is illustrated in Figure 13 on page 30.

**Note:**

The GXDEMOx programs will not open a window at the X server and will simply display output to your 3270 terminal if you have opted to reinstall the GDDM shared segment with a new name. This is because the GXDEMOx programs are built by INSTGDXD before the new GDDM shared segment is installed which means that they point to the existing named shared segment.

To fix this you need to zap the GXDEMOx modules to point to the new named shared segment. Please refer to Appendix C, "Information on Zapping the VM GXDEMOx Programs" on page 177 for more information on how to zap modules.

## 2.4 AIX

The following section describes the installation of the 2D feature of AIXwindows Environment/6000. AIX V3.2.5 was the base operating system used for the activities documented below.

There are several ways to install the software. For our installation we used the internal 2.3 GB tape /dev/rmt0. To complete the installation you can use either a graphic display or an ASCII terminal. It is your decision which images you want to install.

The licensed program product AIXwindows Environment/6000 for the IBM RISC System/6000 is packed in several images. The images that you will install will depend upon your installation requirements; however, we recommend the installation of the following images:

```
X11rte.obj
X11rte.ext.obj
X11dev.obj
X11dev.src
X11dev.motif1.2.obj
X11dev.motif1.2.src
X11dev.im
X11fnt.iso88591.aix.fnt
X11fnt.bim850.pc.fnt
X11deviEn_US.info
X11En_US.msg
```

The minimum installation is the X11rte.obj image which provides the X11 runtime environment. If, in addition to running X11 applications, the development of such applications is required then you should install the X11dev images.

The list in Table 3 on page 41 shows you the contents of the images.

### 2.4.1 Basic Installation

Please use the following steps as a guide to installing AIXwindows Environment on a AIX RISC System/6000. The following steps are for a system with the *En_US* language environment. If you have another language environment please refer to the documentation and the messages in your own language.

1. Log in as root.
2. Insert the tape with the software into your tape drive. Make sure that your device is available. Type the command:

   ```
   lsdev -C -c tape
   ```

   Your output should look similar to this:

   ```
   rmt0 Available 00-08-00-20 2.3 GB 8mm Tape Drive
   ```

3. On the command prompt enter:

   ```
   smitty
   ```

   Starting from the System Management menu select the following options:

   - Software Installation & Maintenance
   - Install / Update Software
   - Install / Update Selectable Software (Custom Install)
   - Install Software Products at Latest Available Level

4. You get the menu Install Software Products at Latest Available Level from where you can select the input device. Press F4 to get the list of the possible devices. Select the device on which you have inserted the tape by pressing the Enter key.
5. Now you get a screen like this:

```
        Install Software Products at Latest Available Level

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

[Entry Fields]
*INPUT device / directory for software             /dev/rmt0.1
*SOFTWARE to install                               [all]         +
 Automatically install PREREQUISITE software?      yes           +
 COMMIT software?                                  yes           +
 SAVE replaced files?                              no            +
 VERIFY software?                                  no            +
 EXTEND file systems if space needed?              yes           +
 REMOVE input file after installation?             no            +
 OVERWRITE existing version?                       no            +
 ALTERNATE save directory                          []         +                          +
***********************************************************************

***********************************************************************





F1=Help          F2=Refresh          F3=Cancel          F4=List
F5=Undo          F6=Command          F7=Edit            F8=Image
F9=Shell         F10=Exit
```

*Figure 16. SMIT Installation Menu*

The values shown above in the entry fields are the default values.

The entry fields and their associated values are explained below:

| | |
|---|---|
| *INPUT device* | This means the device you mounted the installation tape on. |
| *SOFTWARE to install* | By pressing F4 you get a list of the products that are on the tape. |
| *PREREQUISITE software* | To be sure that all prerequisite software will be installed, leave this parameter as *yes*. |
| *COMMIT software* | *Commit* means that you cannot remove the current level of the software to be installed. When you choose not to commit you need additional disk space for the save of the previous version. We recommend to use the parameter *no*. The software will be applied and you can remove the software after installation. |
| *SAVE replaced files* | If you choose *no* for the *Commit* option you should set this parameter to *yes*. |
| *VERIFY software* | If you want to be sure that your installation will be checked, set this parameter to *yes*. |
| *EXTEND* | Set this parameter to *yes* to be sure that there is enough space on your file system for the installation (you must have enough free physical partitions for this option to work correctly). |

| | |
|---|---|
| *REMOVE* | Has no effect for installation from tape. |
| *ALTERNATE* | You can specify here a directory of your choice, instead of the default, to save replaced files (see option above). |

6. Move the cursor to the field *SOFTWARE to install? and press F4. A list will be shown where you can select the desired items with the F7 key. Press Enter to complete the selection. Until this point you can quit the installation by either pressing the F3 or the F10 key. When you press Enter, the installation will start.

   **Note:** On the upper left corner you see the string running. This is the only indication that the installation is running. You have to wait until the string changes to ok. If the string displays failed, refer to your system administrator.

7. When you see the ok string on the upper left corner, then the installation was successful, and you can begin either with customization or starting AIXwindows as described below.

To start the X server your workstation must be equipped with a graphic display. It is possible to start the X server without customization after the installation. To start the X server you first need to log in at your graphic display. You may then enter either the xinit or startx commands at the command line in order to start the X server. The startx command is the preferred method of starting as it is streamlined and includes such functions as setting the user′s DISPLAY environment variable. Don′t switch your session when the X server is starting. The display will change to gray and you will see an hour glass in the middle of the screen. After that the xclock client, an aixterm window, the Power Desktop window, and the Motif Window Manager will be started.

To quit the X server press Alt, Ctrl and the Backspace key together. Don′t worry if you see an error message. This may happen if there are still clients running on your display when you exit.

### 2.4.2 AIXwindows Environment/6000 Images

*Table 3 (Page 1 of 2). Contents of AIXwindows Environment/6000*

| Image | Content |
|---|---|
| X11rte.obj | AIXwindows Run Time Environment includes:<br>• AIXwindows Motif V1.2 Run Time Environment<br>• X Windows Run Time Environment |
| X11rte.ext.obj | AIXwindows Run Time Environment Extensions includes:<br>• X Windows Run Time Environment Extensions |
| | AIXwindows Development Environment includes: |
| X11dev.im | • AIXwindows Development Sample Input Method Servers |
| X11dev.motif1.2.obj | • AIXwindows Motif V1.2 Libraries and Include Files |
| X11dev.motif1.2.src | • AIXwindows Motif V1.2 Sample Programs |
| X11dev.obj | • X Windows Development Environment |
| X11dev.src | • X Windows Development Environment Unsupported Source |

*Table 3 (Page 2 of 2). Contents of AIXwindows Environment/6000*

| **Image** | **Content** |
|---|---|
| X11fnt.iso88591.aix.fnt | AIXwindows Latin 1 (ISO8859-1) fonts |
| X11fnt.iso88592.fnt | AIXwindows Latin 2 (ISO8859-2) fonts |
| X11fnt.iso88593.fnt | AIXwindows Latin 3 (ISO8859-3) fonts |
| X11fnt.iso88594.fnt | AIXwindows Latin 4 (ISO8859-4) fonts |
| X11fnt.iso88595.fnt | AIXwindows Cyrillic 5 (ISO8859-5) fonts |
| X11fnt.iso88597.aix.fnt | AIXwindows Greek 7 (ISO8859-7) fonts |
| X11fnt.iso88599.aix.fnt | AIXwindows Turkish (ISO8859-1) fonts |
| X11fnt.ibm850.pc.fnt | AIXwindows Latin 1 (IBM-850) fonts |
| X11kanji.aix.fnt | AIXwindows Kanji fonts |
| X11fnt.coreX.fnt | AIXwindows Core X11 Fonts |
| X11fnt.oldX.fnt | MIT X11R3 contrib fonts: bmug, info-mac, oldx10 and oldx11 |

Version 3.2 of the AIX operating system for the RISC System/6000 (AIX/6000) is a prerequisite for the installation of AIXwindows Environment/6000. It is not necessary that TCP/IP be installed, because the X server and X clients can run on the same host. However, TCP/IP is required if the RISC System/6000 will be used to communicate with remote client or server platforms.

The `X11rte.obj` image is a prerequisite for all other images of AIXwindows Environment/6000.

## 2.5 OS/2

Below are details of how to install and setup the OS/2 X Window System server (PMX).

### 2.5.1 Setting Up OS/2 X Window System Server

The OS/2 X Window System server (PMX) enables you to display and control X Window System client application programs in OS/2 windowed sessions. These client application programs can reside in one or more IBM or other computing systems that support the X Window System client function. They are connected to the PMX host through a TCP/IP network. IBM systems that currently have X Window System client capability include OS/2, VM, MVS, and AIX* (RISC/6000).

PMX uses large amounts of memory. We recommend that your workstation has at least 8MB of memory for the PMX component. The amount depends on which X Window System client applications are running, and how much memory the individual programs require.

To use PMX, you must have installed on your workstation the TCP/IP protocol stack from the TCP/IP for OS/2 Base Kit. This chapter describes how to install PMX in this environment.

Use the following steps as a guide to installing and customizing the X Window System under OS/2.

It is assumed that the base TCP/IP product has been installed on your OS/2 system.

**Prerequisite Programs**: also In order to install and use this kit, you must also have the following programs:

- IBM TCP/IP Version 2.0 for OS/2: Base Kit
- IBM OS/2 Version 2.0 with Service Pak applied, or higher

**Memory Requirements**: The following table shows the approximate memory requirements for this kit. Note that the actual memory requirements depend heavily on the mix of X applications that are used with the X window system server.

*Table 4. Memory Requirements*

| **TCP/IP function** | **Minimum Recommended Memory (MB)** |
|---|---|
| X Window System server | 8 |

**Disk Space**: it. The following table shows the disk space required for this kit.

*Table 5. Disk Space Requirements*

| **TCP/IP Kit** | **Disk Space (MB)** |
|---|---|
| X Window server Kit | 11.7 |

You should also ensure that you have enough free disk space for the OS/2 swapper. The SWAPPATH statement in your CONFIG.SYS file specifies on which disk that free space must be.

*Table 6. Disk Space Requirements for Swapper*

| | |
|---|---|
| Free Space for Swapper | 10MB |

## 2.5.2 Installing PMX

PMX includes the following components:

- PMX program (PMX.EXE)
- X Window System font support
- National language support for keyboards
- X Window System utilities

To install the PMX Kit using the Installation Program follow these steps:

1. Insert the X Window System server installation diskette.
2. At an OS/2 command prompt, type A:TCPINST and press Enter.
3. Select the X Window System server kit.

   You can also select installation of the BookManager softcopy publications and the IBM Library Reader at this time.
4. Select the Install pushbutton.
5. Insert diskettes as prompted.
6. Exit the program.

7. Restart your system to make the installation effective.

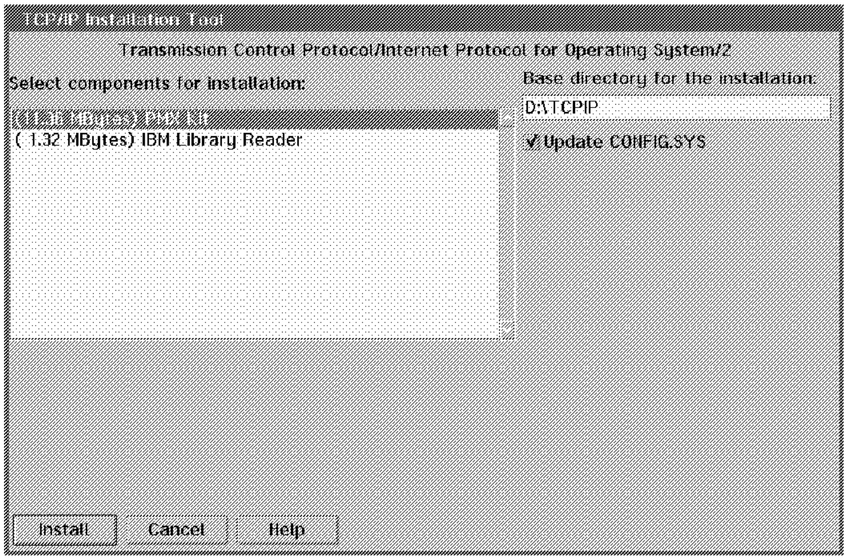

TCP/IP Installation Tool
Transmission Control Protocol/Internet Protocol for Operating System/2
Select components for installation:
(11.36 MBytes) PMX Kit
( 1.32 MBytes) IBM Library Reader
Base directory for the installation:
D:\TCPIP
✓ Update CONFIG.SYS
Install
Cancel
Help

*Figure 17. Installing X Window Server*

### 2.5.3 System Level

We ran the described tests with the following system maintenance level. Note that CSDs are regularly issued for the TCP/IP kits and it is strongly recommended that you obtain and install such CSDs as they become available. Please refer to the X Window System Server Guide for details on obtaining OS/2 TCP/IP CSDs either by FTP from software.watson.ibm.com or by downloading from a bulletin board or by calling IBM service.

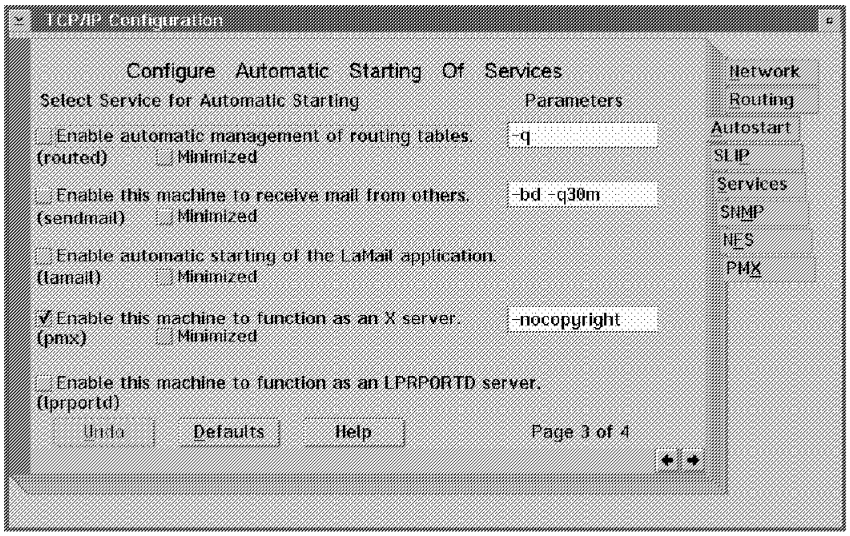
You can display the system maintenance level with the SYSLEVEL command.

```
D:\TCPIP\SYSLEVEL.PMX
                     X-WINDOWS for TCP/IP on OS/2 2.0 and 2.1
Version 2.00     Component ID 562208600
Current CSD level: UN52841
Prior   CSD level: UN00000
```

*Figure 18. Syslevel from Test Installation*

### 2.5.4 Setting PMX to Start Automatically

Most users will want to have PMX start automatically when they start their workstations. You can use the configuration notebook program to achieve this. The configuration notebook program adds the necessary definitions to the startup.cmd file.



Configure Automatic Starting Of Services

Select Service for Automatic Starting — Parameters

☐ Enable automatic management of routing tables. (routed) ☐ Minimized — -q

☐ Enable this machine to receive mail from others. (sendmail) ☐ Minimized — -bd -q30m

☐ Enable automatic starting of the LaMail application. (lamail) ☐ Minimized

✓ Enable this machine to function as an X server. (pmx) ☐ Minimized — -nocopyright

☐ Enable this machine to function as an LPRPORTD server. (lprportd)

Page 3 of 4

*Figure 19. Automatic Startup from X Window System Server*

For more information see the *X Window System Server Guide,* SC31-7070.

## 2.5.5 Setting Environment Variables

To function properly, PMX needs certain environment variables to be set on your workstation. Environment variables can be set in two ways:

- In your CONFIG.SYS file (permanent)
- At an OS/2 prompt (temporary)

The following environment variables should be set:

- XFILES

  Many of the database files (such as RGB.TXT) and fonts required by PMX for initialization and operation are installed by default in a subdirectory named X11.

  You can locate the X11 subdirectory tree wherever you wish, as long as you also set the corresponding environment variable XFILES. Since PMX does not write to these files you can even locate them on a remote read-only filesystem. For example, if XFILES=D:\TCPIP\X11, PMX looks for files it needs in that subdirectory or its subdirectories.

- ETC

  PMX uses the ETC environment variable to find the X0HOSTS file and as the location for writing the PMX.LOG file. In the test installation it′s set to ETC=D:\TCPIP\ETC. PMX writes and reads the PMX.INI file, which contains the PMX settings from TCPIPCFG.

- DISPLAY

  XINIT uses the DISPLAY environment variable to run XMODMAP. XINIT displays an error message and ends if it finds that DISPLAY has not been set. The value for DISPLAY in our environment is display=9.24.104.51:0. You must change it when you are using the X Window System client to display an application on a remote X Window System server, but for test purposes take your local IP address or nickname.

If you do not set the environment variable in the CONFIG.SYS but in an OS/2 session, remember that it will only be set for that OS/2 session. Also, the setting will be cleared when you shut off or reboot your workstation.

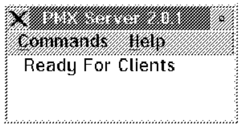
For more information see the *X Window System Server Guide* or the Help function. The Help function is in this case very useful.

### 2.5.6 Starting PMX

You can start PMX in one of the following ways:

1. Use the configuration notebook to specify that PMX is started whenever TCP/IP is started. Refer to 2.5.4, "Setting PMX to Start Automatically" on page 44 for information on how to specify the automatic starting of PMX using the configuration notebook.
2. Include PMX in an OS/2 Presentation Manager group and start it by clicking on the PMX icon.
3. Invoke PMX from an OS/2 command prompt.
4. Run the command file XINIT.CMD, which is located in the X11\BIN directory. We recommend using this method to start PMX. The command file gives you the opportunity to start PMX only when needed and provides logic which makes the startup more streamlined.

Once you have started up PMX you will see a Presentation Manager window opened, as illustrated in Figure 20. This window is for the control of the PMX server. All other windows are created through invoking client applications and are called X client windows. These are also OS/2 Presentation Manager windows which have Presentation Manager frames and title bars.



Ready For Clients

*Figure 20. The PMX Control Window*

It should be noted that PMX can only use PM's built-in window manager for user manipulation of X windows. Other X Window managers such as mwm cannot be run with PMX.

The mouse that is typically used with OS/2 has two buttons. Most X Window System implementations usually expect a mouse with three buttons and many X client applications require a mouse with three buttons at the server. PMX simulates a three button-mouse by supporting both buttons being pressed together on an OS/2 mouse to represent the third button.

## 2.6 Installing OS/2 X Window System Client and OS/2 OSF/Motif Kits

This section explains how to install X Window System client and OSF/Motif Kit. Before you install X Window System client, ensure that the required software is installed and running on the workstation.

## 2.6.1 Requirements to Use X Window System Client and the OSF/Motif Kit

The X Window System client kit requires that the following be installed and running on the OS/2 workstation:

- OS/2 2.x (32-bit)
- IBM TCP/IP Version 2.x for OS/2 (or comparable software that provides the necessary protocol stack)
- For development of X Window applications, the IBM TCP/IP Programmer′s Toolkit
- For development of X Window applications, the High Performance File System (HPFS)
- To run X Window System client applications locally, the X Window System server

In addition to the above, the OSF/Motif kit requires that the X Window System client be installed on the OS/2 workstation.

## 2.6.2 Installing the X Window System Client Files

You can install the executable sample X client programs, the X client programmer′s toolkit, or both. However, the programmer′s toolkit can be installed only to a disk formatted with the HPFS option.

To check which kind of filesystem you are using use the CHKDSK command.

You can also select installation of the BookManager softcopy publications and the IBM Library Reader at this time.

To install just the X Window System client executable files, either to a FAT or an HPFS file system, do the following:

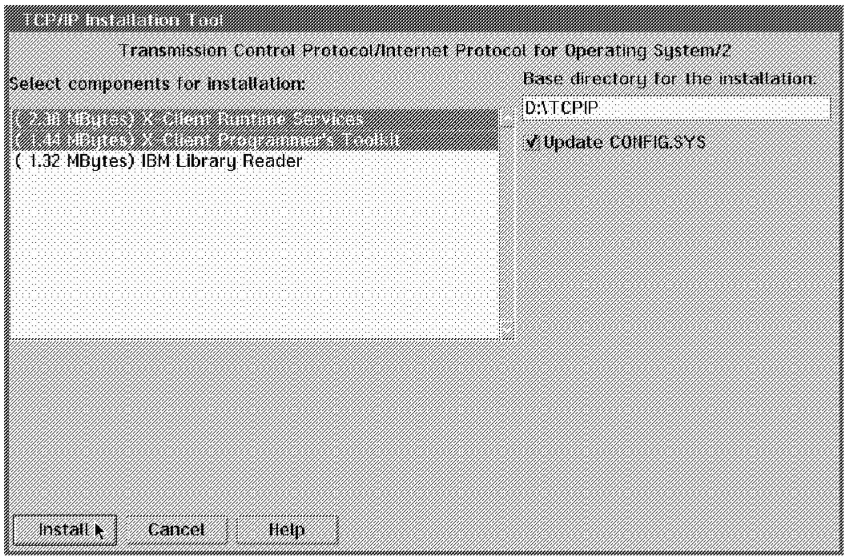
1. Insert the X Window System client diskette in your diskette drive.
2. At the A: prompt, enter `TCPINST`.
3. The TCP/IP Installation window is displayed.
4. Select X Client Runtime Services and select Install.

## 2.6.3 Installing the Programmer′s Toolkit

To install the X Window System client programmer′s toolkit to an HPFS file system, do the following:

1. Insert the X Window System client diskette in your diskette drive.
2. At the A: prompt, enter `TCPINST`. The TCP/IP Installation window is displayed.
3. Select X Client Programmer′s Toolkit and select Install.

You can also install both kits at the same time. See Figure 21 on page 48.


TCP/IP Installation Tool

Transmission Control Protocol/Internet Protocol for Operating System/2

Select components for installation:

( 2.38 MBytes) X-Client Runtime Services
( 1.44 MBytes) X-Client Programmer's Toolkit
( 1.32 MBytes) IBM Library Reader

Base directory for the installation:

D:\TCPIP

✓ Update CONFIG.SYS

*Figure 21. Installing X Window System Client*

## 2.6.4 Installing the OSF/Motif Kit Files

You can install the executable sample Motif programs, the Motif programmer′s toolkit, or both. However, the programmer′s toolkit can be installed only to a disk formatted with the HPFS option.

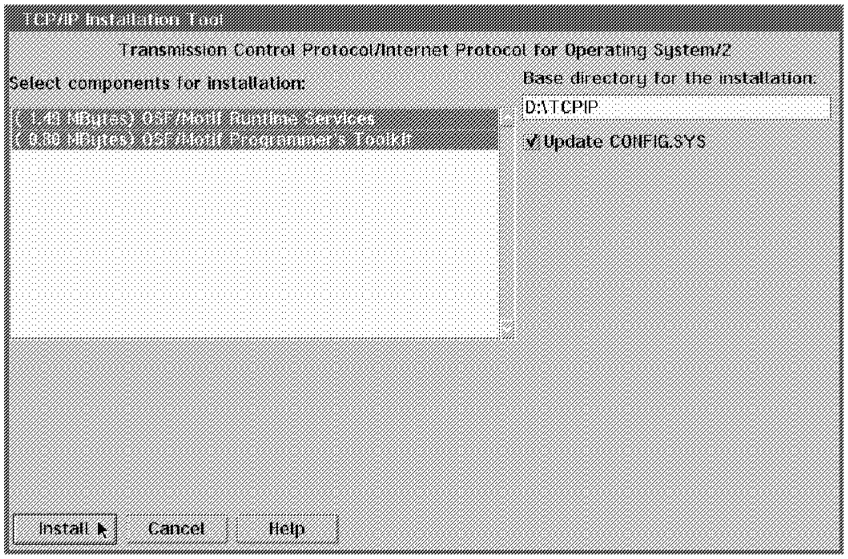
To install just the OSF/Motif kit executable files, either to a FAT or an HPFS file system, do the following:

1. Insert the OSF/Motif diskette in your diskette drive.
2. At the A: prompt, enter `TCPINST`. The TCP/IP Installation window is displayed.
3. Select OSF/Motif Runtime Services and select Install.

To install the OSF/Motif kit programmer′s toolkit to an HPFS file system, do the following:

1. Insert the OSF/Motif diskette in your diskette drive.
2. At the A: prompt, enter `TCPINST`. The TCP/IP Installation window is displayed.
3. Select OSF/Motif Programmer′s Toolkit and select Install.

You can also install both kits at the same time. See Figure 22 on page 49.



TCP/IP Installation Tool

Transmission Control Protocol/Internet Protocol for Operating System/2

Select components for installation:

( 1.48 MBytes) OSF/Motif Runtime Services
( 9.80 MBytes) OSF/Motif Programmer's Toolkit

Base directory for the installation: D:\TCPIP

✓ Update CONFIG.SYS

*Figure 22. Installing X Window System Client*

CSDs are regular issued for the TCP/IP kits and these should be obtained through normal channels and installed.

You can display the system maintenance level with the SYSLEVEL command.

```
D:\TCPIP\SYSLEVEL.XCL
                    X-CLIENT for TCP/IP on OS/2 2.0 and 2.1
Version 2.00     Component ID 562208600
Current CSD level: UN52842
Prior   CSD level: UN00000
```

*Figure 23. System Maintenance Level from Test Installation*

## 2.6.5 Installing from a Code Server

If you have the TCP/IP Version 2.0 for OS/2 base kit, you can install these kits remotely, from another workstation.

These kits comply with IBM's Configuration, Installation, and Distribution (CID) architecture, which provides for unattended, remote installation of programs and applications from code servers to client workstations.

For information about how to install these kits from a code server to a client workstation, see *TCP/IP Version 2.0 for OS/2: Installation and Administration*.

### 2.6.5.1 Setting Environment Variables

After installation, these environment variables should be set:

```
set XFILES=d:\tcpip\x11
set XFILESEARCHPATH=d:\tcpip\x11
set XAPPLRESDIR=d:\tcpip\x11
set XKEYSYMDB=d:\tcpip\x11\XKeysymD
set XENVIRONMENT=d:\tcpip\etc\xdefault
set XUSERFILESEARCHPATH=d:\tcpip\x11\app-def\%n
set XNLSPATH=d:\tcpip\x11\nls
set HOME=d:\tcpip\etc
```

You can change them in your CONFIG.SYS file, or you can change them with the OS/2 set command, from the command prompt, in each window session.

## 2.7 DOS

We installed two versions of the HCL-eXceed software. One for use in conjunction with Microsoft Windows Version 3.1 and the other for DOS.

HCL-eXceed/W Version 3.3.3 for Windows

HCL-eXceed/DOS Version 3.3 for DOS

The prerequisite software is IBM TCP/IP Version 2.0 or 2.1 for DOS (Program Number 5621-219) with the latest CSD or TCP/IP Version 2.1.1.

**Note:** HCL-eXceed release 3.2 does not support IBM TCP/IP Version 2.1 for DOS. You have to use HCL-eXceed V3.3 or higher.

### 2.7.1 Installation and Basic Configuration for HCL-eXceed/DOS

1. Our installation took 12.5MB (including 8.1MB for all font files). Make sure that you have enough free storage and that TCP/IP is not started with TCPSTART before.
2. Insert diskette 1 of 5 and proceed with the following commands:

   ```
   a:> install
   ```

3. On the first screen you will be asked for the installation directory. We changed it to C:\XDOS.
4. Our selection on the next screen was P because it was the first time that we installed HCL-eXceed/DOS.
5. The next screen shows you the TCP/IP interfaces that are supported by HCL-eXceed/DOS. Choose J for TCP/IP Version 2.1.
6. On the next screen you can select your display. We have chosen all to use an automatic selection for the physical installed Display.
7. Select all possible fonts on the next screen
8. The next screen is the validation screen for your selection. Press Enter to continue, Esc to change, or Ctrl-C to quit.

   Follow the instructions displayed on your screen for diskettes 2, 3, 4 and 5.
9. The installation asks you to update the AUTOEXEC.BAT file. There are two entries that are added if you press Enter:

   ```
   set EXCEEDP=C:\XDOS
   path=%path%;C:\XDOS
   ```

   You are also prompted to confirm the system drive (usually C:\)
10. When you customize later, the EXCEEDP variable must be set so that the xconfigp program works correctly.

11. The next step is the basic configuration of HCL-eXceed/DOS software. When the Installation complete message appears on your screen, you can configure your system right away by pressing Y. Otherwise you can terminate installation. If you configure now you are using a guided tour through the configuration utility.

    If you start the configuration utility later you can choose between the normal configuration and the guided tour. The guided tour provides additional panels which describes the next steps. For the normal configuration type at your command line:

    ```
    xconfigp
    ```

    and press Enter. For the guided configuration type at your command line:

    ```
    xconfigp -install
    ```

    and press Enter.

12. In the following steps we added entries to enable us to connect to an IBM RISC System/6000 X client. The configuration is menu driven, and for each field the configuration program delivers a detailed help text. We made the following changes to the default values:

    **Input Settings Menu**

    ```
    Take the default values
    ```

    **Communication Settings Menu**

    ```
    Startup mode          TELNET
    Hosts file            c:\tcp211\etc\hosts
    Connect host          rs60007
    Userid                root
    Backspace             EC
    Transport             IBM TCP/IP for DOS
    ```

    **Note:** We decided to connect to an IBM RISC System/6000 with the TELNET command:

    | | |
    |---|---|
    | `Hosts file` | refers to the IBM TCP/IP hosts file. |
    | `EC` | defines the backspace key on the AIX Telnet Server. |

    **Note:**

    If you define your local host in the hosts file, the nickname used in the hosts local file and the hostname defined in the configuration utility in TCP/IP for DOS Version 2.1.1 (which sets the HOSTNAME variable via the CONFIG.SYS file), must be identical! If you don′t do this, eXceedp won′t start.

    **Video and Color Settings**

    ```
    Foreground            hotpink
    Background            white
    Server Class          PseudoColor
    ```

You can use every color defined in the RGB database. More information about the RGB database can be found in 4.3.2.1, "Customizing Colors" on page 153.

**General settings**

```
hwm                    load
htelnet                load
control panel          load
```

**Note:** To enable hwm, htelnet and the control panel to be started on request later.

13. After completing the configuration choose the options EXIT and SAVE on the panel.
14. Reboot your system.
15. You can start the HCL-eXceed/DOS X Window server with the command:

```
EXCEEDP
```

16. A telnet session to the AIX Server should be started. When the telnet session is established you can submit a command, which starts an X client application on the DOS host. We used the following commands to display a clock on the DOS host:

```
rs60007#> DISPLAY=DOS20:0
rs60007#> export DISPLAY
rs60007#> xclock &
```

*Figure 24. Telnet Session to AIX to Start X Client Application*

To stop the X server press Esc and then Esc again. To restart, type exceedp. To control the HCL-eXceed/DOS X Window server you can use the commands, listed in the following table. Each command is initiated with the following:

```
<Left Alt> + <Esc>
```

*Table 7 (Page 1 of 2). HCL-eXceed Server Commands*

| Keystroke | Operation |
|---|---|
| <Left Alt> + <Esc>, followed by <Esc> | Terminate the server |
| <Left Alt> + <Esc>, followed by C | Access Control Panel |
| <Left Alt> + <Esc>, followed by D | Switch to DOS |
| <Left Alt> + <Esc>, followed by <Enter> | Execute the command in the DOS command field of XCONFIGP's General settings menu |
| <Left Alt> + <Esc>, followed by l | Write a socket summary to the Log file |

*Table 7 (Page 2 of 2). HCL-eXceed Server Commands*

| **Keystroke** | **Operation** |
|---|---|
| <Left Alt> + <Esc>, followed by K | Kill all connections and reset the server |
| <Left Alt> + <Esc>, followed by L | Start the hwm window manager if it is enabled, or restart hwm |
| <Left Alt> + <Esc>, followed by Q | Display a help screen listing commands you can enter from the server status line |
| <Left Alt> + <Esc>, followed by S | Load or reload the htelnet client if it is enabled |
| <Left Alt> + <Esc>, followed by T | Enable tracing |
| <Left Alt> + <Esc>, followed by U | Disable tracing |
| <Left Alt> + <Esc>, followed by Z | Kill all write-blocked clients |
| <Left Alt> + <Esc>, followed by A | Enable/Disable Access database modifications |
| <Left Alt> + <Esc>, followed by H | Reload host Access database |
| <Left Alt> + <Esc>, followed by R | Reload RGB database |
| <Left Alt> + <Esc>, followed by F | Reload Font database |
| <Left Alt> + <Esc>, followed by <Spacebar> | Abort the status line and return to the server without entering a command |

## 2.7.2 Installation and Basic Configuration for HCL-eXceed/W

Make sure that you have 12.5MB disk space available for the installation of HCL-eXceed/W (including 7.7MB for the fonts).

1. Windows must be started before you can install HCL-eXceed/W. Insert diskette 1 of 7. Select **run** from the Windows Program Manager's File menu. Enter `a:setup`.

2. On the first window we selected the following:

**First Window**

```
Setup:                  Personal
Install:                Complete
```

3. On the next window we defined the path in which to install eXceed/W:

**Installation Directory**

```
Home Directory:         C:\XWIN
```

4. On the next window we defined the user directory:

**User directory**

```
User Directory:         C:\XWIN\USER
```

5. On the next window we defined the transport interface:

**Transport Interface**

```
Select:        Window Sockets API
```

**Note:** Do not select IBM TCP/IP V2.1 !

6. On the next window we selected the fonts:

**Installing fonts**

```
We selected all fonts
```

7. The next dialog box covers the local X client support:

**Installing local X clients**

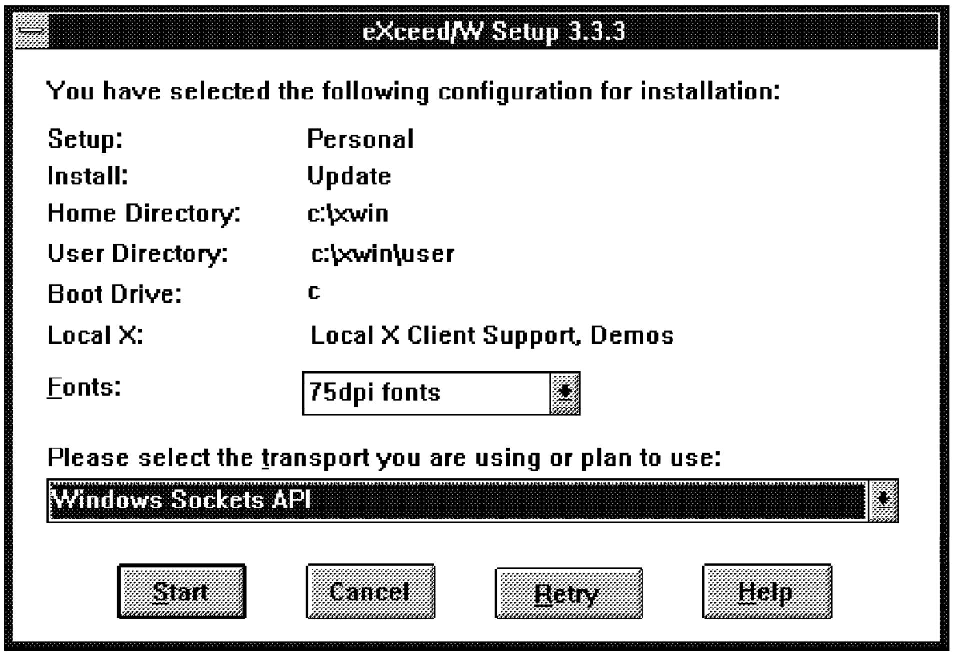
```
We selected:      Local X Client Support, Demos
```

Local X client support allows you to run local X clients. Local X clients are X clients that have been built or rebuilt to run on a PC with Microsoft Windows, rather than on a UNIX host. A number of sample local X clients have been included with eXceed/W. The X Software Development Kit (XDK) allows you to create your own local X clients. **Requirements**:

In order to develop local X clients on your PC using the X Software Development Kit, you must have the following:

- Microsoft Windows Version 3.0 or higher, running in Standard or 386 Enhanced mode. Note, however, that Windows V3.1 is the prerequisite for TCP/IP Version 2.1.
- The Microsoft Windows Software Development Kit.
- The Microsoft C Compiler (Version 6.0AX or 7.0 is recommended.
- The eXceed/W X server.
- Local X support and the X Software Development kit.

8. On the next window all selected items appear. The window should look like Figure 25 on page 55.


eXceed/W Setup 3.3.3

You have selected the following configuration for installation:

| | |
|---|---|
| Setup: | Personal |
| Install: | Update |
| Home Directory: | c:\xwin |
| User Directory: | c:\xwin\user |
| Boot Drive: | c |
| Local X: | Local X Client Support, Demos |
| Fonts: | 75dpi fonts |

Please select the transport you are using or plan to use: Windows Sockets API

*Figure 25. Selected Installation Configuration*

9. The installation reads diskettes 1 to 7. You will be prompted to change the diskettes when required. The installation asks you to modify the AUTOEXEC.BAT. It appends the line:

   ```
   path=c:\xwin\ibm;%path%
   ```

   If for any reason (for example, reinstallation) you are not prompted for the AUTOEXEC.BAT you should add this statement into your AUTOEXEC.BAT file or append the C:\install\ path to your existing path statement.

   Then let the installation create a group for your X server and local X clients.

10. Reboot your system and restart Windows.

11. Now you are able to start the basic configuration for the X server. If you want to have more information, click on the Xconfig/W icon and then restore the Transports and Read Me icons.

12. Now you can start with the configuration. The eXceed/W Window is shown in Figure 26 on page 56.

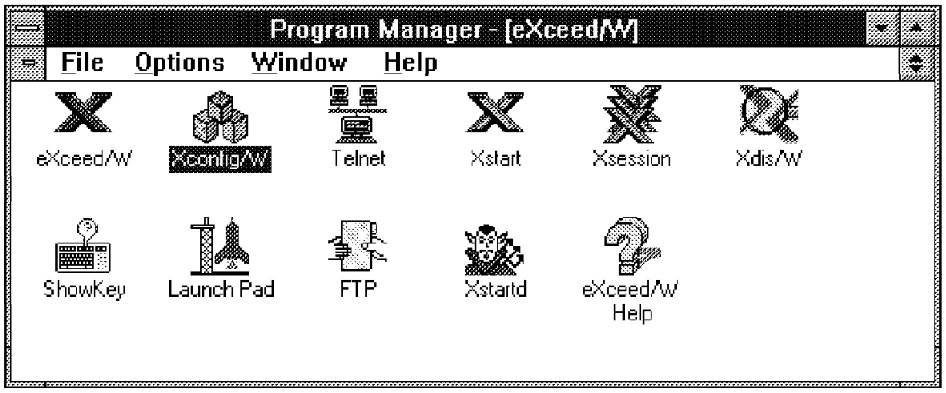


Figure 26. HCL-eXceed/W Window

Restore the Xconfig/W icon and start the configuration by double-clicking on the configuration icon. The configuration window is shown in Figure 27.

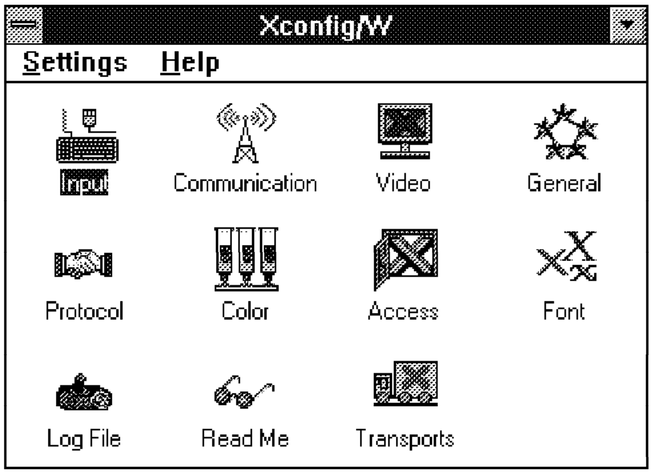


Figure 27. HCL-eXceed/W Configuration Window

Click on the appropriate items to define the following settings:

**Input**

```
Take the default values for US English Keyboards
```

For more information on keyboard remapping see 4.3.1.3, "Remapping the Keyboard" on page 152.

**Communication**

```
Hosts:    c:\tcpdos\etc\hosts:
Startup:  passive
```

**Video**

```
Take the default values (other)
```

**General**

```
Take the default values
```

**Protocol**

```
Take the default values:
Enable:  allow old X11 bugs
Disable: all other selections
```

**Color**

```
Take the default values
```

For more information on customizing Colors see 4.3.1.1, “Customizing Colors” on page 149.

**Access**

```
Take the default values
```

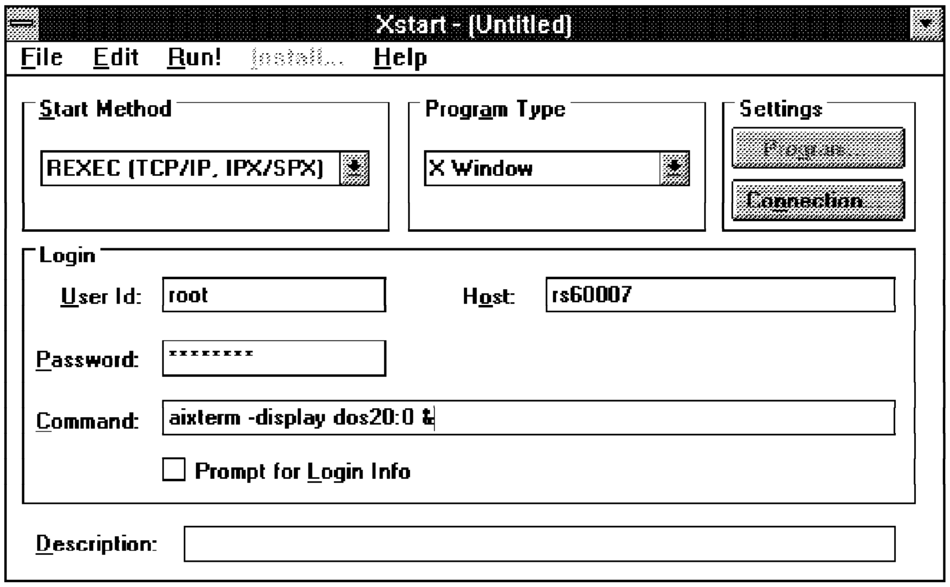
For more information on access control see 4.3.1.4, “Controlling X Client Access” on page 153.

**Font**

```
Take the default values
```

For more information on fonts see 4.3.1.2, “X Fonts” on page 150.

13. Move the cursor to SETTINGS in the Xconfig/W window and select Save.
14. Now you can start the X server by double-clicking on the eXceed/W icon. To initiate a session with the IBM RISC System/6000 restore the Xstart icon and fill in the appropriate fields. In our example we used the settings shown in the following figure:



Start Method: REXEC (TCP/IP, IPX/SPX)
Program Type: X Window
Login
User Id: root
Host: rs60007
Password: ********
Command: aixterm -display dos20:0 &
☐ Prompt for Login Info
Description:

*Figure 28. Xstart*

**Notes:**

1. You can save that window (title bar: File). Then click on Run and you should get an aixterm window on your display. The password will be stored in encrypted form.

2. With Windows you cannot start the Motif Window Manager, because the Microsoft Window Manager controls both the Windows applications and the X clients.

You need not initiate the first X client application with the Xstart icon. You can use telnet from the DOS command line or start the X client application directly on every TCP/IP host with the appropriate software and settings.

To stop the eXceed/W server restore the HCL-eXceed/W icon and select Close. This will also close all clients that are displayed on your screen.

# Chapter 3. X Client Application Considerations

**Warning**

It is outside the scope of this document to serve as an X Window System application programming guide. However, this chapter does provide you with information that will allow you to understand the application environment on each of the IBM X client platforms.

An X client application that uses the Xlib API must be written using C as a programming language. However, the programmer does have a number of choices when deciding how to write an X client application:

1. To use the base Xlib functions directly through the native Xlib interface.
2. To use the X toolkit (Xt) intrinsics which overlay and call the Xlib functions.
3. To use a widget set such as the Athena widget set or the OSF/Motif widget set which makes use of the Xt intrinsics.

Each of the IBM X client products for the MVS, VM and AIX platforms include sample programs that provide the programmer with examples for each of the above cases. The following sections describe how to take your application source and produce an executable load module for each client platform.

## 3.1 Under MVS

IBM provides the C source for three sample programs with the X Window System for MVS:

**XSAMP1X** uses only Xlib functions.

**XSAMP2X** uses the Athena widget set functions.

**XSAMP3X** uses the OSF/Motif widget set functions.

As an application programmer who will write an X client application you will need to understand how to call the functions in either the Xlib, Xt, or widget set libraries. These samples provide examples of how to code these function calls.

When these samples are compiled and link-edited they perform the same function as described for the XSAMP1, XSAMP2, and XSAMP3 in 2.2.1, “Installation Verification for the MVS X Window System API” on page 26.

Also included with the MVS X Window System are a number of the standard MIT X client applications. These provide you with more complex examples of how X client applications can be coded to perform certain functions. There are examples of programs that use Xlib natively or the Athena widget set. For a full list of the supplied MIT X client applications please refer to *TCP/IP Version 2 Release 2.1 for MVS: Programmer's Reference*.

## 3.1.1 Compiling and Link-Editing under MVS

Once you have your X client C program source ready you must compile and link-edit it to produce an executable load module. There is nothing special about using the IBM C/370 compiler and linkage editor to produce an executable X client load module other than ensuring that you access the correct libraries and have the appropriate include statements to resolve all external references during the link-edit stage.

The libraries that you must point to in your SYSLIB DD statement depend upon which X Window functions your application is using. The options are:

- Only Xlib functions
- Xt intrinsics
- Athena widget set functions
- OSF/Motif widget set functions

The libraries are:

- tcpip.SEAZAX11L (Xlib, Xmu and Xext routines)
- tcpip.SEZAOLDX (X Release 10 compatibility routines)
- tcpip.SEZAXTLB (Xt Intrinsics)
- tcpip.SEZAXAWLB (Athena widget set)
- tcpip.SEZAXMLB (OSF/Motif-based widget set)
- tcpip.SEZACMAC (C header files)
- tcpip.SEZACMTX (Sockets routines)
- tcpip.SEZARNT1 (Object data for reentrant programs)

### 3.1.1.1 Using Only Xlib Functions under MVS

Use the following steps as a guide to produce an executable module for XSAMP1X which uses only Xlib functions:

1. You will find the source for XSAMP1X in *tcpip.*SEZAINST where *tcpip* is the high-level data set name qualifier under which the TCP/IP libraries are installed.
2. Allocate three data sets as follows:
   - *userid.*XCLIENT.C for the source code. This should be a fixed block, partitioned data set with a record length of 80 bytes.
   - *userid.*XCLIENT.OBJ for the compiled object code. This should also be a fixed block, partitioned data set with a record length of 80 bytes.
   - *userid.*XCLIENT.LOAD for the executable load modules. This should be a variable length partitioned data set.

   The above data set names are optional but will be assumed for the following discussion. The high-level qualifier *userid* is the user ID of the TSO user who will be compiling, link-editing and running the X client applications.
3. Copy XSAMP1 into *userid.*XCLIENT.C.
4. Code JCL to call and substitute statements for the IBM cataloged procedure EDCC, which is supplied with the C/370 compiler program, and invokes the C compiler. You can find a copy of this procedure in *c370.*SEDCPROC where *c370* is the high-level data set name qualifier under which the C/370 libraries are installed. Refer to Appendix A, "MVS C/370 Catalogued Procedures" on page 163 for a copy of the EDCC cataloged procedure.

An example of the JCL that you would use to call and substitute statements for EDCC to compile XSAMP1 is in Figure 29 on page 61.

```
//PARAVANY JOB 0-111111,MSGCLASS=O,MSGLEVEL=(1,1),REGION=4M,CLASS=I
//*************************************************
//*
//EDCC1    EXEC  EDCC,INFILE='XSAMP1',CPARM='DEF(IBMCPP)'
//*
//*************************************************
//STEPLIB   DD DSN=C370.V2R1M0.SEDCLINK,DISP=SHR
//          DD DSN=PLI.V2R3M0.SIBMLINK,DISP=SHR
//          DD DSN=C370.V2R1M0.SEDCCOMP,DISP=SHR
//SYSIN     DD DSN=PARAVAN.XCLIENT.C(&INFILE),DISP=SHR
//SYSLIB    DD DSN=C370.V2R1M0.SEDCHDRS,DISP=SHR
//          DD DSN=TCPIP.V2R2M1.SEZACMAC,DISP=SHR
//          DD DSN=TCPIP.V2R2M1.SEZAX11L,DISP=SHR
//          DD DSN=TCPIP.V2R2M1.SEZACMTX,DISP=SHR
//SYSLIN    DD DSN=PARAVAN.XCLIENT.OBJ(&INFILE),DISP=SHR
//*
```

*Figure 29. Sample JCL to Compile XSAMP1*

The key points to note about the JCL in Figure 29 are:

a. Set INFILE to equal the name of your application source file.
b. Set CPARM='DEF(IBMCPP)' for the C compiler options.
c. The source code *userid.*XCLIENT.C(XSAMP1) is referenced by the SYSIN DD statement.
d. The compiler output, which is the object module and will be called XSAMP1, is placed in the data set pointed to by the SYSLIN DD statement, *userid.*XCLIENT.OBJ.
e. The SYSLIB DD statement must point to *c370.*SEDCHDRS, which is the library for C headers, *tcpip.*SEZACMAC and SEZACMTX, which contain X client macros and C headers and *tcpip.*SEZAX11L, which contains the Xlib routines.

5. Submit the JCL and ensure that the job runs with a return code of 0. You will then find an object module XSAMP1 in the data set *userid.*XCLIENT.OBJ.

6. Code JCL to call and substitute statements for the IBM cataloged procedure EDCL, which is supplied with the C/370 Compiler program and invokes the linkage editor. You can find a copy of this procedure in *c370.*SEDCPROC. Refer to Appendix A, "MVS C/370 Catalogued Procedures" on page 163 for a copy of the EDCL cataloged procedure.

   An example of the JCL that you would use to call and substitute statements for EDCL to link-edit XSAMP1 is in Figure 30 on page 62.

```
//PARAVANL JOB O-111111,MSGCLASS=O,MSGLEVEL=(1,1),REGION=4M,CLASS=I
//***************************************************
//*
//EDCL1    EXEC  EDCL,INFILE='XSAMP1'
//***************************************************
//SYSLIB    DD DSN=C370.V2R1MO.SEDCBASE,DISP=SHR
//          DD DSN=PLI.V2R3MO.SIBMBASE,DISP=SHR
//          DD DSN=TCPIP.V2R2M1.SEZAX11L,DISP=SHR
//          DD DSN=TCPIP.V2R2M1.SEZACMTX,DISP=SHR
//SYSLIN    DD DSN=PARAVAN.XCLIENT.OBJ(&INFILE),DISP=SHR
//          DD *
  INCLUDE SYSLIB(XMACROS)
  INCLUDE SYSLIB(XLIBINT)
  INCLUDE SYSLIB(XRM)
/*
//SYSLMOD   DD DSN=PARAVAN.XCLIENT.LOAD(&INFILE),DISP=(MOD,PASS)
//*
```

*Figure 30. Sample JCL to Link-Edit XSAMP1*

The key points to note about the JCL in Figure 30 are:

a. Since XSAMP1X only uses X11lib functions, the SYSLIB DD statement must point to the following data sets:

| | |
|---|---|
| *c370.*SEDCBASE | the C/370 library on your system |
| *pli.*SIBMBASE | the PL/1 common library on your system |
| *tcpip.*SEZAX11L | the X11R4 library |
| *tcpip.*SEZACMTX | the common TCP/IP text library |

b. Not all entry points are defined as external references in *tcpip.*SEZAX11L so you must have the INCLUDE statements under your SYSLIN DD statement as illustrated in Figure 30.

7. Submit the JCL and ensure that the job runs with a return code of 0. You will then find a load module XSAMP1 that has been marked executable in the data set *userid.*XCLIENT.LOAD.

8. Ensure that your client host is authorized at the X server and that you have identified the target X server display. Refer to 2.2.1, "Installation Verification for the MVS X Window System API" on page 26 for further details.

9. Run XSAMP1 by typing the following command at the TSO command prompt:

```
CALL 'userid.XCLIENT.LOAD(XSAMP1)'
```

The window that you will see opened at the X server is illustrated in Figure 8 on page 27.

### 3.1.1.2 Using Athena Widget Set Functions under MVS

Use the following steps as a guide to produce an executable module for XSAMP2 which uses the Athena widget set functions:

1. You will find the source for XSAMP2 in *tcpip.*SEZAINST

2. Copy XSAMP2 into *userid.*XCLIENT.C.

3. Use the same JCL that was used for XSAMP1 to compile XSAMP2. Simply code INFILE=XSAMP2. Refer to Figure 29 on page 61 for an example of this JCL.

4. Submit the JCL and ensure that the job runs with a return code of 0. You will then find an object module XSAMP2 in the data set *userid.*XCLIENT.OBJ.

5. The JCL required to link-edit XSAMP2 is different from the JCL used to link-edit XSAMP1 because XSAMP2 uses Athena widget set functions. An example of the JCL used to link-edit XSAMP2 is illustrated in Figure 31.

```
//PARAVANL JOB 0-111111,MSGCLASS=O,MSGLEVEL=(1,1),REGION=4M,CLASS=I
//*************************************************
//*
//EDCL1    EXEC  EDCL,INFILE='XSAMP2'
//*************************************************
//SYSLIB    DD DSN=C370.V2R1M0.SEDCBASE,DISP=SHR
//          DD DSN=PLI.V2R3M0.SIBMBASE,DISP=SHR
//          DD DSN=TCPIP.V2R2M1.SEZAXTLB,DISP=SHR
//          DD DSN=TCPIP.V2R2M1.SEZAX11L,DISP=SHR
//          DD DSN=TCPIP.V2R2M1.SEZACMTX,DISP=SHR
//SYSLIN    DD DSN=PARAVAN.XCLIENT.OBJ(&INFILE),DISP=SHR
//          DD *
  INCLUDE SYSLIB(XMACROS)
  INCLUDE SYSLIB(XLIBINT)
  INCLUDE SYSLIB(XRM)
  INCLUDE SYSLIB(CALLBACK)
  INCLUDE SYSLIB(CONVERT)
  INCLUDE SYSLIB(CONVERTE)
  INCLUDE SYSLIB(INTRINSI)
  INCLUDE SYSLIB(DISPLAY)
  INCLUDE SYSLIB(ERROR)
  INCLUDE SYSLIB(EVENT)
  INCLUDE SYSLIB(NEXTEVEN)
  INCLUDE SYSLIB(TMSTATE)
  INCLUDE SYSLIB(ASCTEXT)
  INCLUDE SYSLIB(ATOMS)
  INCLUDE SYSLIB(ATEXT)
/*
//SYSLMOD   DD DSN=PARAVAN.XCLIENT.LOAD(&INFILE),DISP=SHR
//*
```

*Figure 31. Sample JCL to Link-Edit XSAMP2*

The key points to note about the JCL in Figure 31 are:

a. Since XSAMP2 uses Athena widget set functions, the SYSLIB DD statement must point to the following data sets:

| | |
|---|---|
| *c370.*SEDCBASE | the C/370 library on your system |
| *pli.*SIBMBASE | the PL/1 common library on your system |
| *tcpip.*SEZAXAWL | the Athena widget set library |
| *tcpip.*SEZAXTLB | the X Toolkit intrinsics library |
| *tcpip.*SEZAX11L | the X11R4 library |
| *tcpip.*SEZACMTX | the common TCP/IP text library |

   b. Not all entry points are defined as external references in *tcpip.*SEZAXAWL, *tcpip.*SEZAXTLB, and *tcpip.*SEZAX11L. You must therefore have the INCLUDE statements under your SYSLIN DD statement as illustrated in Figure 31.

6. Submit the JCL and ensure that the job runs with a return code of 0. You will then find a load module XSAMP2 that has been marked executable in the data set *userid.*XCLIENT.LOAD.

7. Ensure that your client host is authorized at the X server and that you have identified the target X server display. Refer to 2.2.1, "Installation Verification for the MVS X Window System API" on page 26 for further details.

8. Run XSAMP2 by typing the following command at the TSO command prompt:

   ```
   CALL 'userid.XCLIENT.LOAD(XSAMP2)'
   ```

   The window that you will see opened at the X Server is illustrated in Figure 9 on page 27.

#### 3.1.1.3 Using OSF/Motif Widget Set Functions under MVS

Use the following steps as a guide to produce an executable module for XSAMP3 which uses the OSF/Motif widget set functions:

1. Find the source for XSAMP3 in *tcpip.*SEZAINST.

2. Copy XSAMP3 into *userid.*XCLIENT.C.

3. Use the same JCL that was used for XSAMP1 and XSAMP2 to compile XSAMP3. Simply code INFILE=XSAMP3. Refer to Figure 29 on page 61 for an example of this JCL.

4. Submit the JCL and ensure that the job runs with a return code of 0. You will then find an object module XSAMP3 in the data set *userid.*XCLIENT.OBJ.

5. The JCL required to link-edit XSAMP3 is different from the JCL used to link-edit XSAMP1 and XSAMP2 because XSAMP3 uses the OSF/Motif widget set functions. An example of the JCL used to link-edit XSAMP3 is illustrated in Figure 32 on page 65.

```
//PARAVANL JOB O-111111,MSGCLASS=O,MSGLEVEL=(1,1),REGION=4M,CLASS=I
//**************************************************
//*
//EDCL1    EXEC  EDCL,INFILE='XSAMP3'
//**************************************************
//SYSLIB    DD DSN=C370.V2R1M0.SEDCBASE,DISP=SHR
//          DD DSN=PLI.V2R3M0.SIBMBASE,DISP=SHR
//          DD DSN=TCPIP.V2R2M1.SEZAXMLB,DISP=SHR
//          DD DSN=TCPIP.V2R2M1.SEZAXTLB,DISP=SHR
//          DD DSN=TCPIP.V2R2M1.SEZAX11L,DISP=SHR
//          DD DSN=TCPIP.V2R2M1.SEZACMTX,DISP=SHR
//SYSLIN    DD DSN=PARAVAN.XCLIENT.OBJ(&INFILE),DISP=SHR
//          DD *
  INCLUDE SYSLIB(XMACROS)
  INCLUDE SYSLIB(XLIBINT)
  INCLUDE SYSLIB(XRM)
  INCLUDE SYSLIB(CALLBACK)
  INCLUDE SYSLIB(CONVERT)
  INCLUDE SYSLIB(CONVERTE)
  INCLUDE SYSLIB(INTRINSI)
  INCLUDE SYSLIB(DISPLAY)
  INCLUDE SYSLIB(ERROR)
  INCLUDE SYSLIB(EVENT)
  INCLUDE SYSLIB(NEXTEVEN)
  INCLUDE SYSLIB(TMSTATE)
  INCLUDE SYSLIB(ATOMS)
  INCLUDE SYSLIB(CUTPASTE)
  INCLUDE SYSLIB(FILESB)
  INCLUDE SYSLIB(GEOUTILS)
  INCLUDE SYSLIB(LIST)
  INCLUDE SYSLIB(MANAGER)
  INCLUDE SYSLIB(PRIMITIV)
  INCLUDE SYSLIB(RESIND)
  INCLUDE SYSLIB(ROWCOLUM)
  INCLUDE SYSLIB(MSELECTI)
  INCLUDE SYSLIB(TEXT)
  INCLUDE SYSLIB(TEXTF)
  INCLUDE SYSLIB(TRAVERSA)
  INCLUDE SYSLIB(VISUAL)
  INCLUDE SYSLIB(XMSTRING)
/*
//SYSLMOD   DD DSN=PARAVAN.XCLIENT.LOAD(&INFILE),DISP=(MOD,PASS)
//*
```

*Figure 32. Sample JCL to Link-Edit XSAMP3*

The key points to note about the JCL in Figure 32 are:

a. Since XSAMP3 uses OSF/Motif functions, the SYSLIB DD statement must point to the following data sets:

| | |
|---|---|
| *c370*.SEDCBASE | the C/370 library on your system |
| *pli*.SIBMBASE | the PL/1 common library on your system |
| *tcpip*.SEZAXMLB | the OSF/Motif widget set library |
| *tcpip*.SEZAXTLB | the X Toolkit intrinsics library |
| *tcpip*.SEZAX11L | the X11R4 library |

*tcpip*.SEZACMTX    the common TCP/IP text library

   b. Not all entry points are defined as external references in *tcpip*.SEZAXMLB, *tcpip*.SEZAXTLB, and *tcpip*.SEZAX11L. You must therefore have the INCLUDE statements under your SYSLIN DD statement as illustrated in Figure 32 on page 65.

6. Submit the JCL and ensure that the job runs with a return code of 0. You will then find a load module XSAMP3X that has been marked executable in the data set *userid*.XCLIENT.LOAD.

7. Ensure that your client host is authorized at the X server and that you have identified the target X server display. Refer to 2.2.1, "Installation Verification for the MVS X Window System API" on page 26 for further details.

8. Run XSAMP3 by typing the following command at the TSO command prompt:

   ```
   CALL 'userid.XCLIENT.LOAD(XSAMP3)'
   ```

   The window that you will see opened at the X server is illustrated in Figure 10 on page 27.

#### 3.1.1.4 Standard MIT X Client Applications under MVS

A number of the standard MIT X client applications are provided with the X Window System under MVS. Information on how to compile and link-edit these applications can be found in the member *tcpip*.INSTALL(I5735HAL). You can also find help files on operating each of the MIT X client applications in *tcpip*.INSTALL.

We compiled and tested three of these applications, all of which use the Athena widget set:

- XLOGO
- XCLOCK
- XCALC

The steps described here refer to compiling and link-editing these three applications. Although these steps are very similar to the steps described for compiling and link-editing XSAMP2 in 3.1.1.2, "Using Athena Widget Set Functions under MVS" on page 62, they are included for completeness.

1. You will find the source for XLOGO, XCLOCK and XCALC in in *tcpip*.SEZAINST.

2. It is assumed that you have allocated three data sets as follows:

   - *userid*.XCLIENT.C for the source code. This should be a fixed block, partitioned data set with a record length of 80 bytes.
   - *userid*.XCLIENT.OBJ for the compiled object code. This should also be a fixed block, partitioned data set with a record length of 80 bytes.
   - *userid*.XCLIENT.LOAD for the executable load modules. This should be a variable length partitioned data set.

   The above data set names are optional but will be assumed for the following discussion.

3. Copy XLOGO, XCLOCK and XCALC into *userid*.XCLIENT.C.

   In order to produce a load module for XCALC you will need to compile two additional modules and link-edit them with XCALC. These are the modules

ACTIONS and MATH. Copy these from *tcpip.*SEZALINK into *userid.*XCLIENT.C.

4. Use the JCL as shown in Figure 29 on page 61 to compile XLOGO, XCLOCK, XCALC, ACTIONS and MATH. The only change you need to make is to specify the source file for the INFILE parameter. Ensure that each job runs with a return code of 0. You will then find the object modules XLOGO, XCLOCK, XCALC, ACTIONS and MATH in the data set *userid.*XCLIENT.OBJ.

5. Use the JCL as shown in Figure 30 on page 62 to link-edit XLOGO and XCLOCK. The only change you need to make is to specify the source file for the INFILE parameter.

   The JCL must be modified to link-edit XCALC because this module needs to be link-edited with ACTIONS and MATH. You need to point these to object modules in the SYSLIN DD statement. This JCL is shown in Figure 33.

   Ensure each job runs with a return code of 0. You will then find the load modules XLOGO, XCLOCK, and XCALC in the data set *userid.*XCLIENT.LOAD.

```
//PARAVANL JOB 0-111111,MSGCLASS=O,MSGLEVEL=(1,1),REGION=4M,CLASS=I
//**************************************************
//*
//EDCL1    EXEC  EDCL,INFILE='XCALC'
//**************************************************
//SYSLIB    DD DSN=C370.V2R1M0.SEDCBASE,DISP=SHR
//          DD DSN=PLI.V2R3M0.SIBMBASE,DISP=SHR
//          DD DSN=TCPIP.V2R2M1.SEZAXAWL,DISP=SHR
//          DD DSN=TCPIP.V2R2M1.SEZAXTLB,DISP=SHR
//          DD DSN=TCPIP.V2R2M1.SEZAX11L,DISP=SHR
//          DD DSN=TCPIP.V2R2M1.SEZACMTX,DISP=SHR
//SYSLIN    DD DSN=PARAVAN.XCLIENT.OBJ(&INFILE),DISP=SHR
//          DD DSN=PARAVAN.XCLIENT.OBJ(ACTIONS),DISP=SHR
//          DD DSN=PARAVAN.XCLIENT.OBJ(MATH),DISP=SHR
//          DD *
  INCLUDE SYSLIB(XMACROS)
  INCLUDE SYSLIB(XLIBINT)
  INCLUDE SYSLIB(XRM)
  INCLUDE SYSLIB(CALLBACK)
  INCLUDE SYSLIB(CONVERT)
  INCLUDE SYSLIB(CONVERTE)
  INCLUDE SYSLIB(INTRINSI)
  INCLUDE SYSLIB(DISPLAY)
  INCLUDE SYSLIB(ERROR)
  INCLUDE SYSLIB(EVENT)
  INCLUDE SYSLIB(NEXTEVEN)
  INCLUDE SYSLIB(TMSTATE)
  INCLUDE SYSLIB(ASCTEXT)
  INCLUDE SYSLIB(ATOMS)
  INCLUDE SYSLIB(ATEXT)
/*
//SYSLMOD   DD DSN=PARAVAN.XCLIENT.LOAD(&INFILE),DISP=SHR
//*
```

*Figure 33. Sample JCL to Link-Edit XCALC*

6. Ensure that your client host is authorized at the X server and that you have identified the target X server display. Refer to 2.2.1, “Installation Verification for the MVS X Window System API” on page 26 for further details.

7. Run XLOGO by typing the following command at the TSO command prompt:

```
CALL 'userid.XCLIENT.LOAD(XLOGO)'
```

The window that you will see opened at the X server is illustrated in Figure 34.

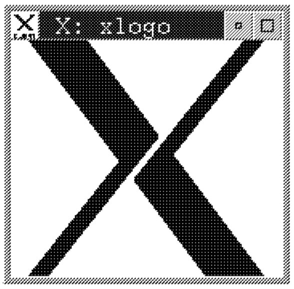


*Figure 34. Display at an OS/2 X Server for MIT XLOGO X Client Program*

8. You can end XLOGO by closing the window at the X server.
9. Run XCLOCK by typing the following command at the TSO command prompt:

```
CALL 'userid.XCLIENT.LOAD(XCLOCK)'
```

The window that you will see opened at the X server is illustrated in Figure 35. Hopefully the clock will display your local MVS system time!

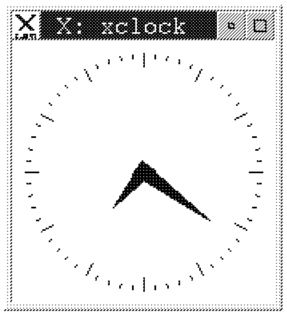


*Figure 35. Display at an OS/2 X Server for MIT XCLOCK X Client Program*

10. You can end XCLOCK by closing the window at the X server.
11. Before running XCALC you need to set up a file that specifies values for the application resources. This is done using an application resource file.

    IBM provides an application resource file with XCALC that will allow you to run the program. This file is *tcpip.*INSTALL(XXCALC). Refer to Appendix B, “Supplied Application Resource File Definitions for XCALC” on page 167 for a copy of the contents of this file. Perform the following tasks to use this as the application resource file.

    a. Allocate a data set called *userid.*X.DEFAULTS which should be a sequential, fixed block data set with a record length of 80 bytes.
    b. Copy *tcpip.*INSTALL(XXCALC) into *userid.*X.DEFAULTS.
12. Run XCALC by typing the following command at the TSO command prompt:

```
CALL 'userid.XCLIENT.LOAD(XCALC)'
```

The window that you will see opened at the X server is illustrated in Figure 36.

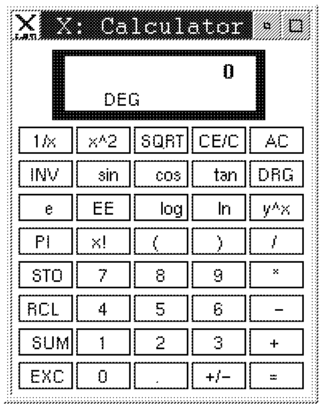


*Figure 36. Display at an OS/2 X Server for MIT XCALC X Client Program*

13. You can end XCALC by closing the window at the X server.

## 3.1.2 MVS Application Resource File

Once you have an executable load module for your X client application, it is possible to change certain characteristics of the application without having to go back and modify, re-compile and link-edit the source code. This is done through an application resource file.

Resources are defined within an application and set to particular values to specify the characteristics of the application when displayed at an X server. These characteristics can include application window size, position on the screen, color, fonts and other functional details.

An application resource file allows you to change these application characteristics at run time. Within this file it is possible to specify values for resources that will override the values that are coded within the application.

Under MVS the application resource file is in the data set *userid.*X.DEFAULTS. The following is an example that you can use as a guide to building an application resource file under MVS.

### 3.1.2.1 Building an Application Resource File Under MVS

As discussed in 3.1.1.4, "Standard MIT X Client Applications under MVS" on page 66 we compiled and link-edited some MIT X client applications that are supplied with the X Window System for MVS. Both XLOGO and XCLOCK can be run without an application resource file. When you execute either XLOGO or XCLOCK under TSO and direct the output to an X server display each application takes certain defaults for its resources.

XCALC is a little different in that it is an example of an X client application that requires an application resource file to provide resource values to operate. This is because the resources are not specified within the program itself. However, the principle is exactly the same.

We performed the following steps to build an application resource file to override the default values for resources and change the characteristics of XLOGO when displayed at the X server.

1. When building an application resource file, the first thing you need to do is to understand the application resources that are available for modification. Normally you would refer to the documentation for a program, or look at the program source code.

   If you look in the member *tcpip.*SEZAINST(HLPXLOGO) you will find a number of application resources that can be modified for XLOGO. These are described in Table 8.

*Table 8. Application Resources for XLOGO*

| Resource | Description |
|---|---|
| Width | This specifies the width of the XLOGO window in pixels. The application default is 100 pixels. |
| Height | This specifies the height of the XLOGO window in pixels. The application default is 100 pixels. |
| Foreground | This specifies the color of the X in the XLOGO. The application default is black. |
| Background | This specifies the background color. The application default is white. |

2. Ensure that your client host is authorized at the X server and that you have identified the target X server display. Refer to 2.2.1, “Installation Verification for the MVS X Window System API” on page 26 for further details.

3. Run XLOGO by typing the following command at the TSO command prompt:

   ```
   CALL 'userid.XCLIENT.LOAD(XLOGO)'
   ```

   The window that you will see opened at the X server is illustrated in Figure 34 on page 68. Notice the default dimensions and colors used to draw the X.

4. Close down the XLOGO window at the X server to end the program.

5. Allocate a data set called *userid.*X.DEFAULTS which should be a sequential, fixed block data set with a record length of 80 bytes. Edit this file to change the values of some of the application resources for XLOGO. An example of an application resource file is illustrated in Figure 37.

   If you already have an existing *userid.*X.DEFAULTS file for another X client application, then simply append the lines for XLOGO to the existing definitions.

```
XLogo*width: 300
XLogo*height: 300
XLogo*foreground: blue
XLogo*background: red
```

*Figure 37. Example of the Contents of the Application Resource File for XLOGO*

In this example we have set the width and the height of the XLOGO window to 300 pixels. We have also set the color of the X to blue and the background color to red.

**Note:** The application resource file is case sensitive. You must type the resource names exactly as they are specified for the application.

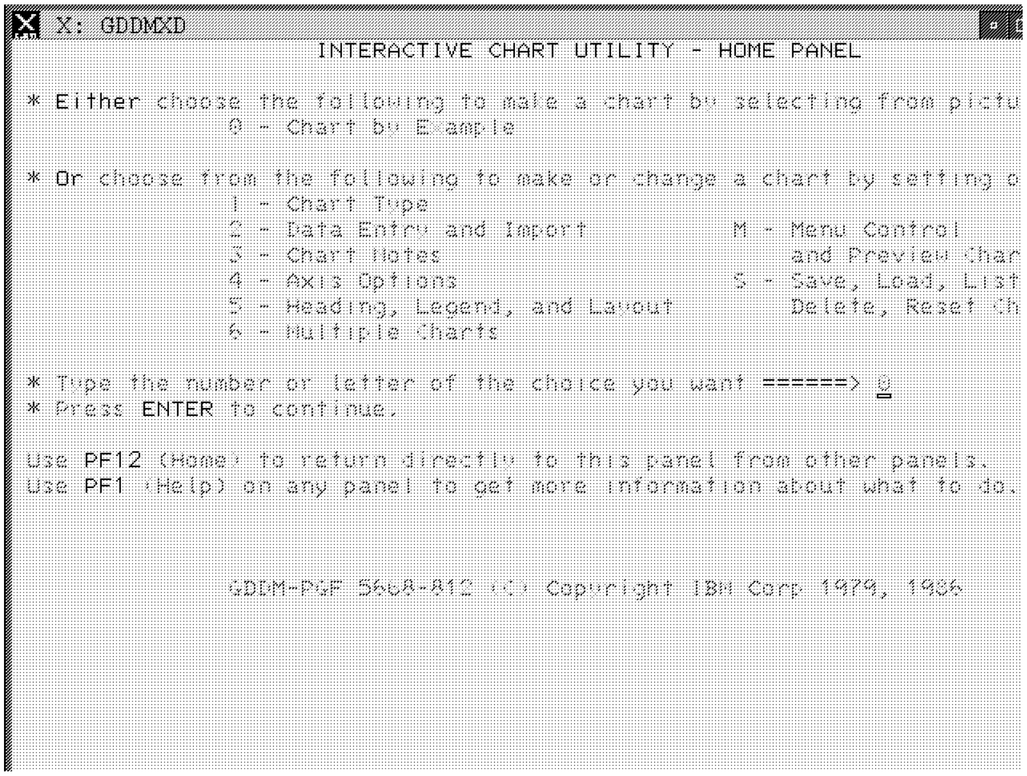
6. Run XLOGO again.
7. Notice the modified characteristics of XLOGO window.

### 3.1.3 Using GDDM Applications under MVS

As with user-written applications, it is possible to modify the runtime characteristics for GDDM applications by specifying resource values in the application resource file *userid*.X.DEFAULTS. In addition, it is possible to specify a keyboard map that allows you to use an APL2 keyboard at the X server.

An example of a GDDM application that is commonly used under TSO is ADMCHART. We tested ADMCHART with the X Window System GDDM interface.

After starting the MVS X Window System GDDM interface as described in 2.2.2.2, "Installation Verification" on page 29 and identifying an OS/2 X server as the target display, we started ADMCHART from the TSO command line. ADMCHART was able to display its panels at the OS/2 X server. An example of the ADMCHART home panel is illustrated in Figure 38.



```
X: GDDMXD
                          INTERACTIVE CHART UTILITY - HOME PANEL

 * Either choose the following to male a chart by selecting from pictu
                0 - Chart by Example

 * Or choose from the following to make or change a chart by setting o
                1 - Chart Type
                2 - Data Entry and Import          M - Menu Control
                3 - Chart Notes                        and Preview Char
                4 - Axis Options                   S - Save, Load, List
                5 - Heading, Legend, and Layout        Delete, Reset Ch
                6 - Multiple Charts

 * Type the number or letter of the choice you want ======> 0
 * Press ENTER to continue.

 Use PF12 (Home) to return directly to this panel from other panels.
 Use PF1 (Help) on any panel to get more information about what to do.



                GDDM-PGF 5668-812 (C) Copyright IBM Corp 1979, 1986
```

*Figure 38. An MVS X Window System GDDM Interface Window for ADMCHART*

The ADMCHART panel shown in Figure 38 is partly obscured or clipped. This window has been opened at an OS/2 X server that had an 8513 screen. The maximum window size possible on this screen is not big enough to accommodate the minimum GDDM graphics display area specified by the MVS X

Window System GDDM interface. Please refer to 3.1.3.1, "Application Resource File for GDDM under MVS" on page 72 for more information.

In order to interact with ADMCHART using the X server keyboard you will need to alter the keyboard mapping for the Enter key. We discovered that the default keyboard mappings provided with the X Window Systems for both OS/2 and AIX had the Enter key set to the keysym name *RETURN*. You need to remap this key's keysym name to *EXECUTE*. Please refer to 4.2.2.1, "Remapping the Keyboard under OS/2" on page 128 for a description of how to perform the remapping with OS/2 and 4.1.5, "Remapping the Keyboard Under AIX/6000" on page 123 for AIX.

**Note**

When using ADMCHART you will notice that after a number of inputs from the keyboard at the X server (approximately 20), the keyboard locks up. This is because the TSO screen goes into a *holding* state. In order to free up the keyboard you need to go to the TSO session at the 3270 terminal and hit the Enter key a number of times to allow the screen to scroll through the messages associated with the X server keystrokes.

**This is the case for all X client applications under TSO**.

### 3.1.3.1 Application Resource File for GDDM under MVS

In Chapter 10 of the *TCP/IP Version 2 Release 2.1 for MVS: User's Guide* you will find a description of the application resources for which you can define values to specify the characteristics for GDDM applications that use the MVS X Window System GDDM interface. Those resources are listed in Table 9.

*Table 9 (Page 1 of 2). Application Resources for GDDM under MVS*

| Resource | Description |
|---|---|
| CMap | Specifies whether to load the default color map or not. This is specified as *N* if the color map at the X server is going to be used as opposed to the default color map that otherwise would be loaded by the MVS X Window System GDDM interface. |
| GColornn | Specifies the GDDM color that is mapped onto an X Window System color. |
| Geometry | Specifies the size and location of the initial GDDM window at the X server. The size of the GDDM graphics display area is dependent upon the size of the window. There are in fact only two possible GDDM display area sizes available. If the width of the window is specified as less that 1000 pixels, then the size of the GDDM display area will be 720 pixels high by 512 pixels wide; otherwise it will be 1200 pixels wide by 864 pixels high. If the window is specified as less than 720 pixels by 512 pixels then the GDDM display area will be clipped. |
| GMCPnn | Specifies the GDDM multicolor pattern that is mapped onto an X Window System color. |
| HostRast | Specifies whether the raster processing is done at the host or the workstation. The default for this option is *N*, which means that the raster processing is done at the workstation. You should choose *Y* when the GDDM application involves multiplane symbol sets or GDDM color mixing. |

*Table 9 (Page 2 of 2). Application Resources for GDDM under MVS*

| Resource | Description |
|---|---|
| XCIConn | Specifies whether the X Window System GDDM interface should close the window at the X server when the GDDM application finishes. The default is *Y*, which means the window will close when the GDDM application finishes. *N* means that the window will be left open and will continue to display the last GDDM graphics window displayed at the workstation even though the application has closed down. The only way to close this window is at the X server itself. |
| XSync | Specifies that the X Window System GDDM interface send one request at a time to the X server (operate in a synchronous mode) when set to *Y*. The default is that the interface communicate with the X server asynchronously which is more common for the X protocol. |
| ZWL | Specifies that the server use the fastest drawing algorithm to draw lines. The default is *N* because if this option is used it may result in lines that are not the width they were intended to be by the application. |

You specify these resource options in the application resource file *userid*.X.DEFAULTS. You simply edit this file and enter the X Window System GDDM interface resource options after existing entries that you may have for other X client applications. Please refer to 3.1.3.1, “Application Resource File for GDDM under MVS” on page 72 for further details on the application resource file under MVS.

A sample application resource file for GDDM applications is provided with the X Window System for MVS. You will find it in the member *tcpip*.SEZAINST(XDEFAULT).

An example of application resource file entries for a GDDM application is shown in Figure 39. Note that the application resource file is case sensitive and so the variables must be coded exactly as shown.

```
gddmx*CMap: Y
gddmx*GColor1: Purple
gddmx*GColor2: Orange
gddmx*GColor4: White
gddmx*Geometry: 1200x864+50+50
gddmx*HostRast: N
gddmx*XCIConn: Y
gddmx*XSync: N
gddmx*ZWL: N
```

*Figure 39. Example Application Resource File Entries for GDDM Applications*

The application resource file entries shown in Figure 39 specify the following:

| | |
|---|---|
| **gddmx*CMap: Y** | directs the X Window System GDDM interface to load the default color map. |
| **gddmx*GColor1: Purple** | specifies that the X Window color purple should be displayed for the GDDM color blue. |

|  |  |
|---|---|
| **gddmx*GColor2: Orange** | specifies that the X Window color orange should be displayed for the GDDM color red. |
| **gddmx*GColor4: White** | specifies that the X Window color white should be displayed for the GDDM color green. |
| **gddmx*Geometry: 1200x900+50+50** | specifies a window at the X server that will be 1200 pixels wide by 900 pixels high and will be 50 pixels in the X direction and 50 pixels in the Y direction from the top left hand corner of the display. This means that the GDDM graphics display area will be 1200 pixels by 864 pixels. |
| **gddmx*HostRast: N** | directs that raster processing be done at the workstation. |
| **gddmx*XClConn: Y** | specifies that the connection to the X server be closed when the application is closed. |
| **gddmx*XSync: N** | specifies that the communication to the X server be asynchronous. |
| **gddmx*ZWL: N** | specifies that line widths will be displayed at normal width. |

We set these options in *userid.*X.DEFAULTS and started up the MVS X Window System GDDM interface as described in 2.2.2.2, “Installation Verification” on page 29. We identified an OS/2 X server as the target display and started ADMCHART from the TSO command line. A window like the one illustrated in Figure 40 was opened at the OS/2 X server.

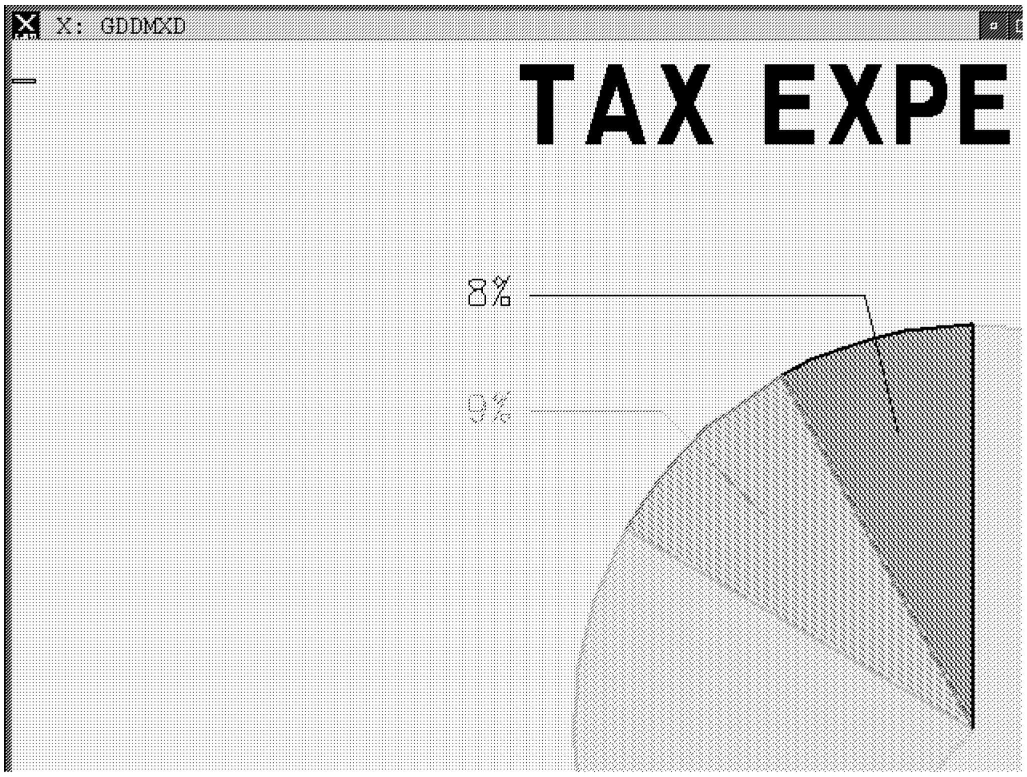


*Figure 40. Display at an OS/2 X Server for ADMCHART under MVS*

Remember that we set the geometry for this window in the application resource file as 1200 pixels wide and 900 pixels high. This was far too big for the 8513 screen so PMX, the OS/2 X Window Manager, made the window as large as the display will permit. However, because we set the window to greater than 1000 pixels wide, the GDDM graphics display area has still been set to 1200 pixels wide by 864 pixels high.

Even when we set the window geometry at 720 pixels wide and 512 pixels wide, the GDDM graphics display area is too large for an 8513 screen. An example of a window with this geometry is illustrated in Figure 38 on page 71.

Under PMX we re-focused the window by clicking on the button at the right of the window title bar used to maximize the window. The resulting window is illustrated in Figure 42 on page 76. Notice that the GDDM graphics display area is now much smaller than the default minimum of 720 pixels wide by 512 pixels high. On a 8514 display we had good results with the following geometry data: the whole graphic could be displayed.

```
gddmx*CMap: Y
gddmx*GColor1: Purple
gddmx*GColor2: Orange
gddmx*GColor4: White
gddmx*Geometry: 750x650+10+10
gddmx*HostRast: N
gddmx*XCIConn: Y
gddmx*XSync: N
gddmx*ZWL: N
```

*Figure 41. Modified Application Resource File Entries for 8514 Display*

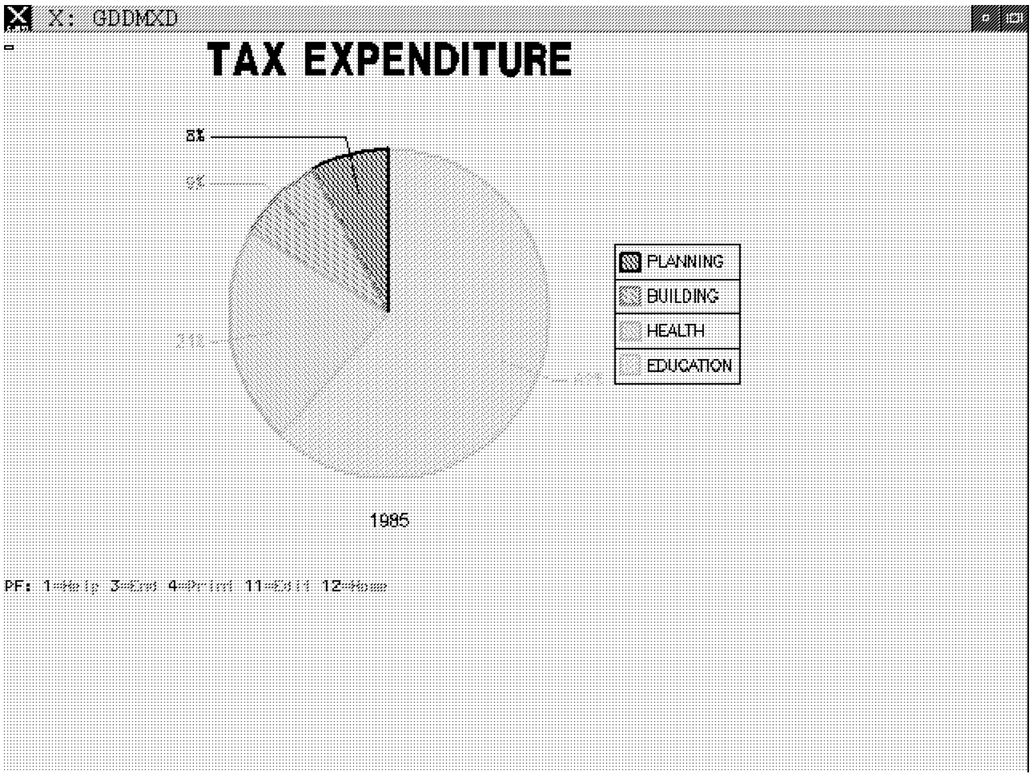


*Figure 42. Re-focused Display at an OS/2 X Server for ADMCHART*

### 3.1.3.2 APL2 Character Set Keyboard for GDDM under MVS

Unfortunately, GDDM applications that require APL2 characters to be input from a keyboard are not classic X Window System applications because they are not truly device independent. These applications are designed to operate with an IBM 3179-G display and an APL2 keyboard where physical keys correspond to specific character representations.

The X Window System GDDM interface for MVS provides a mechanism which allows the keyboard at an X server to be used as an APL2 keyboard. A map is provided which allows the X Window System GDDM interface to accept keystrokes from the X server and translate them into valid APL2 data for the X client application.

When an application is started it initializes the X Window System GDDM interface, which looks for a data set called *userid.*GDXALTCS.PSS where *userid* is the TSO user ID under which the X client application is started. This data set provides the default mapping for the primary and alternate character sets associated with an IBM 3179-G display.

Use the following steps as a guide to setting up this data set:

1. You can find this mapping in the supplied member *tcpip.*SEZAINST(GDXALTCS) where *tcpip* is the high-level qualifier under which TCP/IP has been installed under MVS.
2. Allocate a data set called *userid.*GDXALTCS.PSS which should be a sequential fixed block data set with a record length of 80 bytes.
3. Copy the contents of *tcpip.*SEZAINST(GDXALTCS) into *userid.*GDXALTCS.PSS.

In an X Window environment, each physical key is associated with a keycode. The MVS X Window System GDDM interface uses *userid.*GDXALTCS.PSS to map keycodes into characters that are then sent to the GDDM application.

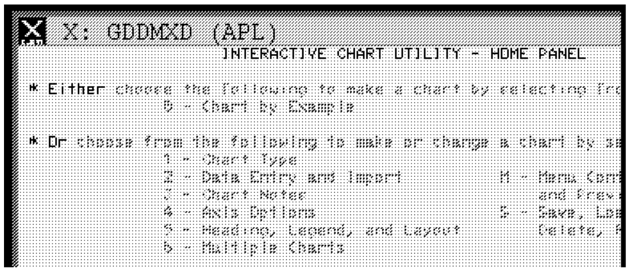
Characters are represented by keycodes in one of the following ways:

1. A single keycode only.
2. A keycode with a modifier keycode.

The two modifier keys that are used are the Shift and the Alt keys.

Once you have *userid.*GDXALTCS.PSS in place, you will notice that by pressing the Backspace key together with the Alt key you will toggle on the APL2 character set. This is indicated by the characters (APL) in the title bar of the window at the X server as shown in Figure 43. This now means that you are able to use the APL2 character set at your X server keyboard.

**Note:** You only need to install *userid.*GDXALTCS.PSS if you are planning to use the APL2 character set.



```
X: GDDMXD (APL)
                    INTERACTIVE CHART UTILITY - HOME PANEL

* Either choose the following to make a chart by selecting fro
            0 - Chart by Example

* Or choose from the following to make or change a chart by se
            1 - Chart Type
            2 - Data Entry and Import          M - Menu Con
            3 - Chart Notes                        and Prev
            4 - Axis Options                   S - Save, Lo
            5 - Heading, Legend, and Layout        Delete, R
            6 - Multiple Charts
```

*Figure 43. APL2 Character Set Indicator on at the X Server Window Title Bar*

The mapping in *userid.*GDXALTCS.PSS assumes a keyboard at the X server that is equivalent to the IBM 101-key Enhanced Keyboard. What this actually means is that it is assumed that the physical key represented by a particular keycode has the appropriate primary and alternate characters on that key as you would find on an IBM 3179-G 101-key Enhanced Keyboard. Where this is not the case it is possible to override the default mapping provided by *userid.*GDXALTCS.PSS.

If the keyboard at your X server is not set out such that characters correspond to particular keycodes as expected by the MVS X Window System GDDM interface, then you can use the following steps as a guide to providing a customized mapping for your keyboard:

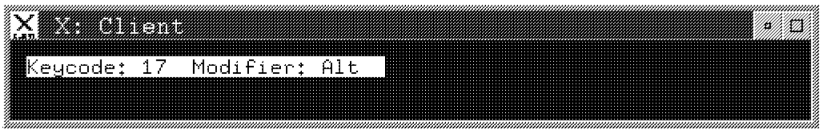
1. The first thing you need to know is the keycode associated with each physical key on your keyboard. Provided with the X Window System GDDM interface is an X client program called KEYCODE. When invoked this program opens a window at your server which will allow you to determine the keycode for each key that is pressed. Invoke KEYCODE as follows:

   a. Ensure that the X server has authorized the MVS host as an X client and that the X client target display variable has been set in the data set *userid.*XWINDOWS.DISPLAY to identify the X server target display. Refer to 2.2.1, "Installation Verification for the MVS X Window System API" on page 26 for more information on the data set *userid.*XWINDOWS.DISPLAY.

   b. From the TSO command line type:

```
KEYCODE
```

c. KEYCODE will open a window at your X server. Press any key and notice the corresponding keycode. Figure 44 shows the KEYCODE window after the Alt and Backspace keys have been pressed.



```
Keycode: 17  Modifier: Alt
```

Figure 44. X Client KEYCODE Display for the ALT BACKSPACE Key Sequence

The above display shows that the keycode for the Backspace key is hexadecimal 17 and the Alt key is the modifier.

2. IBM provides a sample APL2 character set map in *tcpip.*SEXAINST(GDXAPLCS). Allocate the data set *userid.*GDXAPLCS.MAP which should be a sequential fixed block data set with a record length of 80 bytes.

3. Copy the contents of *tcpip.*SEZAINST(GDXAPLCS) into *userid.*GDXAPLCS.MAP. When the MVS X Window System GDDM interface is initialized it looks for *userid.*GDXAPLCS.MAP, which it will use to override the default APL2 character set mappings.

4. Based on the keycodes you obtained from the KEYCODE application you can edit *userid.*GDXAPLCS.MAP and change the character codes associated with each keycode to match your keyboard. A section of the contents of *userid.*GDXAPLCS.MAP is illustrated in Figure 45.

```
0a  00  f1  08  72  08  da
0b  00  f2  08  a0  08  fb
0c  00  f3  00  4c  08  dc
0d  00  f4  08  8c  08  dd
0e  00  f5  00  7e  08  cd
0f  00  f6  08  ae  08  cf
10  00  f7  00  6e  08  ed
11  00  f8  08  be  08  fd
12  00  f9  08  78  08  cb
13  00  f0  08  71  08  ca
14  00  4e  00  60  08  db
15  08  b6  08  b8  08  ee
19  00  d8  00  6f  08  58
1a  00  e6  08  b4  08  66
1b  00  c5  08  b1  08  45
1c  00  d9  08  b3  08  59
1d  00  e3  08  80  08  63
```

Figure 45. Example of the Contents of tcpip.GDXAPLCS.MAP

The column values in the map in Figure 45 have the following meaning:

**Column 1** is the hexadecimal keycode for the physical key.

| | |
|---|---|
| **Column 2** | defines whether the character is in the primary or alternate character set when the key corresponding to the keycode is pressed alone. 0 means the character is in the primary set while 8 means the alternate character set. |
| **Column 3** | is the EBCDIC code for the character in the character set when the key corresponding to the keycode is pressed alone. |
| **Column 4** | defines whether the character is in the primary or alternate character set when the key corresponding to the keycode and the Shift key are pressed together. 0 means the character is in the primary set while 8 means the alternate character set. |
| **Column 5** | is the EBCDIC code for the character in the character set when the key corresponding to the keycode and the Shift key are pressed together. |
| **Column 6** | defines whether the character is in the primary or alternate character set when the key corresponding to the keycode and the Alt key are pressed together. 0 means the character is in the primary set while 8 means the alternate character set. |
| **Column 7** | is the EBCDIC code for the character in the character set when the key corresponding to the keycode and the Alt key are pressed together. |

The primary character set is available when the APL2 character set is toggled either on or off. The alternate character set is only available when the APL2 character set is toggled on. Consider the first row in the table as an example:

| | |
|---|---|
| **0a** | is the keycode for what is normally the 1 key. |
| **00** | means that when this key is pressed alone the character represented by the code in column 3 (in this case f1) will be sent to the application. |
| **f1** | is the EBCDIC code for the character *1.* |
| **08** | means that when the APL2 character set is toggled on, then when this key and the Shift are pressed together the character represented by the code in column 5 (in this case 72) will be sent to the application. |
| **72** | is the EBCDIC code for the *diaeresis* character. |
| **08** | means that when the APL2 character set is toggled on, then when this key and the Alt are pressed together the character represented by the code in column 7 (in this case da) will be sent to the application. |
| **da** | is the EBCDIC code for the *Down Tack Up Tack* character. |

For a full list of the character codes and the associated default keycodes please refer to Appendix B, *TCP/IP Version 2 Release 2.1 for MVS: User's Guide.*

5. To test the mapping we changed the entry for the keycode for the 1 key. The default entry in *userid.*GDXAPLCS.MAP for this keycode is:

```
0a  00  f1  08  72  08  da
```

We altered it to:

```
0a  00  f2  08  72  08  da
```

This means that whenever this key is pressed, the character *2* will be sent to the GDDM application.

**Note:** Remember that this mapping will only happen when the APL2 character set is toggled on. The *userid*.GDXAPLCS.MAP data set has no effect when the APL2 character set is off.

## 3.2 Under VM

IBM provides the C source for three sample programs with the X Window System for VM. These programs execute exactly the same under VM as they do under MVS. Refer to 3.1, "Under MVS" on page 59 for further details. You can use these sample programs to understand how to code function calls for either the Xlib, Xt or widget set libraries.

In 2.2.1, "Installation Verification for the MVS X Window System API" on page 26 we document the steps required to produce executable modules for the sample applications under VM.

Also included with the VM X Window System are the same standard MIT X client applications provided with MVS. These provide more complex examples of how X client applications can be coded to perform certain functions. For a full list of the supplied MIT X client applications please refer to *TCP/IP Version 2 Release 2 for VM: Programmer's Reference.*

### 3.2.1 Compiling and Link-Editing Under VM

When compiling and link-editing your X client application under VM you need to ensure that you access the appropriate libraries to resolve all external references during the link-edit stage.

The libraries that you must point to using a CMS GLOBAL command depend upon which X Window functions your application is using. The options are:

- Only Xlib functions
- Xt intrinsics
- Athena widget set functions
- OSF/Motif widget set functions

We compiled and tested two of the MIT applications under VM:

- BITMAP which uses only Xlib functions.
- OCLOCK which uses the Athena widget set functions.

Note that there is an XCLIENT EXEC on the product tape which can be used to generate all X client application programs.

#### 3.2.1.1 Using Only Xlib Functions Under VM

Use the following steps as a guide to produce an executable module for BITMAP which uses only Xlib functions:

1. You need to have access to the following disks:
   - TCPMAINT 592 minidisk for TCP/IP client code
   - TCPMAINT 5C4 minidisk for the sample programs
   - The minidisk that has the C/370 compiler
2. On the 5C4 minidisk you will find the following source files:

- BITMAP C
- BMDIALOG C

3. Produce an executable module for BITMAP by doing the following:

   a. Set the LOADLIB and TXTLIB search order using the following CMS commands:

      ```
      SET LDRTBLS 25
      GLOBAL LOADLIB EDCLINK
      GLOBAL TXTLIB X11LIB COMMTXT EDCBASE IBMLIB CMSLIB
      ```

   b. Compile BITMAP C using the following command:

      ```
      CC BITMAP (DEFINE(IBMCPP)
      ```

      This creates BITMAP TEXT on your A disk.

   c. Compile BMDIALOG C using the following command:

      ```
      CC BMDIALOG (DEFINE(IBMCPP)
      ```

      This creates BMDIALOG TEXT on your A disk.

      When you compile both BITMAP and BMDIALOG you will see warning messages that indicate that external names have been truncated. This is normal since VM can only support external names of eight characters or less.

   d. Link-edit BITMAP with BMDIALOG using the following command:

      ```
      CMOD BITMAP BMDIALOG
      ```

      This creates BITMAP MODULE on your A disk.

4. Ensure that the X server has authorized the VM host as an X client. For specific details on how to do this please refer to 4.1.6, “Controlling X Client Access to AIX/6000” on page 125 for AIX as the X server, 4.2.4, “Controlling X Client Access to OS/2” on page 132 when OS/2 is the X server, and 4.3.1.4, “Controlling X Client Access” on page 153 for a DOS X server.

5. Identify the target X server display using the following CMS command:

   ```
   GLOBALV SELECT CENV SET DISPLAY <X client display variable>
   ```

   where *<X client display variable>* is the variable that the X client must use to access the X server. It has the format *host:0.0* where *host* is the Internet address or host name of the X server and *0.0* represents the *target server.server screen*. For example, when using our OS/2 machine as the X server, since it had the Internet address 9.67.38.89, we typed the command:

   ```
   GLOBALV SELECT CENV SET DISPLAY 9.67.38.89:0.0
   ```

6. The BITMAP program allows you to create and edit bitmaps that can be used by X client application programs to display cursors, icons and titles. Start BITMAP by typing the following command at the CMS command line:

```
BITMAP -FN 6x10 TEST
```

where TEST is the name of the bitmap to be created. The option *-FN 6x10* specifies the font. We discovered we had to choose a smaller font than the default font for BITMAP. When we used the default font to display the BITMAP window at our OS/2 X server with an 8513 display we got an error message indicating that the window was not big enough. We were able to use the default BITMAP font when using a RISC System/6000 with a 6091 Model 19 screen (1280x1024 pixel resolution).

The window displayed at an OS/2 X server by the BITMAP X client application is illustrated in Figure 46.

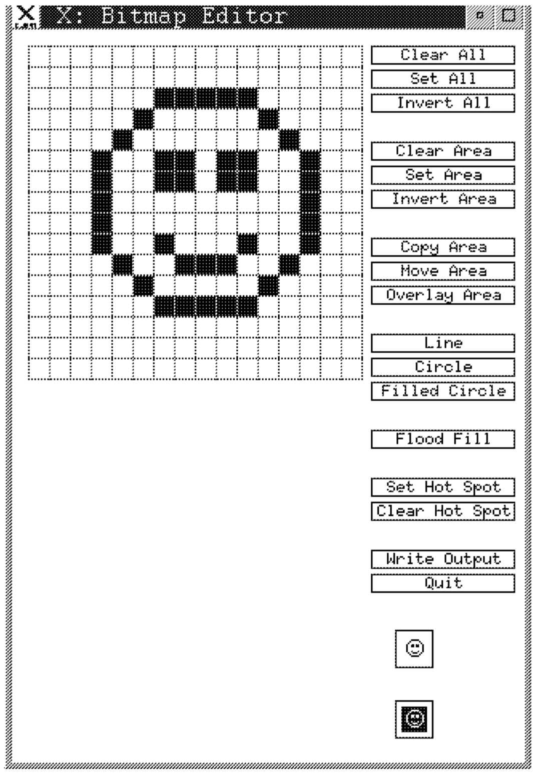


*Figure 46. Display at an OS/2 X Server for MIT BITMAP X Client Program*

You can look at the file BITMAP HLPX11 on the TCPMAINT 5C4 minidisk for detailed instructions on how to use BITMAP.

### 3.2.1.2 Using Athena Widget Set Functions under VM

Use the following steps as a guide to produce an executable module for OCLOCK, which uses the Athena widget set functions:

1. You need to have access to the following disks:
   - TCPMAINT 592 minidisk for TCP/IP client code
   - TCPMAINT 5C4 minidisk for the sample programs
   - The minidisk that has the C/370 Compiler
2. On the 5C4 minidisk you will find the following source files:

- OCLOCK C
- NCLOCK C
- TRANSFOR C

3. Produce an executable module for OCLOCK by doing the following:

   a. Set the LOADLIB and TXTLIB search order using the following CMS commands:

      ```
      SET LDRTBLS 25
      GLOBAL LOADLIB EDCLINK
      GLOBAL TXTLIB XAWLIB XTLIB X11LIB COMMTXT EDCBASE IBMLIB CMSLIB
      ```

      Note that we are now pointing to the Athena widget set library and the X toolkit intrinsics library.

   b. Compile OCLOCK C using the following command:

      ```
      CC OCLOCK (DEFINE(IBMCPP)
      ```

      This creates OCLOCK TEXT on your A disk.

   c. Compile NCLOCK C using the following command:

      ```
      CC NCLOCK (DEFINE(IBMCPP)
      ```

      This creates NCLOCK TEXT on your A disk.

   d. Compile TRANSFOR C using the following command:

      ```
      CC TRANSFOR (DEFINE(IBMCPP)
      ```

      This creates TRANSFOR TEXT on your A disk.

      When you compile OCLOCK, NCLOCK, and TRANSFOR, you will see warning messages that indicate that external names have been truncated. This is normal since VM can only support external names of eight characters or less.

   e. Link-edit OCLOCK with NCLOCK and TRANSFOR using the following command:

      ```
      CMOD OCLOCK NCLOCK TRANSFOR
      ```

      This creates OCLOCK MODULE on your A disk.

4. Ensure that the X server has authorized the VM host as an X client. For specific details on how to do this please refer to 4.1.6, “Controlling X Client Access to AIX/6000” on page 125 for AIX as the X server, 4.2.4, “Controlling X Client Access to OS/2” on page 132 when OS/2 is the X server, and 4.3.1.4, “Controlling X Client Access” on page 153 for a DOS X server.

5. Identify the target X server display using the following CMS command:

   ```
   GLOBALV SELECT CENV SET DISPLAY <X client display variable>
   ```

where *<X client display variable>* is the variable that the X client must use to access the X server. It has the format *host:0.0* where *host* is the Internet address or host name of the X server and *0.0* represents the *target server.server screen*. For example, when using our OS/2 machine as the X server, since it had the Internet address 9.67.38.89, we typed the command:

```
GLOBALV SELECT CENV SET DISPLAY 9.67.38.89:0.0
```

6. Run OCLOCK by typing the following command at the CMS command line.

```
OCLOCK
```

The OCLOCK provided with VM is quite different from the standard MIT OCLOCK provided with AIXwindows Environment/6000. When you run the OCLOCK from VM you will see a window opened at the X server that is identical to that for the program XCLOCK. This window is illustrated in Figure 35 on page 68.

The file OCLOCK HLPX11 does not seem to apply to the OCLOCK program provided with VM. It would, however, apply to the standard MIT OCLOCK program!

You can end the OCLOCK program by closing down the window at the X server.

#### 3.2.1.3 Using OSF/Motif Widget Set Functions under VM

Please refer to 2.3.1, "Installation Verification for the VM X Window System API" on page 31 and the steps used to compile and link-edit XSAMP3 as a guide to producing an executable module for a program that uses OSF/Motif widget set functions.

### 3.2.2 VM Application Resource File

As with MVS, it is possible to use an application resource file for X client applications on VM. It is used to change the characteristics of an application at run time by specifying values for resources that will override the values that are coded within the application. Characteristics include application window size, position on the screen, color, fonts, and other functional details.

Under VM the application resource file is called X DEFAULTS. The following is an example that you can use as a guide to building an application resource file under VM.

#### 3.2.2.1 Building an Application Resource File under VM

In 3.1.2.1, "Building an Application Resource File Under MVS" on page 69 we describe how to build an application resource file under MVS for the X client application XLOGO. That example, in terms of the contents of the application resource file and the effect on the characteristics of XLOGO, would apply equally well to VM.

As discussed in 3.2.1.1, "Using Only Xlib Functions Under VM" on page 80, we compiled and link-edited the MIT client application BITMAP that is provided with the X Window System for VM. In order to provide an example of building an application resource file under VM, we performed the following steps to build an application resource file for BITMAP:

1. Ensure that you are linked to the following disks:
   - TCPMAINT 592 minidisk
   - TCPMAINT 5C4 minidisk
2. When building an application resource file, the first thing you need to do is to understand the application resources that are available for modification. Normally you would refer to the documentation for a program, or look at the program source code.

   If you look in the file BITMAP HELPX11 on the TCPMAINT 5C4 minidisk you will find a list of the application resources that can be modified for BITMAP. These are described in Table 10.

*Table 10. Application Resources for BITMAP*

| **Resource** | **Description** |
|---|---|
| Geometry | Determines the size and location of the BITMAP window. |
| Background | Specifies the window′s background color. The default is white. |
| Foreground | Specifies the window′s foreground color. The default is black. |
| BorderColor | Specifies the color of the window′s border. The default is black. |
| BorderWidth | Specifies the width of the window′s border. The default is 2 pixels. |
| BodyFont | Determines the text font that appears in the BITMAP buttons. The default is *-misc-fixed-medium-r-semicondensed--13-120-75-75-c-60-iso8859-1* or *fixed* for short. |
| Dashed | Specifies whether the BITMAP grid has dashed or solid lines. The default is *on* which means a dashed grid. |
| Highlight | Specifies the color for areas that have been highlighted on the grid for moving, setting, clearing or inverting. |
| Mouse | Specifies the color of the pointer. |

3. Set the following CMS global variables by issuing these three commands:

```
SET LDRTBLS 25
GLOBAL LOADLIB EDCLINK
GLOBAL TXTLIB X11LIB COMMTXT EDCBASE IBMLIB CMSLIB
```

4. Ensure that the X server has authorized the VM host as an X client. For specific details on how to do this please refer to 4.1.6, “Controlling X Client Access to AIX/6000” on page 125 for AIX as the X server, 4.2.4, “Controlling X Client Access to OS/2” on page 132 when OS/2 is the X server, and 4.3.1.4, “Controlling X Client Access” on page 153 for a DOS X server.
5. Identify the target X server display using the following CMS command:

```
GLOBALV SELECT CENV SET DISPLAY <X client display variable>
```

   where *<X client display variable>* is the variable that the X client must use to access the X server. It has the format *host:0.0* where *host* is the Internet address or host name of the X server and *0.0* represents the *target server.server screen*. For example, when using our OS/2 machine as the X server, since it had the Internet address 9.67.38.89, we typed the command:

```
GLOBALV SELECT CENV SET DISPLAY 9.67.38.89:0.0
```

6. Run BITMAP by typing the following command at the CMS command prompt:

   ```
   BITMAP TEST
   ```

   The window that you will see opened at the X server is illustrated in Figure 46 on page 82. Notice the default window dimensions and colors.

7. Close down the BITMAP window at the X server to end the program.

8. A sample X DEFAULTS file is provided on the TCPMAINT 592 minidisk. You can either copy this file onto your A disk or edit it directly. Add some entries to change the resources for BITMAP. An example of some application resource file entries for BITMAP is illustrated in Figure 47.

```
bitmap*Foreground: red
bitmap*Background: blue
bitmap*Highlight: green
bitmap*BodyFont: 6x10
bitmap*Geometry: 600x400
bitmap*Dimensions 10x10
bitmap*BorderColor: black
bitmap*BorderWidth: 5
bitmap*Dashed: off
bitmap*Mouse: black
```

*Figure 47. Example of the Contents of the Application Resource File for BITMAP*

   **Note:** The application resource file is case sensitive. You must type the resource names exactly as they are specified for the application.

9. Run BITMAP again.

10. Notice the modified characteristics of the BITMAP window. The application resource file shown in Figure 47 produced a BITMAP window at our OS/2 X server as illustrated in Figure 48 on page 87. Compare this to the BITMAP window illustrated in Figure 46 on page 82 which, except for the font, used application resource defaults.

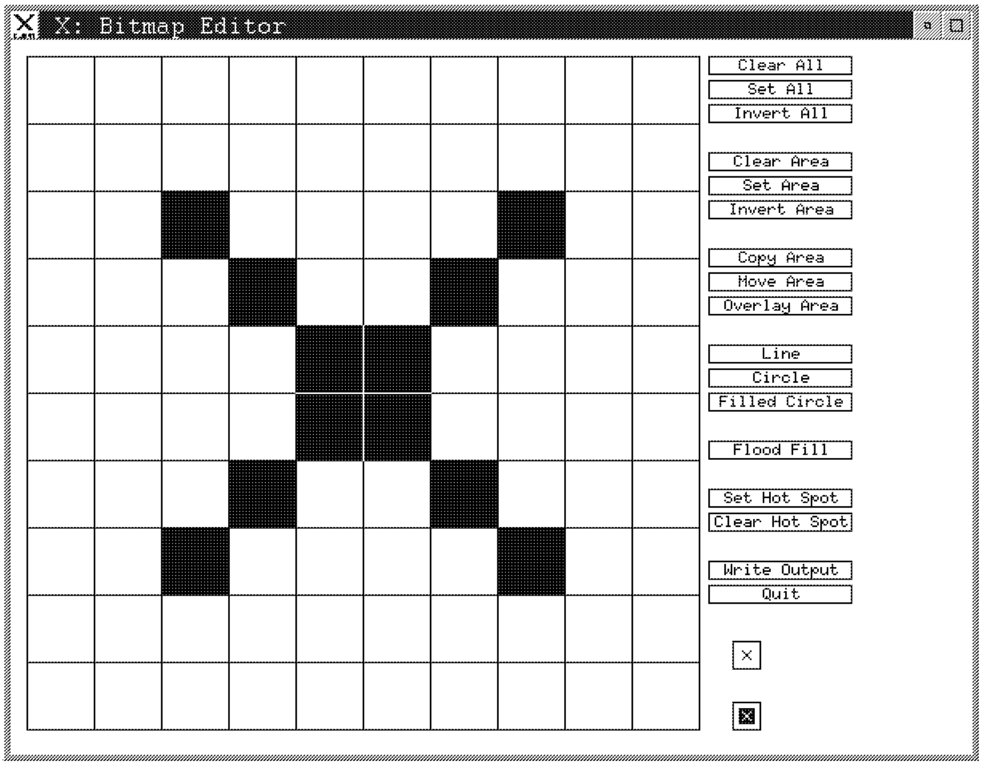


*Figure 48. Display at an OS/2 X Server for Modified MIT BITMAP X Client Program*

**Note**

When using an OS/2 X server to display the BITMAP window with the application resource file as specified in Figure 47 on page 86 we got the following message at VM:

```
bitmap: unable to allocate color cells
```

The BITMAP window at the OS/2 X server still displayed the default colors black and white as opposed to the colors set in the application resource file. We did not get this problem when using the AIX X server or the HCL-eXceed X server. This problem occurred because PMX was running in StaticColor only mode and other X servers were in PseudoColor mode. The solution is to run BITMAP with PMX running in PseudoColor mode.

### 3.2.3 Using GDDM Applications under VM

The GDDM shared segment needs to be reinstalled when installing the X Window System GDDM interface to allow the interface modules to access the shared segment. This is described in 2.3.2, “Installing the VM X Window System GDDM Interface” on page 33.

When you install the X Window System GDDM interface you have two choices for reinstalling the GDDM shared segment:

1. Reinstall the existing named GDDM shared segment. If this is done, then you may need to reassemble the bootstrap modules for each of the products that use GDDM.

2. Make a copy of the existing named GDDM shared segment with a different name for use by the X Window System GDDM interface modules. If you do this, then you must zap the load modules for those products that will be used with the new GDDM shared segment for the X Window System GDDM interface. This must be done to point the module to the name of the new GDDM shared segment that must be accessed in order to use the X Window System GDDM interface.

On our VM system we opted to make a copy of the existing named GDDM shared segment with a different name for use by the X Window System GDDM interface modules.

An example of a GDDM application that is commonly used under VM is ADMCHART. We tested ADMCHART with the X Window System GDDM interface for VM and, in order to do this, we had to zap the ADMCHART load module to have it point to the new named GDDM shared segment. You can use the following steps as a guide to using ADMCHART with the VM X Window System GDDM interface:

1. Ensure you have access to the TCPMAINT 592 minidisk, the GDDM minidisk and the minidisk that holds the C/370 libraries.
2. Copy the file ADMCHART MODULE to your A disk.

   On our VM system we have a production GDDM shared segment called ADMXA230. When we installed the X Window System GDDM interface we reinstalled a copy of this GDDM shared segment and called it GDDMXD.

   The ADMCHART MODULE points to ADMXA230. In order to use ADMCHART with the X Window System GDDM interface we ″zapped″ the ADMCHART MODULE to point to GDDMXD.
3. You can zap ADMCHART using the ZAP command under VM. This command can either accept input from the CMS command line or from an input file called *filename* ZAP, where *filename* can be any name you choose. We recommend using an input file because it minimizes the chance of a typing error.

   Create a ZAP input file called ADMCHART ZAP on your A disk. The contents of this file are illustrated in Figure 49. In this file you must name the module you are going to zap (in this case ADMCHART), verify the contents of the bytes you are going to replace, and specify the new byte values that will replace the bytes that were verified. The starting location of the bytes to be replaced is specified by an offset from the beginning of the module.

   In the ZAP input file in Figure 49 we verify that the contents of the 8 bytes at offset 0CB0 is C1C4D4E7C1F2F3F0 which is the byte representation of ADMXA230. We then replace these bytes, again at offset 0CB0 with C7C4C4D4E7C44040, which is the byte representation of GDDMXD.

```
NAME ADMCHART
VER 0CB0 C1C4D4E7C1F2F3F0
REP 0CB0 C7C4C4D4E7C44040
DUMP ADMCHART ALL
```

*Figure 49. Contents of the ZAP Input File ADMCHART ZAP*

4. Execute ZAP by typing the following command at the CMS command line:

```
ZAP MODULE (INPUT ADMCHART
```

5. For VM/XA systems only, enter the following command from the CMS command prompt:

   ```
   SET STORECLR ENDCMD
   ```

   This ensures that GETMAIN requests by the X Window System GDDM interface code are processed correctly.

6. Activate the X Window System GDDM interface by issuing the following command:

   ```
   GDDMXD ON
   ```

   You should see the message `GDDMXD/VM active`.

7. Set the following CMS global variables by issuing these three commands:

   ```
   SET LDRTBLS 25
   GLOBAL LOADLIB EDCLINK
   GLOBAL TXTLIB ADMNLIB GDDMXD ADMPLIB ADMGLIB X11LIB COMMTXT IBMLIB
   EDCBASE CMSLIB
   ```

8. Identify the target X server display using the following CMS command:

   ```
   GLOBALV SELECT CENV SET DISPLAY <X client display variable>
   ```

   where the *<X client display variable>* is the variable that the X client must use to access the X server. It has the format *host:0.0* where *host* is the Internet address or host name of the X server and *0.0* represents the *target server.server screen*. For example, when using our OS/2 machine as the X server, since it had the Internet address 9.67.38.89, we typed the command:

   ```
   GLOBALV SELECT CENV SET DISPLAY 9.67.38.89:0.0
   ```

9. Ensure that the X server has authorized the VM host as an X client. For specific details on how to do this please refer to 4.1.6, "Controlling X Client Access to AIX/6000" on page 125 for AIX as the X server, 4.2.4, "Controlling X Client Access to OS/2" on page 132 when OS/2 is the X server, and 4.3.1.4, "Controlling X Client Access" on page 153 for a DOS X server.

10. In order to interact with ADMCHART using the X server keyboard you will need to alter the keyboard mapping for the Enter key. We discovered that the default keyboard mappings provided with the X Window Systems for both OS/2 and AIX had the Enter key set to the keysym name *RETURN*. You need to remap this key's keysym name to *EXECUTE*. Please refer to 4.2.2.1, "Remapping the Keyboard under OS/2" on page 128 for a description of how to perform the remapping with OS/2 and 4.1.5, "Remapping the Keyboard Under AIX/6000" on page 123 for AIX.

11. Invoke ADMCHART from the CMS command line.

An example of the ADMCHART home panel for ADMCHART is illustrated in Figure 38 on page 71.

You will notice that the ADMCHART panel illustrated in Figure 38 on page 71 is partly obscured or clipped. This window has been opened at an OS/2 X server that had an 8513 screen. The maximum window size possible on this screen is not big enough to accommodate the default GDDM graphics display area specified by the VM X Window System GDDM interface. Please refer to 3.2.3.1, "Application Resource File for GDDM under VM" for more information.

#### 3.2.3.1 Application Resource File for GDDM under VM

In 3.2.2.1, "Building an Application Resource File under VM" on page 84 we describe how to build an application resource file under VM. As with user-written applications, it is also possible to modify the runtime characteristics for GDDM applications by specifying resource values in the application resource file X DEFAULTS.

In Chapter 12 of the *TCP/IP Version 2 Release 2 for VM: User's Guide* you will find a description of the application resources for which you can define values to specify the characteristics for GDDM applications that use the VM X Window System GDDM interface. There are more resources available under VM as opposed to MVS. The VM GDDM resources are described in Table 11.

*Table 11 (Page 1 of 2). Application Resources for GDDM under VM*

| Resource | Description |
|---|---|
| Geometry | Specifies the size and location of the initial GDDM window at the X server. The size of the GDDM graphics display area is dependent upon the size of the window. Unlike MVS, VM provides four possible GDDM graphics display area sizes which are determined by the width of the X server window. They are shown in Table 12 on page 91. The default GDDM graphics display area is 720 pixels wide by 512 pixels high, while the minimum is 480 pixels wide by 352 pixels high. In order to achieve the minimum size, VM implements a contraction algorithm as described below for the *Compr* resource option. If the window is specified as less than this, then the GDDM display area will be clipped. |
| GColornn | Specifies the GDDM color that is mapped onto an X Window System color. |
| GMCPnn | Specifies the GDDM multicolor pattern that is mapped onto an X Window System color. |
| ZWL | Specifies that the server use the fastest drawing algorithm to draw lines. The default is *N* because if this option is used it may result in lines that are not the width they were intended to be by the application. |
| XSync | Specifies that the X Window System GDDM interface send one request at a time to the X server (operate in a synchronous mode) when set to *Y*. The default is that the interface communicate with the X server asynchronously which is the more usual for the X protocol. |
| CMap | Specifies whether to load the default color map or not. This is specified as *N* if the color map at the X server is going to be used, as opposed to the default color map that otherwise would be loaded by the MVS X Window System GDDM interface. |

*Table 11 (Page 2 of 2). Application Resources for GDDM under VM*

| Resource | Description |
|---|---|
| HostRast | Specifies whether the raster processing is done at the host or the workstation. The default for this option is *N*, which means that the raster processing is done at the workstation. You should choose *Y* when the GDDM application involves multiplane symbol sets or GDDM color mixing. |
| XClConn | Specifies whether the X Window System GDDM interface should close the window at the X server when the GDDM application finishes. The default is *Y*, which means the window will close when the GDDM application finishes. *N* means that the window will be left open and will continue to display the last GDDM graphics window displayed at the workstation even though the application has closed down. The only way to close this window is at the X server itself. |
| Compr | Controls the technique used to compress bitmapped data when the X server window size has been defined as 480 pixels by 352 pixels. The default value is *O* for OR. The other choice is *A* for AND. |
| PDTrace | Specifies whether the problem determination trace should be *Y* (for ON), or *N* (for OFF, the default) which means X Window System GDDM interface trace data will not be generated. |

*Table 12. GDDM Window Width and Graphics Display Area Relationship*

| Window Width (pixels) | GDDM Graphics Display Area (pixels) |
|---|---|
| < 650 | 450 wide by 350 high |
| 650 to 849 | 720 wide by 512 high |
| 850 to 1024 | 960 wide by 682 high |
| >1024 | 1200 wide by 864 high |

You specify these resource options in the application resource file X DEFAULTS. You simply edit this file and enter the X Window System GDDM interface resource options after existing entries that you may have for other X client applications.

A sample application resource file for GDDM applications is provided with the X Window System for VM. It is called X DEFAULTS and you will find it on the TCPMAINT 592 minidisk.

In 3.1.3.1, “Application Resource File for GDDM under MVS” on page 72 you will find an example of the application resource file for the MVS X Window System GDDM interface. The following example of an application resource file for VM is included to demonstrate some of the additional resources available with the VM X Window System GDDM interface.

An example of application resource file entries for a VM GDDM application is shown in Figure 50 on page 92.

**Note:** The application resource file is case sensitive and so the variables must be coded exactly as shown.

```
gddmx*Geometry: 500x400+10+10
gddmx*GColor1: Purple
gddmx*GColor2: Orange
gddmx*GColor4: White
gddmx*ZWL: N
gddmx*XSync: N
gddmx*CMap: Y
gddmx*HostRast: N
gddmx*XCIConn: Y
gddmx*Compr: A
gddmx*PDTrace: N
```

*Figure 50. Example Application Resource File Entries for GDDM Applications*

The application resource file entries shown in Figure 50 specify the following:

**gddmx*Geometry: 500x500+10+10** specifies a window at the X server that will be 500 pixels wide by 500 pixels high and will be 10 pixels in the X direction and 10 pixels in the Y direction from the top left-hand corner of the display. This will mean that the GDDM graphics display area will be 480 pixels by 352 pixels.

**gddmx*GColor1: Purple** specifies that the X Window color purple should be displayed for the GDDM color blue.

**gddmx*GColor2: Orange** specifies that the X Window color orange should be displayed for the GDDM color red.

**gddmx*GColor4: White** specifies that the X Window color white should be displayed for the GDDM color green.

**gddmx*ZWL: N** specifies that line widths will be displayed at normal width.

**gddmx*XSync: N** specifies that the communication to the X server is asynchronous.

**gddmx*CMap: Y** directs the X Window System GDDM interface to load the default color map.

**gddmx*HostRast: N** directs that raster processing be done at the workstation.

**x*XCIConn: Y** specifies that the connection to the X server be closed when the application is closed.

**x*Compr: A** specifies that the compression algorithm used to achieve a 480 pixel by 352 pixel graphics display should use the logical AND function.

**gddmx*PDTrace: N** specifies that no X Window System GDDM interface trace data should be generated.

We set these options in an X DEFAULTS file on our A disk and started up the VM X Window System GDDM interface as described in 3.2.3, "Using GDDM Applications under VM" on page 87. We identified an OS/2 X server as the target display and started ADMCHART from the CMS command line. A window as illustrated in Figure 51 on page 93 was opened at the OS/2 X server.

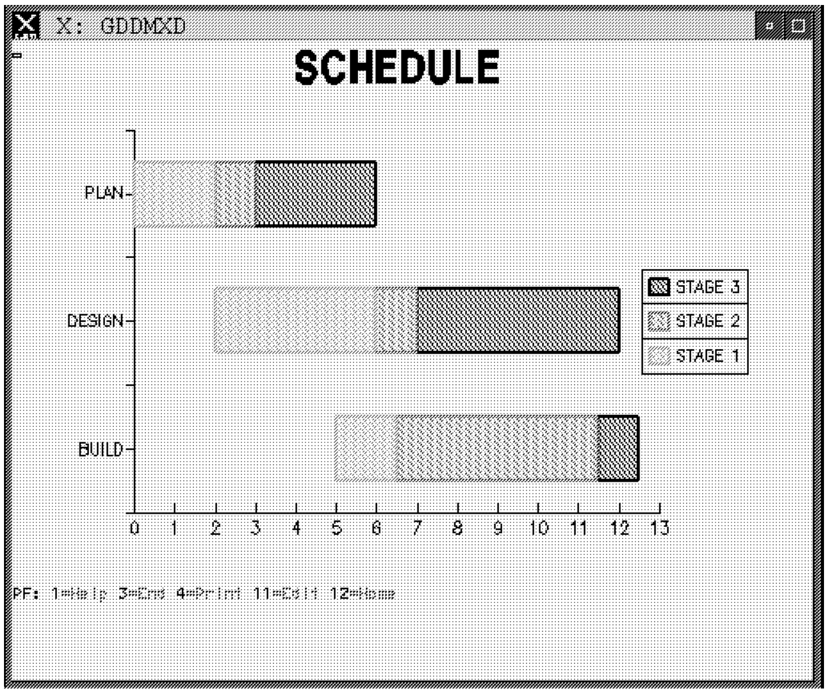


*Figure 51. Display at an OS/2 X Server for ADMCHART under VM/CMS*

Notice that the GDDM display area in Figure 51 is not clipped at all. This is because we set the X window width as 500 pixels, which means that the GDDM display area will be set at 450 pixels wide by 350 pixels high. This can be displayed on a PS/2 8513 screen.

### 3.2.3.2 APL2 Character Set Keyboard for GDDM under VM

The support for the APL2 character set under VM is exactly the same as that provided with the MVS X Window System GDDM interface. The following description differs only in terms of the file names and operating system terminology.

Unfortunately GDDM applications that require APL2 characters to be input from a keyboard are not classic X Window System applications because they are not truly device independent. These applications are designed to operate with an IBM 3179-G display and an APL2 keyboard where physical keys correspond to specific character representations.

The X Window System GDDM interface for VM provides a mechanism which allows the keyboard at an X server to be used as an APL2 keyboard. A map is provided which allows the X Window System GDDM interface to accept keystrokes from the X server and translate them into valid APL2 data for the X client application.

When an application is started it initializes the X Window System GDDM interface which looks for a file called GDXALTCS PSS. This file provides the default mapping for the primary and alternate character sets associated with an IBM 3179-G display. You will find this file on the TCPMAINT 592 minidisk.

In an X Window environment, each physical key is associated with a keycode. The VM X Window System GDDM interface uses GDXALTCS PSS to map keycodes into characters that are then sent to the GDDM application.

Characters are represented by keycodes in one of the following ways:

1. A single keycode only.
2. A keycode with a modifier keycode.

The two modifier keys that are used are the Shift and the Alt keys.

The APL2 character set mode is toggled on by pressing the Backspace key together with the Alt key. This is indicated by the characters (APL) in the title bar of the X window at the X server as shown in Figure 43 on page 77. This means that you are able to use the APL2 character set at your X server keyboard.

GDXALTCS PSS only comes into play when the APL2 character set is toggled on.

The default mapping in GDXALTCS PSS assumes a keyboard at the X server that is equivalent to the IBM 101-key Enhanced Keyboard. What this actually means is that it is assumed that the physical key represented by a particular keycode has the appropriate primary and alternate characters on that key as you would find on an IBM 3179-G 101-key Enhanced Keyboard. Where this is not the case it is possible to override the default mapping.

If the keyboard at your X server is not set out such that characters correspond to particular keycodes as expected by the VM X Window System GDDM interface, then you can use the following steps as a guide to providing a customized mapping for your keyboard:

1. The first thing you need to know is what is the keycode associated with each physical key on your keyboard. Provided with the X Window System GDDM interface is an X client program called KEYCODE. You will find this on the TCPMAINT 592 minidisk. When invoked this program opens a window at your server which will allow you to determine the keycode for each key that is pressed. Invoke KEYCODE as follows:

   a. Ensure that the X server has authorized the VM host as an X client.

   b. Set the LOADLIB and TXTLIB search order using the following CMS commands:

   ```
   SET LDRTBLS 25
   GLOBAL LOADLIB EDCLINK
   GLOBAL TXTLIB X11LIB COMMTXT EDCBASE IBMLIB CMSLIB
   ```

   c. Identify the target X server display using the following CMS command:

   ```
   GLOBALV SELECT CENV SET DISPLAY <X client display variable>
   ```

   where *<X client display variable>* is the variable that the X client must use to access the X server. It has the format *host:0.0* where *host* is the Internet address or host name of the X server and *0.0* represents the *target server.server screen*. For example, when using our OS/2 machine as the X server, since it had the Internet address 9.67.38.89, we typed the command:

   ```
   GLOBALV SELECT CENV SET DISPLAY 9.67.38.89:0.0
   ```

d. From the CMS command line type:

```
KEYCODE
```

e. KEYCODE will open a window at your X server. Press any key and notice the corresponding keycode. Figure 44 on page 78 shows the KEYCODE window after the Alt and Backspace keys have been pressed.

   The display in Figure 44 on page 78 shows that the keycode for the Backspace key is 17 and the Alt key is the modifier.

2. IBM provides a sample APL2 character set map in GDXAPLCS SAMPMAP on the TCPMAINT 592 minidisk. Copy this to GDXAPLCS MAP on your A disk. When the VM X Window System GDDM interface is initialized it looks for GDXAPLCS MAP, which it will use to override the default APL2 character set mappings.

3. Based on the keycodes you obtained from the KEYCODE application you can edit GDXAPLCS MAP and change the character codes associated with each keycode to match your keyboard. A section of the contents of GDXAPLCS MAP is illustrated in Figure 52.

```
0a  00  f1  08  72  08  da
0b  00  f2  08  a0  08  fb
0c  00  f3  00  4c  08  dc
0d  00  f4  08  8c  08  dd
0e  00  f5  00  7e  08  cd
0f  00  f6  08  ae  08  cf
10  00  f7  00  6e  08  ed
11  00  f8  08  be  08  fd
12  00  f9  08  78  08  cb
13  00  f0  08  71  08  ca
14  00  4e  00  60  08  db
15  08  b6  08  b8  08  ee
19  00  d8  00  6f  08  58
1a  00  e6  08  b4  08  66
1b  00  c5  08  b1  08  45
1c  00  d9  08  b3  08  59
1d  00  e3  08  80  08  63
```

*Figure 52. Example of the Contents of GDXAPLCS MAP*

The column values in the map in Figure 52 have the following meaning:

**Column 1** is the hexadecimal keycode for the physical key.

**Column 2** defines whether the character is in the primary or alternate character set when the key corresponding to the keycode is pressed alone. 0 means the character is in the primary set while 8 means the alternate character set.

**Column 3** is the EBCDIC code for the character in the character set when the key corresponding to the keycode is pressed alone.

**Column 4** defines whether the character is in the primary or alternate character set when the key corresponding to the keycode and the Shift key are pressed together. 0 means the character is in the primary set while 8 means the alternate character set.

| | |
|---|---|
| **Column 5** | is the EBCDIC code for the character in the character set when the key corresponding to the keycode and the Shift key are pressed together. |
| **Column 6** | defines whether the character is in the primary or alternate character set when the key corresponding to the keycode and the Alt key are pressed together. 0 means the character is in the primary set while 8 means the alternate character set. |
| **Column 7** | is the EBCDIC code for the character in the character set when the key corresponding to the keycode and the Alt key are pressed together. |

The primary character set is available when the APL2 character set is toggled either on or off. The alternate character set is only available when the APL2 character set is toggled on. Consider the first row in the table as an example:

| | |
|---|---|
| **0a** | is the keycode for what is normally the 1 key. |
| **00** | means that when this key is pressed alone the character represented by the code in column 3 (in this case f1) will be sent to the application. |
| **f1** | is the EDCDIC code for the character *1.* |
| **08** | means that when the APL2 character set is toggled on, then when this key and the Shift are pressed together the character represented by the code in column 5 (in this case 72) will be sent to the application. |
| **72** | is the EBCDIC code for the *diaeresis* character. |
| **08** | means that when the APL2 character set is toggled on, then when this key and the Alt are pressed together the character represented by the code in column 7 (in this case da) will be sent to the application. |
| **da** | is the EBCDIC code for the *Down Tack Up Tack* character. |

For a full list of the character codes and the associated default keycodes please refer to Appendix C, *TCP/IP Version 2 Release 2 for VM: User's Guide.*

4. To test the mapping we changed the entry for the keycode for the 1 key. The default entry in GDXAPLCS MAP for this keycode is:

```
0a  00  f1  08  72  08  da
```

We altered it to:

```
0a  00  f2  08  72  08  da
```

This means that whenever this key is pressed, the character *2* will be sent to the GDDM application.

**Note:** Remember that this mapping will only happen when the APL2 character set is toggled on. GDXAPLCS MAP has no effect when the APL2 character set is off.

## 3.3 Under AIX/6000

There are many clients delivered with AIXwindows Environment/6000. Some are already compiled, the others are in source code. The examples that can be compiled reside in the /usr/lpp/X11/Xamples directory. The files in this directory are not mandatory for AIXwindows. You can delete the whole directory, but it is useful to have some of these clients available as executable programs.

### 3.3.1 Compiling and Linking under AIX/6000

You can build all of the clients in the /usr/lpp/Xamples directory in one step, or you can build the clients selectively. Building only the clients you need is useful when you are low on disk space. For building all the clients you will need at least 20MB of free disk space in your /usr file system. This requirement is valid for AIX V3.2 but more space is required if using more recent levels of the AIX operating system. The compilation of all examples took over 4 hours on our IBM RISC System/6000 Mod. 530. First let′s see how to create all the samples. You can find information in the /usr/lpp/X11/Xamples/README file.

#### 3.3.1.1 Creating the Sample Clients

Use the following steps as a guide to creating all sample X clients:

***Installing the Imake Configuration:*** *Imake* is a project management tool that is used for building the X Window System from source code. In particular, it is the tool of choice for public domain X programs that are distributed in source form. Imake uses a combination of *make* and the C preprocessor (*cpp*). Cpp does not do anything except process text files: it cannot invoke other programs. Although *make* can actually do work, it lacks the ability to test the value of a variable, making it inflexible. What imake does is combine the best features of cpp and make. The real value of imake is that it allows easy creation of Makefiles under changing conditions.

To build and install the imake configuration (including the xmkmf and makedepend commands) without building the entire sample tree located in /usr/lpp/X11/Xamples, do the following:

For imake:

1. Enter `cd /usr/lpp/X11/Xamples/config`.
2. Enter `make -f Makefile.ini` to build the imake command.
3. Enter `/imake -DTOPDIR=/usr/lpp/X11/Xamples` to create the Makefile for the config directory.
4. Enter `make install` to install the config directory (this directory contains the template files that imake requires) and the ′imake′ command.

For xmkmf and makedepend:

1. Enter `cd /usr/lpp/X11/Xamples/util`.
2. Enter `imake -I/usr/lib/X11/config -DTOPDIR=/usr/lpp/X11/Xamples` to make the Makefile for the util directory.
3. Enter `make Makefiles` to make all Makefiles in subdirectories.
4. Enter `make install` to build and install the xmkmf and makedepend commands.

***Creating the Sample Clients:*** To create all of the executables in the Xamples directory tree, follow these steps:

1. Make sure the date is set correctly.
2. Enter `cd /usr/lpp/X11/Xamples`.
3. Enter `make World`.
4. Optional - Enter `make install`. This will put all samples into /usr/lpp/X11/Xamples/bin; add this directory into your search path.
5. Optional - Enter `make install.man`. This will put all sample man-pages into /usr/lpp/X11/Xamples/man; add this directory to the MANPATH environment variable.
6. If a problem occurs while in the `make World` process, in the /usr/lpp/X11/Xamples directory copy the file Makefile.bak to Makefile and restart the command.

**Note:** If you don′t want to compile all of the clients the easiest way is by deleting the /usr/lpp/X11/Xamples/<clients> directories that you don′t want and then starting with step 1 as described above.

**Note:** IBM does not support the clients delivered in source code.

The Makefiles needed are created dynamically using the seed Makefile found in the Xamples directory. The dependencies are then added to these Makefiles. Finally, each subdirectory is visited, making the executables. This may take several hours to finish.

The native C compiler must be installed before the executables can be created.

For convenience, all unsupported sample binaries are supplied in /usr/lpp/X11/Xamples/bin and symbolically linked back to /usr/lpp/X11/bin.

***Linking the Sample Extensions into the Server:*** The shell script Xamples/server/makeServer can be used to link the sample server extensions into the server. Before running this script, the samples must be created (as described above).

X.new is the new server. Rename it to X and place into /usr/lpp/X11/bin to use.

The sample server is required for execution of programs in Xamples/extension/test.

***Building X Client Samples Selectively:*** To install a client selectively perform the following steps:

1. Log in as root and type the following commands at the command prompt:

   ```
   cd /usr/lpp/X11/Xamples
   touch Imakefile
   make Makefile
   make Makefiles
   make linklibs
   ```

2. Use the `cd` command to point to the directory of the sample you wish to build.
3. In this directory type:

```
make
```

The executable code of that client is stored in the current directory. Copy or move the executable in a directory where your PATH variable points to.

### 3.3.2 Customizing Application Resources under AIX/6000

To ensure the device independence of an application the X clients can be set up to make use of resources. A resource is a variable which can control a feature of an X application. Its value can be Boolean, numeric, or string. We can set and change the value of these resources in different ways. Resources are used to define the size of the window, for setting the background and foreground colors and much more. When customizing AIXwindows we normally use the $HOME/.Xdefaults file or the xrdb (X resource database manager) command.

When an application is started it reads the resources set by the user or by the server. Each resource is an entry in a file. The following list shows the locations where the resource information is searched in ascending order of priority (the last item has the final say).

1. /usr/lib/X11/${LANG}/app-defaults/*<class>*
2. /usr/lib/X11/app-defaults/*<class>*
3. ${XAPPLRESLANGPATH}*<class>*
4. ${XAPPLRESDIR}*<class>*
5. RESOURCE_MANAGER property of the root window
6. $XENVIRONMENT
7. /usr/lpp/X11/lib/X11/app-defaults/*application*
8. -xrm command line arguments
9. Your widgets argument list

where:

| | |
|---|---|
| *<class>* | is the name supplied as the second parameter to *XtInitialize*. |
| **RESOURCE_MANAGER** | means the file that was accessed when *xrdb* was started. When *xrdb* was not started the `$HOME/.Xdefaults` file is accessed instead. If this environment variable is not set then look at the `$HOME/.Xdefaults-host`, where *host* is the name of the machine that the client is running on. One or more `-xrm` options can be specified on the command line, when a client is started. The widgets arguments are specified in the source code of your client application. |

The xrdb command stores resources directly in the server, making them available to all clients, regardless of the machine the clients are running on. The xrdb command is used to get or set the contents of the RESOURCE_MANAGER property on the root window of screen 0. In this way the resources are available for all clients which run on the specified server. It allows dynamic changing of defaults without editing files.

The xrdb client won′t be invoked when you start AIXwindows. To start xrdb each time you start your AIX server add the entry `xrdb resourcefile` in your $HOME/.xinitrc file.

Placing resources in files allows you to set many resources at once, without the restriction encountered, when using command line options. You can set different resources for every user who uses the same X application. As most of the common clients are written to use the X toolkit they can use the resources. A resource file normally has this format:

```
! object.subobject[.subobject...].attribute: value
```

where:

| | |
|---|---|
| *!* | comments the resource specification. |
| *object* | is the client program or a specific instance of the program. |
| *subobjects* | correspond to levels of the widget hierarchy (usually the major structures within an application, such as windows, menus, scrollbars, etc.). |
| *attribute* | is a feature of the last subject (perhaps a command button), such as background color or a label that appears on it. |
| *value* | is the actual setting of the resource attribute (Boolean, numeric, or string). |

Resource components can be linked in two ways:

- By a tight binding, represented by a dot (.).
- By a loose binding, represented by an asterisk (*).

A tight binding means that the components on either side of the dot must be next to one another in the widget hierarchy. A loose binding is signaled by an asterisk, a wild card character which means there can be any number of levels in the hierarchy between the two surrounding components.

There is also a difference between instances and classes. Each component of a resource has an associated class. For example, in the case of xterm, the color of text (foreground), the pointer color, and the text cursor are all defined as instances of the class Foreground. This makes it possible to set the value of all three with a single resource specification. Initial capitalization is used to distinguish class names from instance names. The executable file will be created in the directory. Copy the executable file in the directory where you have set your PATH environment variable.

With the *appres* client you can get a list of resources that every X client might access. The resource information can reside in different files which will be accessed at the start. However, when a resource is commented out it won′t be printed. To obtain a complete list of resources for a client, you have to refer to the documentation delivered with the application.

Let′s follow the way a client handles the resource information (for example the background color). A user starts a client program. The client finds the entry for the background color in the app_default file. It sets the background color to that value. Then it searches the user′s home/.Xdefaults file for the entry background color. If the entry is found, the actual setting of the background color changes to the new value. The client passes the corresponding *colorname* to the server which has to display the client′s window. When the server finds a background color entry in its resource database (loaded with xrdb) this value will be taken. However, when the user sets the background color with an option, this will be the final colorname. With the colorname the server selects the color values in the *colormap* table. This color will be shown on the display.

## 3.3.3 How to Start an AIX/6000 Client

We will show you which steps are necessary in order to have your local X client display output on a remote X server. There is no difference between a local and a remote server.

TCP/IP allows us to execute a command on a remote system in several ways. In our case we assume that we log in to an IBM RISC System/6000. It doesn′t matter if your terminal is a graphics terminal, a local terminal, or if you have access through a terminal emulator.

### 3.3.3.1 xterm and aixterm

If you want to display a program which is not programmed as an X client on a server, AIXwindows Environment/6000 provides you with two X clients which provide terminal emulators that you can use to start AIX programs such as *vi*, *smitty*, and 3270 emulation. These clients are *aixterm* and *xterm*. IBM has introduced support for xterm in AIX V3.2.5.

The xterm client requires BSD symbolic links to pseudo-devices. The AIX limitation for these links is 64, which means it is not possible to open more than 64 xterms. xterm supports either the VT102 emulation or the Tektronix 4014 emulation. Xterm is documented in InfoExplorer. Also, there are ″man″ pages for xterm, or you can type `xterm -help` that gives you a list and description of the options.

The IBM version of xterm is *aixterm*.

In particular, aixterm has no limitation on the number of started emulations. It also supports the hft (high function terminal) and VTxxx terminal emulations. As aixterm is an official client in AIX/6000, use InfoExplorer to get a description. We recommend the use of aixterm when you start specific AIX commands (such as smitty) or when the server is AIX-based (uses the same font base). Otherwise you can use xterm.

The following is a list of the resources for the aixterm client.

```
! aixterm.autoRaise:        false
! aixterm.autoRaiseDelay:   2
aixterm.background:         cyan
! aixterm.boldFont:         Bld14.500
! aixterm.borderColor:      black
aixterm.borderWidth:        0
! aixterm.c132:             false
! aixterm.curses:           false
! aixterm.cursorColor:      black
! aixterm.deiconifyWarp:    false
! aixterm.font:             Rom14.500
! aixterm.foreground:       black
! aixterm.geometry:         80x25+0+0
! aixterm.iconBitmap:
! aixterm.iconGeometry:
! aixterm.iconStartup:      false
! aixterm.internalBorder:   2
! aixterm.jumpScroll:       false
! aixterm.logFile:          XtermLog.XXXX
! aixterm.logging:          false
! aixterm.logInhibit:       false
! aixterm.marginBell:       false
! aixterm.nMarginBell:      10
! aixterm.pageOverlap:      1
! aixterm.pageScroll:       false
! aixterm.pointerColor:     black
! aixterm.pointerShape:     XC_left_ptr
! aixterm.reverseVideo:     false
! aixterm.reverseWrap:      false
! aixterm.saveLines:        64
! aixterm.scrollBar:        false
! aixterm.scrollInput:      true
! aixterm.scrollKey:        false
! aixterm.statusLine:       false
! aixterm.statusNormal:     false
! aixterm.textUnderIcon:    true
! aixterm.title:            aixterm
! aixterm.visualBell:       false
! aixterm.vt102:            false
! aixterm.warp:             false
! aixterm.suppress:         false
! aixterm.language:
```

*Figure 53. Available Resources for aixterm*

You can change some parameters dynamically. Move the mouse to the appropriate aixterm window and press the Ctrl key and one of the mouse buttons. Using the menus that are displayed, you can set resources such as the scroll bar.

### 3.3.3.2 Starting a Client

To start a client on a X server perform the following steps:

- Make sure that you have a TCP/IP connection to your server system. For testing you can use the *ping* command.
- The X server must be started on the remote system. Unfortunately there is no standard X client command that can check this.

- Your host must be authorized to use the X server display. To set or reset the host authorization use the *xhost* command at the X server.
- Start your X client with the option:

  ```
  client -display hostname:0:
  ```

  where *hostname* is the name of the server you want to display on. 0 is the number of the display.

If you cannot display the client on the server, you may get one of the following error messages:

```
Xlib:  connection to ″rs60003:0″ refused by server
Xlib:  Client is not authorized to connect to Server
1356-300 xclock: Cannot make a connection to X server rs60003:0.
        If the X server is not running, run the xinit command.
        If the X server is running, check the specified display number.
```

*Figure 54. Error Message: Not Authorized to Connect to Server*

This indicates that your host has not authorized the client to display a window on the server. Set the authorization on the server with the xhost command.

```
1356-300 xclock: Cannot make a connection to X server rs60003:0.
        If the X server is not running, run the xinit command.
        If the X server is running, check the specified display number.
```

*Figure 55. Error Message: X Server is Not Running*

This means that the server is has not been started and the client cannot establish a connection to the server.

```
 XIO:  fatal IO error 73 (Connection reset by peer) on X server ″″
      after 0 requests (0 known processed) with 0 events remaining.
```

*Figure 56. Error Message: X Server is Running, but Connection was Broken*

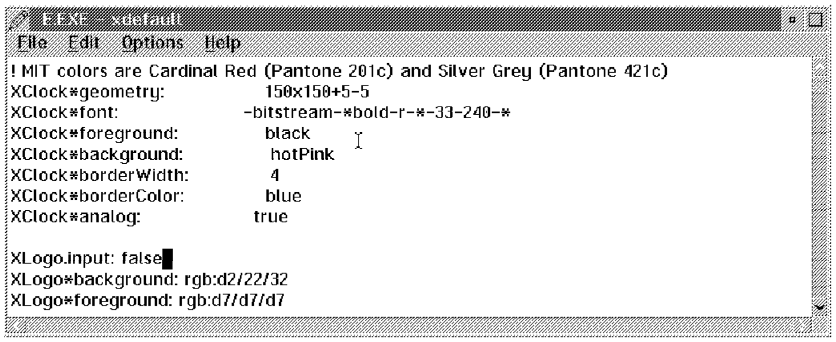
This error message indicates that the connection has been broken. Either the client window has been closed or the X server has been killed.

## 3.4 Running OS/2 X Window Clients and OS/2 OSF/Motif Applications

### 3.4.1 Application Resource File

The X Window System lets you modify certain characteristics of an application at run time by means of application resources. Typically, application resources are set to tailor the appearance and possibly the behavior of an application. The application resources may specify information about an application′s window sizes, placement, coloring, font usage, and other functional details.

On a UNIX** system, this information can be found in the user′s home directory in a file called Xdefaults. In the OS/2 environment, this file is called Xdefault and is in the \ETC subdirectory. Each line of this file represents resource information for an application. Figure 57 on page 104 shows an example of a set of resources specified for a typical X Window System application.



```
! MIT colors are Cardinal Red (Pantone 201c) and Silver Grey (Pantone 421c)
XClock*geometry:          150x150+5-5
XClock*font:            -bitstream-*bold-r-*-33-240-*
XClock*foreground:        black
XClock*background:         hotPink
XClock*borderWidth:        4
XClock*borderColor:       blue
XClock*analog:           true

XLogo.input: false
XLogo*background: rgb:d2/22/32
XLogo*foreground: rgb:d7/d7/d7
```

Figure 57. Entry of Xdefault

In this example, the Xclock application automatically creates a window in the lower left corner of the screen with a digital display in black letters on a pink background.

You can also use an application-specific resource file on an OS/2 workstation. The name of this file is set in the XENVIRONMENT variable. For example, if you wanted to use a special resource file for the Xclock application and the information was stored in the file C:TCPIPETCXCLOCK.AD, then you would include this command in your CONFIG.SYS or execute it from a command prompt:

```
SET XENVIRONMENT=C:TCPIPETCXCLOCK.AD
```

or:

```
xrdb -load xclock.ad
```

You can use the XCLISET.CMD as well to set this variable (as in the sample provided).

If you fail to use an application resource file, you may experience strange behavior from some of your X Window applications such as extremely large or peculiar fonts. This is caused by some poor guesses by the application in the absence of information normally provided from the application resource file.

There are several utilities provided to manipulate information in the application resource file. Listed below is a description of each of these utilities.

| **Utility** | **Description** |
|---|---|
| **XRDB** | X client application to get or set values in the Application Resource File. |
| **APPRES** | Lists the resources that currently might apply to a client. |
| **LISTRES** | Lists the resource database for one or more specified widgets. |
| **VIEWRES** | Displays the hierarchy of the widget set. You can expand each node in the display to see the resources associated with the node. |

These resources can also be set on the RESOURCE_MANAGER property of the X server. This property provides a single, central place where resources, which control all applications that are displayed on an X server, are found.

**Note:** Although the utility EDITRES is described in the documentation it is not used. The X Window System client uses the environment variable XUSERFILESEARCHPATH to locate the directories where application default files reside. With CSD UN60006 (newly available at the time of writing) updated documentation is available which describes this variable.

## 3.4.2 Running OS/2 X Window Clients and OSF/Motif Applications

The OS/2 X Window system is supplied with a number of X client programs and several utilities. They are provided to assist with customizing and administering the PMX server. There are also utilities to control a remote X Window System server and to start the OS/2 X Window System and other X Window System client applications. They are listed in Table 13 and Table 14 on page 106.

*Table 13 (Page 1 of 2). X Window Utilities*

| **Program** | **Description** | **Where to find more information** |
|---|---|---|
| XINIT | Starts the PMX | X Window System Server Guide |
| XPROB | Displays window and font properties | X Window System Server Guide, 3.4.2.3, “XPROP” on page 109 |
| XEV | Displays PMX events such as movement, re-sizing or text entry. | X Window System Server Guide |
| XCLISET | Set Environment Variables for X client | 2.6.5.1, “Setting Environment Variables” on page 49 |
| XFD | Displays the characters of a font | X Window System Server Guide, 4.2.7, “OS/2 X Fonts” on page 138 |
| XHOST | Controls client host access | X Window System Server Guide, 4.2.4, “Controlling X Client Access to OS/2” on page 132 |
| XLSFONTS | Generates a listing of fonts used on PMX | X Window System Server Guide |
| XFONTSEL | Prewiewing and Selecting Fonts | Does currently not work 1 |
| XMODMAP | Displays or alters the X keyboard modifier map and keysym table | X Window System Server Guide, 4.2.2, “Keyboard Definition” on page 127 |
| XSCOPE | Displays the X protocol activity between an X client and PMX | X Window System Server Guide |
| XSET | Sets characteristics for PMX | X Window System Server Guide, 4.2.7.1, “Adding New Fonts” on page 142 |
| XWININFO | Displays PMX window information | X Window System Server Guide 3.4.2.1, “XWININFO” on page 108 |
| XSTDCMAP | Defines colormap properties | X Window System Server Guide |

*Table 13 (Page 2 of 2). X Window Utilities*

| **Program** | **Description** | **Where to find more information** |
|---|---|---|
| XRDB | X Client application to get or set the application Resource File | X Window System Client Guide, 3.4.1, “Application Resource File” on page 103 |
| APPRES | Lists the resources that currently might apply to a client | X Window System Client Guide |
| LISTRES | Lists the resource database for one or more specified widgets | X Window System Client Guide |
| VIEWRES | Displays the hierarchy of the widget set. You can expand each node in the display to see the resources associated with the node. | X Window System Client Guide |

**Note:** Xfontsel had problems with displaying the defaults in the current release. We were unable to display the fonts; only the selection bar was displayed. After installing newly developed code including an update of the Xdefault file it works fine. This update should be available in the newly available CSD UN60006.

*Table 14 (Page 1 of 2). X Window System Client Programs*

| **Program** | **Description** | **Where to find more information** |
|---|---|---|
| XMPIANO | Sample for application developed with OSF/Motif widget | 3.4.2.6, “XMPIANO” on page 110 |
| XEYES | Eyes that watch your mouse pointer (background is viewable) | |
| XEDIT | Editor | X Window System User′s Guide 1 3.4.2.2, “XEDIT” on page 108 , Appendix E, “XEDIT Subcommands” on page 183 |
| MAZE | Labyrinth | |
| XCLOCK | Clock | 3.4.1, “Application Resource File” on page 103 |
| OCLOCK | Clock without tick marks | |
| XSETROOT | Setting root window characteristics | X Window System User′s Guide 1 2 |
| XCALC | Calculator Program | 3.4.2.4, “XCALC” on page 109, Appendix B, “Supplied Application Resource File Definitions for XCALC” on page 167 |
| XANT | 3270 terminal emulator | 3.4.2.5, “XANT” on page 109 |
| XLOGO | Opens a window with a ′X′ sign | |
| XHW | Opens a window with a ″Hello World″ message. This is a good test facility because it is very simple. | |

*Table 14 (Page 2 of 2). X Window System Client Programs*

| Program | Description | Where to find more information |
|---|---|---|
| XHELLO | Opens a window with a ″Hello World″ message under PMX. This is a good test facility because it is very simple. | |

**Note:**

**1** See bibliography at "Additional Publications" on page xix.

**2** PMX does not represent an X Window manager, so the PM color can′t be changed. You would need to use another platform, such as AIX, to test this function.

To set up an environment to run these applications follow these steps:

1. Start the X Window System server from the TCP/IP folder on your OS/2 desktop. This provides access to the resources that are shared among many X applications, such as the following:

   - Screen
   - Keyboard
   - Mouse
   - Fonts
   - Graphics contexts

2. Ensure that you have set your display environment variables. Whenever you start an X Window application, it will look up the variable DISPLAY in your OS/2 system environment. You can set this variable in the configuration notebook or from an OS/2 command prompt with this command:

   ```
   SET DISPLAY=9.24.104.51:0
   ```

   where 9.24.104.51 is our IP address and 0 is the screen on which we want to display the information.

   You can also display these applications on any X Window server. To define where the specific application is displayed depends on the setting of the DISPLAY environment variable. To display the application on the AIX X Window System server we had to set the variable to either of the following:

   ```
   SET DISPLAY=9.67.38.75:0
   SET DISPLAY=RS60007:0
   ```

3. Many of the commonly used X Window functions are stored in dynamic link libraries which are called by X Window applications at runtime. These dynamic link libraries are normally copied to your TCPIPDLL directory at installation. The files that contain the dynamic link libraries are:

   - Xaw.dll
   - oldX.dll
   - Xext.dll
   - Xmu.dll
   - Xlib.dll
   - Xt.dll

   Please ensure that these files are on your system.

You can start any of these applications from an OS/2 command prompt. In the following section you will find some samples of the utilities and X Window System client applications.

#### 3.4.2.1 XWININFO

To display information about a window use the command XWININFO.

```
OS2 D:\tcpip\bin>xwininfo

xwininfo: Please select the window about which you
          would like information by clicking the
          mouse in that window.


xwininfo: Window id: 0x800009 "xclock"

  Absolute upper-left X:  8
  Absolute upper-left Y:  609
  Relative upper-left X:  8
  Relative upper-left Y:  609
  Width: 150
  Height: 150
  Depth: 8
  Visual Class: StaticColor
  Border width: 0
  Class: InputOutput
  Colormap: 0x21 (installed)
  Bit Gravity State: NorthWestGravity
  Window Gravity State: NorthWestGravity
  Backing Store State: NotUseful
  Save Under State: no
  Map State: IsViewable
  Override Redirect State: no
  Corners:  +8+609  -866+609  -866-9  +8-9
  -geometry 150x150+8+609
```

*Figure 58. Sample Output from XWININFO Command*

#### 3.4.2.2 XEDIT

XEDIT is a simple text editor for X. It provides a window consisting of the following four areas:

1. Commands Section

   A set of commands that allows you to exit XEDIT, save the file, or load a new file into the edit window.

2. Message Window

   Displays XEDIT messages. In addition, this window can be used as a scratch pad.

3. Filename Display

   Displays the name of the file currently being edited, and whether this file is read-write or read-only.

4. Edit Window

Displays the text of the file that you are editing or creating.

Cursor and Command control are provided through several CTRL-key combinations. For further information for the editor see either *X Window System User's Guide* (see "Additional Publications" on page xix) or Appendix E, "XEDIT Subcommands" on page 183. For example see Figure 95 on page 147.

### 3.4.2.3 XPROP

If you need info about the window and font properties use the XPROP command.

```
OS2 C:\>xprop
WM_PROTOCOLS(ATOM): protocols  WM_DELETE_WINDOW
WM_CLASS(STRING) = "D:\\TCPIP\\BIN\\xclock.exe", "XClock"
WM_HINTS(WM_HINTS):
                Client accepts input or input focus: False
                Initial state is Normal State.
                bitmap id # to use for icon: 0x400001
                bitmap id # of mask for icon: 0x400003
WM_NORMAL_HINTS(WM_SIZE_HINTS):
                user specified location: 5, 605
                user specified size: 150 by 150
                window gravity: SouthWest
WM_CLIENT_MACHINE(STRING) = "ITSO51"
WM_COMMAND(STRING) = { "D:\\TCPIP\\BIN\\xclock.exe" }
WM_ICON_NAME(STRING) = "D:\\TCPIP\\BIN\\xclock.exe"
WM_NAME(STRING) = "D:\\TCPIP\\BIN\\xclock.exe"
```

*Figure 59. Sample Output XPROP Command*

### 3.4.2.4 XCALC

Figure 60 shows the calculator program running on the Workplace Shell:

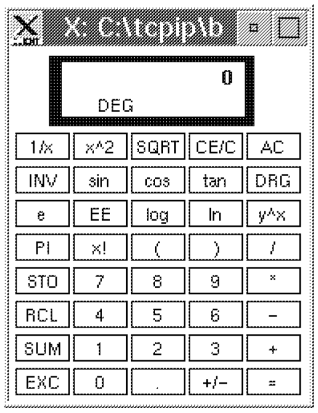


*Figure 60. Xcalc (OS/2 X Window Client Application)*

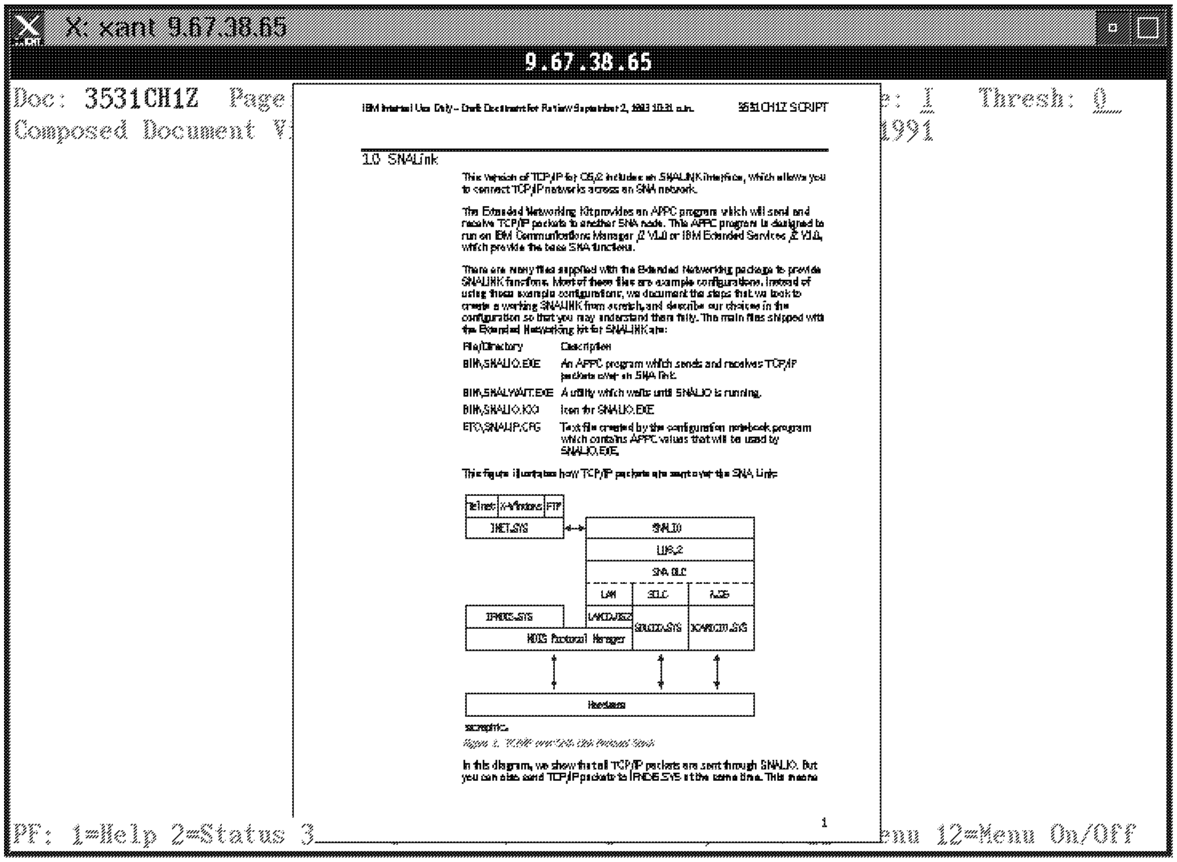
### 3.4.2.5 XANT

If you have access to a S/390 processor running TCP/IP for MVS or VM, then you should try running the X Window 3270 emulator supplied with X Window System client. Use this command to start the emulator:

```
XANT 9.67.38.65
```

where 9.67.38.65 is the address of the S/390 processor.

Figure 61 on page 110 shows the graphics capabilities of the XANT 3270 terminal emulator:


X: xant 9.67.38.65

9.67.38.65

Doc: 3531CH1Z Page … e: 1 Thresh: 0

Composed Document V… 1991

IBM Internal Use Only – Draft Document for Review September 2, 1993 10:31 a.m. 3531CH1Z SCRIPT

10 SNALink

This version of TCP/IP for OS/2 includes an SNALINK interface, which allows you to connect TCP/IP networks across an SNA network.

The Extended Networking Kit provides an APPC program which will send and receive TCP/IP packets to another SNA node. This APPC program is designed to run on IBM Communications Manager /2 V1.0 or IBM Extended Services /2 V1.0, which provide the base SNA functions.

There are many files supplied with the Extended Networking package to provide SNALINK functions. Most of these files are example configurations. Instead of using these example configurations, we document the steps that we took to create a working SNALINK from scratch, and describe our choices in the configuration so that you may understand them fully. The main files shipped with the Extended Networking kit for SNALINK are:

| File/Directory | Description |
|---|---|
| BIN\SNALIO.EXE | An APPC program which sends and receives TCP/IP packets over an SNA link. |
| BIN\SNALYWAIT.EXE | A utility which waits until SNALIO is running. |
| BIN\SNALIO.ICO | Icon for SNALIO.EXE |
| ETC\SNALIP.CFG | Text file created by the configuration notebook program which contains APPC values that will be used by SNALIO.EXE. |

This figure illustrates how TCP/IP packets are sent over the SNA Link:

screphic.

Figure 1. TCP/IP over SNA Link Protocol Stack

In this diagram, we show that all TCP/IP packets are sent through SNALIO. But you can also send TCP/IP packets to IPNDIS.SYS at the same time. This means

1

PF: 1=Help 2=Status 3… enu 12=Menu On/Off

Figure 61. Xant (OS/2 X Window Client Application)

When you install the OSF/Motif kit, an application called XMPIANO comes as a sample application developed using the OSF/Motif widget. To run this application you need to have the X Window System client installed at your workstation.

### 3.4.2.6 XMPIANO

On an OS/2 command prompt type XMPIANO and press Enter. You should see the following window at the X Window System server screen:

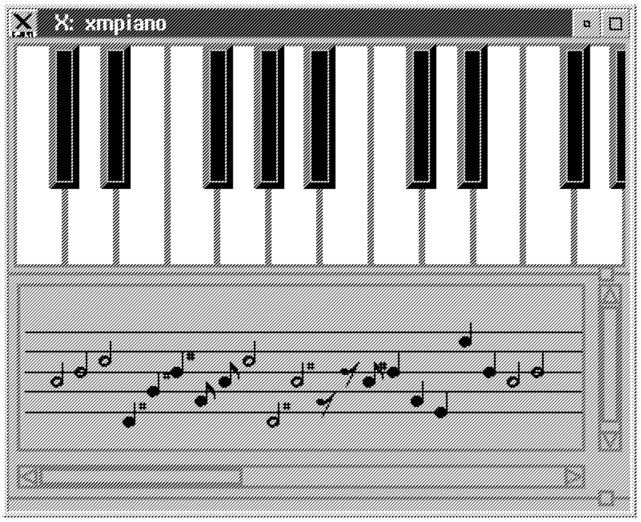


*Figure 62. OSF/Motif Application XMPIANO*

### 3.4.3 Developing of X Window Client and OSF/Motif Applications

The ability to run X client and Motif applications on your OS/2 workstation is provided by the X Client Runtime Services and the OSF/Motif Runtime Services, respectively. To develop such applications you also need X Client Programmer′s Toolkit and OSF/Motif Programmer′s Toolkit installed on your OS/2 workstation. For more information refer to 2.6, "Installing OS/2 X Window System Client and OS/2 OSF/Motif Kits" on page 46.

The X Window System client provides a set of application programming interfaces that enable you to create an X Window program which will use the X protocol to send and receive information to and from an X server for presentation on a screen. The X program communicates to an X server using sockets.

The X Window System client provides these components for development of an X client application for OS/2:

| **Component** | **Subdirectory** |
|---|---|
| **Library files** | LIB |
| **Sample Source Code** | SAMPLESX11 |
| **X Header Files** | INCLUDEX11 |
| **X Bitmap Files** | INCLUDEX11BITMAPS |

The Import Library files consist of:

| **File** | **Description** |
|---|---|
| **Xlibi.lib** | X |
| **Xti.lib** | X Intrinsics |
| **Xexti.lib** | X extensions |
| **Xmui.lib** | X Miscellaneous Utilities |
| **oldXi.lib** | X10 compatibility routines |
| **Xawi.lib** | X Athena Widgets |

The X Window System client requires that the following be installed and running on the OS/2 workstation for application development:

- TCP/IP Version 2.0
- High Performance File System (HPFS)
- A C 32 bit compiler (for instance, the IBM CSET++)

If you intend to develop OSF/Motif applications, you need the OSF/Motif kit, which includes:

- Motif library (Xm)
- Motif header files
- Motif resource manager library (Mrm)
- Motif resource manager header library
- A proof of licence

For more information on how to develop X Window System client and OSF/Motif applications, refer to *X Window System Client Guide*.

### 3.4.4 Tips for Porting Applications from UNIX

When porting an application from a UNIX system to an OS/2 system, you should consider these important recommendations:

1. The C Set/2 /sm switch is often needed to get the maximum migration allowances for constructs often found in UNIX C code.
2. The /ms switch is needed to ensure _System linkage instead of _Optlink linkage for all calls. When you use this switch, you must explicitly include header files. Many UNIX applications do not include these header files. Without them, the application will exit abnormally under OS/2 stating `exception in C library routine`.
3. Extensive use of signals must be avoided or rewritten.
4. Some UNIX system calls such as pipe and fork have no OS/2 equivalent. Try to simply avoid these applications and run the application in one process. If that is not possible, the application will likely need significant modification to run under OS/2.
5. Makefiles must be written to be compatible with NMAKE under OS/2. Refer to the samples directory provided with the X client product to see an example of an NMAKE compatible makefile for X Window applications.

**Note:** More information on this subject will be available in the documentation which will be shipped with the next CSD.

# Chapter 4. Customizing the X Server

## 4.1 AIX/6000 X Server

In the following section we describe how to customize the AIXwindows X server component. Although the Motif window manager is actually an X client application, we also describe how to customize it here with the server. This is because the Motif window manager is an integral part of the X server.

### 4.1.1 AIXwindows Environment/6000 V1.2

#### 4.1.1.1 Starting AIXwindows Environment/6000

Before we start customizing AIXwindows, we will show you how AIXwindows is started.

Generally there are three methods:

1. Entering X at the command line. This starts the X server on the local graphic terminal. No X client is started. Only the root window is displayed on your screen. You don′t have a root window menu provided by a window manager. You cannot enter any command because there is no command prompt available. You have to log in from another terminal to your host, so that you can start a client that connects to your X server.
2. Entering xinit at the command line. This will start both the X server and some X clients such as *xmodmap, aixterm, xclock*, and *mwm*. This is the normal way to start the X server at your display.
3. Entering startx at the command line. This command is a preferred alternative to xinit as it is streamlined and includes such functions as setting the user′s DISPLAY environment variable.
4. Entering mwm on an *aixterm* or *xterm* prompt starts the Motif window manager. This is useful when you want to restart a window manager or when your window should run remotely.

   **Note:** For this command to work correctly the X server must be running. This will not work if you already have started a window manager on your display.

We chose the xinit command for our further investigations.

The xinit command is a shell script which performs the following actions:

1. Selects the color database. You can specify your own color database with the `-D pathname` option, when you type the xinit command. The xinit command searches the color database in the following order:
   - *xinit -D pathname*
   - /usr/lib/X11/rgb.map.*

   **Note:** The files /usr/lib/X11/$LANG/rgb.*, /usr/lpp/X11/lib/X11/rgb.*, and /usr/lpp/X11/lib/X11/$LANG/rgb.* are links to the /usr/lib/X11/rgb.* files.
2. Defines the display variable for the server.
3. Starts the X server on the current display.

4. Executes the xinitrc file to start the X client programs. The *xinit* shell script searches the xinitrc file in the following order:

   - $XINITRC
   - $HOME/.xinitrc
   - /usr/lib/X11/$LANG/xinitrc
   - /usr/lpp/X11/defaults/$LANG/xinitrc
   - /usr/lpp/X11/defaults/xinitrc

5. Once the server has been started, xinitrc starts the first X client application. The xinitrc file is a shell script as well. It handles the keyboard mapping and starts the first clients including the window manager. The keyboard is assigned with the *xmodmap* command, which searches the appropriate files in the following order:

   - $HOME/.Xkeyboard
   - IMKEYMAPPATH/$KBD_LANG/keyboard
   - /usr/lpp/X11/defaults/xmodmap/$KBD_LANG/keyboard
   - /usr/lpp/X11/defaults/$LANG/keyboard

   For each country-specific keyboard there exists a file description in the directories /usr/lpp/X11/defaults/xmodmap/$LANG. The .Xkeyboard file is normally a copy of one of the files in these directories.

6. When the keyboard has been mapped, there are some X clients which will be started. At least one of the X clients should be *aixterm*. The starting of a window manager (*mwm*) is the last command in the xinitrc file.

7. The *mwm* (Motif window manager) reads some configuration files. These are the mwmrc file and the .Xdefaults file in which the resources for the window manager are stored. The window manager searches its configuration files in the following order:

   - For the mwmrc file:

     a. $HOME/.mwmrc
     b. /usr/lib/X11/$LANG/system.mwmrc if $LANG is defined
     c. /usr/lib/X11/system.mwmrc if $LANG is not defined

     **Note:** /usr/lib/X11 is a link to the /usr/lpp/X11/lib/X11 directory.

   - For the order of the resource file access please refer to 3.3.2, "Customizing Application Resources under AIX/6000" on page 99.

#### 4.1.1.2 Customizing AIXwindows Environment/6000

When you have started *xinit*, you can work with AIXwindows. We assume that the user is familiar with the window manipulation and knows how to move, scale, activate, close, and iconize a window, but there are some more features a user must know in order to work with AIXwindows.

There are files critical to proper AIXwindows customization. These files should be copied and into your home directory instead of changing the system defaults. This will allow other users of a particular system to customize their own windows according to their own requirements. Customizing system files limits users to system-wide constraints.

X Windows uses several files for initialization and the user can modify several of them to create his own environment. The following is a list of appropriate files for the AIXwindows customization:

| | |
|---|---|
| /usr/lpp/X11/defaults/Xdefaults.tmpl | A sample file which contains resource information for some X clients. |
| /usr/lib/X11/app-defaults/Mwm | This file contains default Motif settings. Other default files for X applications reside in the app-defaults directory. |
| /usr/lib/X11/system.mwmrc | This file contains the system defaults for the AIXWindows startup. |
| /usr/lpp/X11/defaults/xinitrc | Contains the appropriate default AIXwindows resource information. |
| /usr/lib/X11/rgb.txt | Contains a list of default colors available. Each color has a set of corresponding numbers determining the intensities for red, green, and blue. These colors can be used to customize items in the xinitrc and Xdefaults files. |
| /usr/include/X11/bitmaps | Is the directory where the bitmaps reside, which you can use for defining your background, icons, etc. |
| /usr/lpp/X11/fonts | Contains a set of fonts usable with AIXwindows. |

With the following commands we can create a specific environment for each user on the system in the /usr/lib/X11/rgb.txt file:

```
    cp /usr/lpp/X11/defaults/Xdefaults.tmpl $HOME/.Xdefaults
    cat /usr/lib/X11/app-defaults/Mwm >>$HOME/.Xdefaults
    cp /usr/lib/X11/system.mwmrc $HOME/.mwmrc
    cp /usr/lpp/X11/defaults/xinitrc $HOME/.xinitrc
```

To specify another font you can use the xlsfonts command, which gives a listing of all the fonts available on your system.

#### 4.1.1.3 Loading xrdb

You have to decide which resources you want to take effect when you display a client′s output on your screen. There are two ways:

1. If you don′t invoke the xrdb command when you start the X server on your workstation, the clients that display on your server read their resources from their own workstations. That means if you start several aixterms each from another machine, it may be that each background color is different, because the resource definition varies on each workstation.

2. When you start xrdb with a resource file (which may be .Xdefaults) the same aixterms will have the same background color. This is the case if you have defined that resource in your resource file and you have not started each aixterm with a different -bg (background color) option.

### 4.1.2 Customizing Motif Window Manager

The customization of the Motif window manager (called mwm) is controlled in two ways:

- Through a special file called .mwmrc copied in your home directory.
- Through mwm resources that you can specify in your .Xdefaults file.

The default operation of mwm is largely controlled by a system-wide file called system.mwmrc, which establishes the contents of the Root menu and Window menu, how menu functions are invoked, and what key and button combinations can be used to manage windows. Normally the .mwmrc file in your home directory is a copy of the system.mwmrc file with user-specified changes.

Let us take a closer look at the following topics:

- The menus and how menus are invoked
- The keyboard focus policy
- How icons are organized

If you have changed your .mwmrc or .Xdefaults file, the changes will not take effect automatically. When you change the .mwmrc file you have to restart the Motif window manager (mwm). This can be done by pressing the Shift Ctrl Alt ! key combination twice. If you have changed resource definitions in the .Xdefaults file for the window manager, restart the window manager too. If you are working with the xrdb RESOURCE_MANAGER your resource definitions can be loaded dynamically without restarting the window manager. This can be done with the following command:

```
xrdb -load .Xdefaults
```

This will update the resource settings previously restored in the resource database. In that case we used the .Xdefaults file as resource file for xrdb. If you don′t work with xrdb you can use the same .Xdefaults file as resource file for your clients.

The system.mwmrc file can be divided into three sections:

- Menu specification (root menu, window menu)
- Key Bindings
- Button Bindings

***Customizing the Menu Specifications:*** mwm has a number of predefined functions. Each of these has a name beginning with f. The following is an example with a cascaded menu and using bitmaps.

```
Menu TCP/IPMenu
{
  ″TCP/IP″              f.title
  @˜/host.px            f.exec ″aixterm -e tn3270 mvs18 &″
  ″TN VM″               f.exec ″aixterm -e tn3270 vm14 &″
  ″TN RISC″             f.exec ″rexec rs60002 aixterm -display rs60001:0 &″
  ″FTP OS/2″            f.exec ″aixterm -e ftp paulb &″
}

Menu ToolsMenu
{
    ″Tools″         f.title
    ″mouse slow″    f.exec ″xset m 1″
    @˜/test.bm      f.exec ″xset m 4″
    @shellk.bm      f.exec ″aixterm &″
}
```

*Figure 63. Menu Specifications for the Root Menu*

The syntax for entries in the root menu as shown in Figure 63 is as follows:

```
″label″   function
```

**Notes:**

1. To include bitmaps in your root menu, you have to replace the ″label″ with the @ character followed by the bitmap filename. @~ stands for your home directory. When you use only the @ character, mwm searches in the bitmap directory which is defined by the bitmap directory resource:

   ```
   Mwm*bitmapDirectory: <path>
   ```

2. Be aware that when you use a bitmap from the AIXwindows X.desktop, they are in a bitmap format (most of them are in pixmap format). You can convert those maps using the bitmap icon editor from the desktop. Sometimes you must change the color attribute, because the icon will be shown as transparent in your root menu.

```
# Default Window Menu Description

Menu DefaultWindowMenu MwmWindowMenu

{
    ″Restore″   _R      Alt<Key>F5      f.normalize
    ″Move″      _M      Alt<Key>F7      f.move
    ″Size″      _S      Alt<Key>F8      f.resize
    ″Minimize″  _n      Alt<Key>F9      f.minimize
    ″Maximize″  _x      Alt<Key>F10     f.maximize
    ″Lower″     _L      Alt<Key>F3      f.lower
    no-label                            f.separator
    ″Close″     _C      Alt<Key>F4      f.kill
}
```

*Figure 64. Definitions for the Window Menu with Accelerator*

The syntax for entries in the window menu as shown in Figure 64 is as follows:

```
″label″   mnemonics   accelerator   function
```

where:

**″label″** is the character string that appears when Window Menu is invoked.

**mnemonics** is the underlying part of the label (optional). Once the window is displayed, you can select an item by typing its mnemonic abbreviation.

**accelerator** is the key combination used for invoking the action (optional).

```
# no accelerator window menu

Menu NoAccWindowMenu
{
    ″Restore″   _R      f.normalize
    ″Move″      _M      f.move
    ″Size″      _S      f.resize
    ″Minimize″  _n      f.minimize
    ″Maximize″  _x      f.maximize
    ″Lower″     _L      f.lower
    no-label            f.separator
    ″Close″     _C      f.kill
}
```

*Figure 65. Definition for the Window Menu without Accelerator*

***Key Bindings:*** The following text defines the key bindings.

```
#
# key binding descriptions
#

Keys DefaultKeyBindings
{
    Shift<Key>Escape            icon|window             f.post_wmenu
    Meta<Key>space              icon|window             f.post_wmenu
    Meta<Key>Tab                root|icon|window        f.next_key
    Meta Shift<Key>Tab          root|icon|window        f.prev_key
    Meta<Key>Escape             root|icon|window        f.next_key
    Meta Shift<Key>Escape       root|icon|window        f.prev_key
    Meta Ctrl Shift<Key>exclam  root|icon|window
f.set_behavior
#    Meta<Key>Down              root|icon|window        f.circle_down
#    Meta<Key>Up                root|icon|window        f.circle_up
    Meta<Key>F6                 window f.next_key transient }
```

*Figure 66. Key Bindings*

The syntax for the key bindings as shown in Figure 66 is as follows:

```
[modifier_keys]<Key>key_name     context   function
```

**Notes:**

1. The useful contexts for the key bindings are root, window, and icon.
2. Meta is equal to the Alt key.

The keyboard shortcuts can be invoked when the pointer is within the defined context. You can define the keyboard shortcuts for more than one context.

***Button Bindings:*** The following text defines the default button bindings.

```
#
# button binding descriptions
#

Buttons DefaultButtonBindings
{
    <Btn1Down>          frame|icon      f.raise
    <Btn3Down>          frame|icon      f.post_wmenu
    <Btn1Down>          root            f.menu  RootMenu
    <Btn3Down>          root            f.menu  RootMenu
    Meta<Btn1Down>      icon|window     f.lower
    Meta<Btn2Down>      window|icon     f.resize
    Meta<Btn3Down>      window          f.move
}

Buttons ExplicitButtonBindings
{
    <Btn1Down>          frame|icon      f.raise
    <Btn2Down>          frame|icon      f.post_wmenu
    <Btn3Down>          frame|icon      f.lower
    <Btn1Down>          root            f.menu  RootMenu
    Meta<Btn1Down>      window|icon     f.lower
    Meta<Btn2Down>      window|icon     f.resize
    Meta<Btn3Down>      window|icon     f.move

}

Buttons PointerButtonBindings
{
    <Btn1Down>          frame|icon      f.raise
    <Btn2Down>          frame|icon      f.post_wmenu
    <Btn3Down>          frame|icon      f.lower
    <Btn1Down>          root            f.menu  RootMenu
# If (Mwm*passButtons == False)
    Meta<Btn1Down>      window|icon     f.raise
# Else
#    <Btn1Down>          window         f.raise
#    Meta<Btn1Down>     window|icon     f.lower
    Meta<Btn2Down>      window|icon     f.resize
    Meta<Btn3Down>      window|icon     f.move
}
```

*Figure 67. Button Bindings*

The syntax for a button specification is very similar to that of a key binding:

```
    [modifier-key]<button-event>    context    function
```

Each button binding can have one or more modifier keys (modifiers are optional) but a single button event invokes the function. For button bindings the valid contexts are:

|  |  |
|---|---|
| **root** | Background window |
| **window** | Includes the application windows and the frame |
| **icon** | Icon area |
| **title** | Title area of the frame |
| **border** | Frame (excluding titlebar) |
| **frame** | Entire frame (title and border) |
| **app** | Window exclusive frame |

You can run AIXwindows Environment/6000 without customization. But normally every user has his or her requirements. In the following sections we will investigate the workstation′s environment to satisfy the user′s requirements.

### 4.1.3 Customizing Colors

The colormap provides a translation between pixel values writing color images into workstation displays. Many hardware displays have a single colormap. It depends on the graphic adapter how many colormaps are supported. Because colormaps are associated with windows, AIXwindows supports displays with multiple colormaps and different types of colormaps (8-bit, 24-bit). A colormap is a collection of colorcells. A colorcell consists of a triple of red, green, and blue. As each pixel is read out of the display memory, its value is taken and looked up in the colormap. The values of the cell determine what color is displayed on the screen. The number of bits per pixel indicates the depth of the display. When the depth is 8 bits the values 0 through 255 are defined, which means the display can be 256 colors at one time. Some adapters allow the use of more than one colormap.

The visual describes the color capability of the workstation. It is an interface between the device-dependent display and the application program. There are different strategies for translating pixel values into colors. These strategies are called *visuals*. AIXwindows defines six different visuals. We can set the visual class by starting the X Server. The default visual class is PseudoColor for color displays and GrayScale for monochrome displays. On the IBM RISC System/6000 you cannot set the visual class for each window.

The visual is the description of the colormap for the application. There are six different visuals available:

|  |  |
|---|---|
| **0** | StaticGray |
| **1** | GrayScale |
| **2** | StaticColor |
| **3** | PseudoColor |
| **4** | TrueColor |
| **5** | DirectColor |

The frame buffer is the display memory in which each pixel is stored. The range for each pixel is from 1-bit up to 24-bit. The value of the pixel is represented as an index to the colormap. For example, the frame buffer of our graphics color adapter has 8 bits for each pixel. When the resolution is 1024 pixels vertical and 1280 pixels horizontal the size of the frame buffer memory is approximately 1.5MB. This adapter can display 256 colors at one time.

The colormap is an array of 256 entries in length. Each entry represents the value for the colors red, green, and blue. Each entry is 24-bit. In this way we can display 256 colors at a time, out of 16777216 possible colors.

To determine the color names, the X server (which loads the colormap) uses a color database. These are the files /usr/lib/X11/rgb*. The printable copy is stored in the rgb.txt file.

On many workstations the display is limited to 256 or fewer colors, and most workstations have only one hardware lookup table for colors. In this case only one application colormap can be installed at a given time. Every application can allocate and install a new colormap. However, a color that is right for one application may be wrong for another application which was based on the colormap previously defined on the system.

You can define your own color database or change the existing database. We built our own color table. The following are the steps for creating your own color database:

1. Enter `cd /usr/lib/X11`
2. Create the file color.txt

```
255     255      255      white
  0       0        0      black
255       0        0      red
  0     255        0      green
  0       0      255      blue
```

*Figure 68. A Sample Color Table*

   Before we invoked the rgb command, we saved rgb.pag and rgb.dir:

```
    mv rgb.pag rgb.pag.ori
    mv rgb.dir rgb.dir.ori
```

3. Enter `rgb <color`
4. Enter `mv rgb.pag color.pag`
5. Enter `mv rgbg.dir color.dir`
6. Now you are ready to start the X server with your own color database by entering the following command:

```
    xinit -D /usr/lib/X11/color
```

**Note:** It is possible that you will see more than these five defined colors on your screen. The reason is that the Motif window manager has defined its own color resources, and therefore has more colors available than you have defined.

### 4.1.4 AIX/6000 X Fonts

Each font that AIXwindows can use is stored in a file. The fonts files are located in one of several directories or in fact may be obtained from a font server.

Please refer to 4.2.7.2, "Using a Font Server" on page 143 for more information about the font server and an example of its use by an X server. When an X client requests a server to load a font, the X server searches in the defined font directories. The default font paths for the AIXwindows server is the /usr/lpp/X11/lib/X11/fonts directory.

**Note:** There is also a path /usr/lib/X11/fonts, but it is linked to the /usr/lpp/X11/lib/X11 font directory.

The font files may have different extensions, as listed below:

**snf** Server normal font. This is the normal font used by the X server. These fonts will normally not work when ported to another server.

**snf.Z** As the font files can be very large, the X server can load compressed font files.

Sources from fonts are usually in BDF format (Bitmap Distribution Format), if you want to use fonts which are not already on your local X-Window server you can either get those BDF Files via FTP and compile them with the bdftopcf utility or use a font server. Both methods are described in the next chapters.

**pcf** Portable compiled font.

The SNF fonts used in earlier levels of AIX have been replaced in V3.2.5 by PCF fonts. The background is the change from X11R4 to X11R5: in X11R4 the SNF (Server Normal Fonts) fonts were used while X11R5 has switched to PCF (Portable Compiled Font). The advantage of PCF fonts compared to SNF fonts is the capability to transfer compiled font files, which eliminates compilation on each X Window server and gives the capability of using a font server.

Note that .bdf files may be converted with the bdftosnf utility into .snf font files. Also, an additional utility, bdftopcf, is provided for converting .bdf files to .snf font files.

There are two other files in a font directory:

**fonts.dir** Every font has a font ID (the XLFD description) which identifies a font. In the fonts.dir file each XLFD description (font ID) maps to one font file. When you start an X application that uses a font, you have to specify the font ID. The client sends the font ID to the server that loads the font file which maps the font ID. If the server doesn′t find the font ID in its fonts.dir files, it generates an error message on the screen and chooses a default font instead. You can generate a fonts.dir file for a specific font directory with the mkfontdir command.

**fonts.alias** This is a file you have to edit yourself. It maps each font ID to an alias. With this alias name you can specify a font for an X client on this server. An alias is the logical name for a font. You can set several fonts for one physical font. This is useful when a client asks for a font that is not implemented on your server, and you select the font which is the closest to the one requested and give it the alias name of the requested font.

#### 4.1.4.1 Utilities for Customization

There are some very useful clients which you can use in conjunction with the font customization.

| | |
|---|---|
| **xlsfonts** | lists all font IDs of the fonts that you have installed on your workstation. |
| **xfd** | displays the printable character of a font. |
| **mkfontdir** | is used when you install new fonts on your X server. It creates a new version of fonts.dir. |
| **xfontsel** | is a client you can use when you have to extract a font from the XLFD description. xfontsel will give you the font name. |
| **showsnf** | prints the contents of a .snf font file. |
| **xset** | sets option for the X server. |

There may be some font resources in your .Xdefaults file for customizing the default fonts for specific X clients.

#### 4.1.4.2 ISO Codesets for Fonts

The standard MIT fonts use the ISO 8859 code set, whereas IBM′s default code set for the RISC System/6000 is 850. AIXwindows Environment/6000 is delivered with both the IBM PC 850 codeset and the ISO 8859 codeset. You should migrate to the ISO codeset when you are working in a heterogeneous X Window System environment.

#### 4.1.4.3 Installing Fonts or Updating fonts.dir

Perform the following steps:

1. Copy the fonts file in the appropriate directory (.snf, .bdf, compressed file):

   ```
   mkfontdir <directory>
   ```

   **Note:** mkfontdir decompresses and converts the files into .snf format.

2. Edit the fonts.alias file (optional).
3. If you have created a new directory, run:

   ```
   xset +fp <directory>
   xset fp rehash
   ```

### 4.1.5 Remapping the Keyboard Under AIX/6000

AIXwindows provides the X client xmodmap for mapping a keyboard. xmodmap is used when AIXwindows Environment/6000 is active and a user wants to initiate a change in the keyboard mapping. The keymaps are set during the execution of the xinitrc shell script by using the xmodmap command. For every window you start, the current definition of the keyboard map will be active.

It is possible to change the keyboard mapping dynamically. You can set another keymap for a new window on your display. Every time you invoke the xmodmap command on your server the changes will take effect for the new X clients that you will start.

There are several keymap files included in AIXwindows Environment/6000. The default language keymap files are located in the directory /usr/lpp/X11/defaults/xmodmap/$LANG. The $LANG environment variable has two styles:

| | |
|---|---|
| **Xx.XX** | This means the IBM PC 850 code page |
| **xx.XX** | This represents the ISO 8859 code page |

The codeset of your keyboard and the fonts you are using within a client should be the same. The keyboard.alt file defines the RT PC (IBM 6150) keyboard.

To change your keymap on your display, enter the command:

```
    xmodmap /usr/lpp/X11/defaults/xmodmap/<LANG>/keyboard
```

at the command line of an X client on your display. The next X client you start will use the new keymap; however, the old X clients use the previously defined keymap. In this way you can have several keymaps on your display.

**Note:** <LANG> means the language environment that you want to use.

***Customizing your own keymap file:*** There are different ways to change the keymaps. The easiest way to customize your own keymap file is by copying an existing file and changing the entries. Let us have a closer look at an example of a keymap file.

First, we need to know the following expressions:

| | |
|---|---|
| **Keycode** | The assignment of a number to a physical key. |
| **Keysym** | Standard symbolic encoding for the symbols engraved on keyboard keys. |
| **Modifier** | These are logical key names which can be assigned to keysyms. X Windows defines eight modifiers: Shift, Ctrl, Lock, Mod1, Mod2, Mod3, Mod4, Mod5. They are called modifier keys, because they modify the action of other keys. When users press the modifier key Shift which is assigned to the Shift key, and press the keycode for a character, they get a capital character. |

The X Window System was designed as a device-independent system. With this design the X Window System is not concerned with the layout of the keyboard. Every application should be able (with the right keymaps) to understand the keys pressed by the user. To support all the different keyboards, there are three translation processes.

Pressing a key on the keyboard generates a keycode. This keycode is unique to the appropriate key. The workstation generates a keycode event and reports this to the application. Keycodes are numbers in the range from 8 to 255. The application translates the keycode into a keysym where the application knows if the user presses the Shift key or another modifier at the same time. The third step is the translation from the keysym into a string.

The keyboard file has the following syntax:

```
!  keycode  xxx = keysym0   keysym1    keysym2
```

where:

| | |
|---|---|
| **!** | Comments the line |
| **xxx** | Value from 8 - 255 |
| **keysym0** | Keysym without any modifier |
| **keysym1** | Keysym with modifier Shift pressed |
| **keysym2** | Keysym with modifier AltGr (right Alt key on keyboard) |

**Note:** Some keycode lines are commented out. These are keycodes which are the same on every IBM keyboard available. However, you can change these keysyms assigned to those keycodes (change the keysym and delete the !).

The list of the defined keysyms is defined in the /usr/include/X11/keysymdef.h file.

To see the definitions of your keyboard (keycode and assigned keysyms) you can use the xev command. This is an X client which is delivered in source form. Enter xev at the command prompt of an *aixterm*. A new window appears on the display. Activate the window and press any keys. On the aixterm you see the information which shows which keycode you pressed and the keysym assigned to the keycode. With this client you can verify if all keycodes are recognized and to which character or string they are translated.

To change the keymap, edit either the keyboard file and run xmodmap, or make your change at the command prompt.

## 4.1.6 Controlling X Client Access to AIX/6000

The X Window System is designed to allow every client to connect to any server in the network. To restrict the access to the server, AIXwindows Environment/6000 provides two mechanisms to set or reset the host authorization. The authorization is only host based. If you grant access to a host, every user on this remote host can connect to your server and display a window on your screen.

- One way to show, set, or reset the X server access is provided by the xhost command. To invoke the command, the X server must be started. As xhost is an X client, the command must be invoked locally on the host which runs the X server. When you stop the X server the authorization information will be lost.
- The other way is by manipulating the /etc/Xn.hosts file where n means the server number (usually 0) on your X server system. You must have root authority to change this file. Each entry in that file represents a host allowed to connect to your server.

  **Note:** When you create this file, if it doesn′t exist, ensure that the file name begins with an X. The file system in AIX/6000 is case sensitive and files whose name begins with an x will never be accessed. Each time you start the server this file is scanned and all hosts for which an entry exists are allowed to connect to the server.

### 4.1.7 Interoperability

The key points for a successful connection establishment are:

- A TCP/IP connection to the X server host.
- The X server must be started.
- The X client must be allowed to connect to the X server host.

There are a number of ways of establishing a connection to the client platform and starting the X client application from the AIX/6000 X server. The user can log in to a remote system, or he can execute a command via rexec or rsh to start a client application on his or her display.

*Table 15. How to Connect to a Client System*

| **Connection** | **Supported** |
|---|---|
| AIX - AIX | rlogin, telnet, rexec, rsh |
| AIX - VM | tn3270, rexec |
| AIX - MVS | tn3270 |

## 4.2 Customizing PMX

The core component of the OS/2 X Window System is the PMX window manager and therefore we will know consider the customizing of PMX.

### 4.2.1 Using the Configuration Notebook Program to Configure PMX

This section explains how to configure PMX using the configuration notebook or the PMX configuration option. It also directs you to the information you need to configure PMX manually.

To access configuration information for PMX with the configuration notebook, double-click with mouse button 1 on the TCP/IP Configuration icon in the TCP/IP folder. To use the PMX configuration option, select the Configuration option from the Commands pull-down menu. Then select Initial Settings.

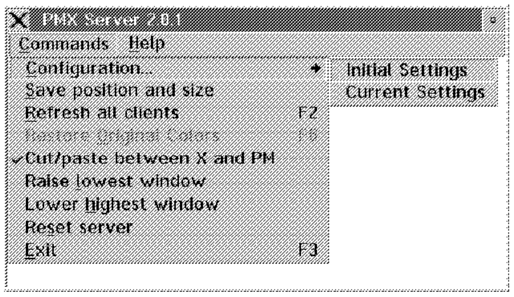


*Figure 69. Using the PMX Configuration Options*

To save your changes, close the configuration notebook. If you used the X Window System server configuration menu, some settings will take effect immediately. If you used the configuration notebook, you must restart the X Window System server for the changes to take effect.

The configuration panels you find in the configuration notebook and in PMX itself are identical; the same parameters are affected.

Configuring PMX using the configuration menu or the configuration notebook is more convenient than configuring it manually, which involves editing configuration files and entering parameters.

**Note:** Many of the configuration options can be set at the X Window System server, the Configuration notebook and the XINIT command lines. If you set parameters in all places, the command line parameters will override the options set in the configuration notebook or PMX Configuration option.

## 4.2.2 Keyboard Definition

Figure 70 shows the Keyboard Options window. Use this page to specify what keyboard you are using.

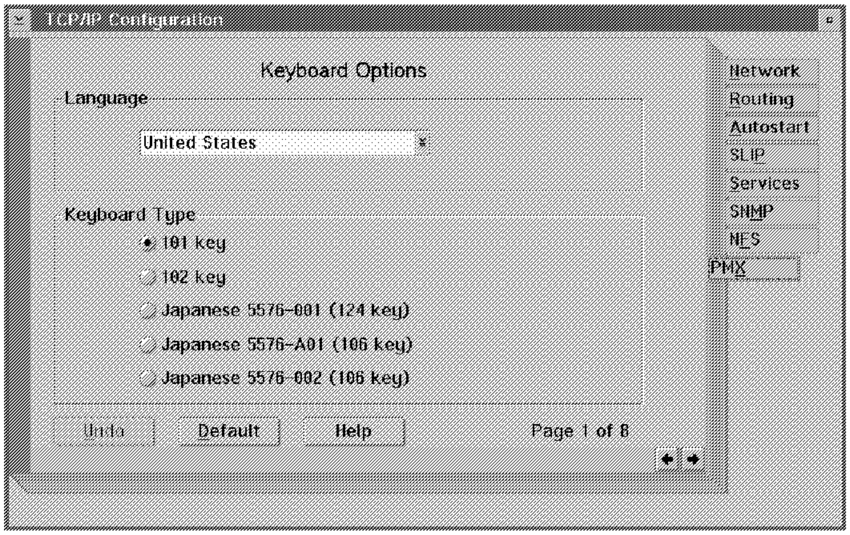


*Figure 70. Keyboard Options*

Selecting the 101 key option sets the keyboard type to a 101-key keyboard. Most USA Keyboards have 101 keys. Most European keyboards have 102 keys.

Use the language field to select the setting for PMX. This setting will be used by XINIT.CMD at startup to configure the language for your keyboard.

PMX provides support for non-USA keyboards. You can define your country's keyboard specifications. PMX uses the XMODMAP utility to set up your keyboard language.

A simple way to use the new keyboard feature:

Define a new environment variable called LANG, with the appropriate language as its value (see the example in the Table 16). If it is not defined, the En_US keyboard is assumed.

*Table 16 (Page 1 of 2). X Window Utilities*

| Language | Keyboard |
|---|---|
| Belgian | nl_BE |
| Belgian French | fr_BE |
| Canadian French | fr_CA |

*Table 16 (Page 2 of 2). X Window Utilities*

| **Language** | **Keyboard** |
|---|---|
| Danish | da_DK |
| Dutch | nl_NL |
| Finnish | fi_FI |
| French | fr_FR |
| German | de_DE |
| Greek | el_GR |
| Icelandic | is_IS |
| Italian | it_IT |
| Japanese | ja_JP |
| Japanese English | en_JP |
| Latin American Spanish | es_LA |
| Norwegian | no_NO |
| Portuguese | pt_PT |
| Spanish | es_ES |
| Swedish | sv_SV |
| Swiss French | fr_CH |
| Swiss German | de_CH |
| Turkish | tr_TR |
| United Kingdom | en_GB |
| United States | en_US |

The table shows the national language keyboard definitions that are distributed. The keyboard definitions shipped with the PMX kit are copied unchanged from RS/6000 AIX. For example, if you want the German keyboard definitions definitions, you could add the following line to your CONFIG.SYS file:

```
set LAN=de_DE
```

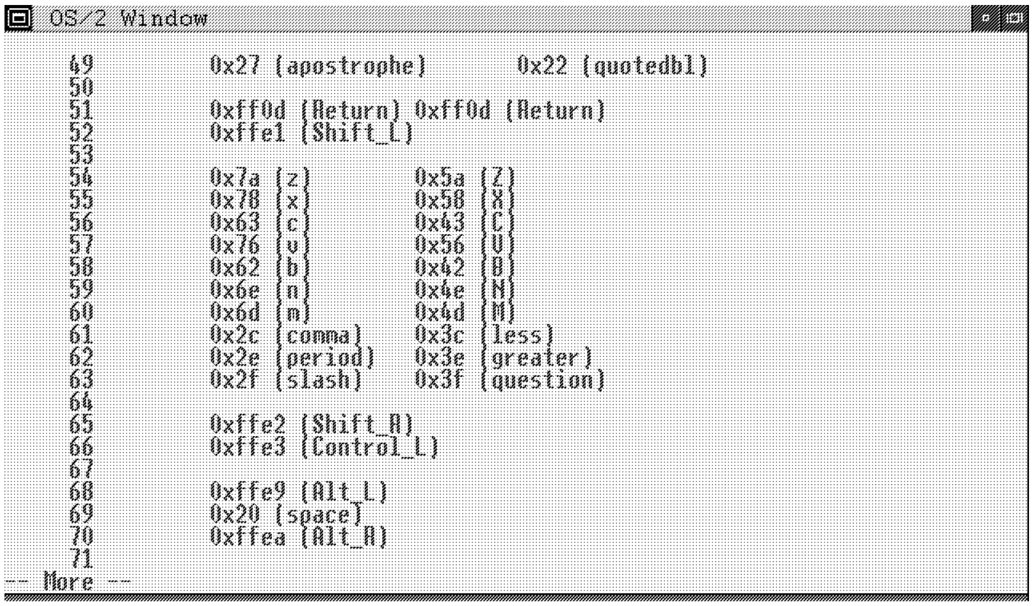
#### 4.2.2.1 Remapping the Keyboard under OS/2

Use the following example as a guide to altering the keyboard mapping for the OS/2 X server.

1. You can use the PMX utility XMODMAP to find out the current keyboard settings on your OS/2 X server. Issue the following command from an OS/2 command prompt:

   ```
   XMODMAP -pk | more
   ```

   Figure 71 on page 129 shows the output of the XMODMAP command.



```
     49         0x27 (apostrophe)       0x22 (quotedbl)
     50
     51         0xff0d (Return) 0xff0d (Return)
     52         0xffe1 (Shift_L)
     53
     54         0x7a (z)        0x5a (Z)
     55         0x78 (x)        0x58 (X)
     56         0x63 (c)        0x43 (C)
     57         0x76 (v)        0x56 (V)
     58         0x62 (b)        0x42 (B)
     59         0x6e (n)        0x4e (N)
     60         0x6d (m)        0x4d (M)
     61         0x2c (comma)    0x3c (less)
     62         0x2e (period)   0x3e (greater)
     63         0x2f (slash)    0x3f (question)
     64
     65         0xffe2 (Shift_R)
     66         0xffe3 (Control_L)
     67
     68         0xffe9 (Alt_L)
     69         0x20 (space)
     70         0xffea (Alt_R)
     71
-- More --
```

*Figure 71. Issuing the XMODMAP -pk Command*

Each physical key has an assigned keycode. For example the Enter key has a keycode of decimal 51. The key function is identified by the keysym. Notice that in Figure 71 the keysym supplied for keycode 51 is Return.

2. In order to change the keysym of keycode 51 from Return to Execute you need to create an input file for the XMODMAP command. We created a file called MAP which had a one-line entry as illustrated in Figure 72.

```
keycode 51 = Execute
```

*Figure 72. An Input File for XMODMAP*

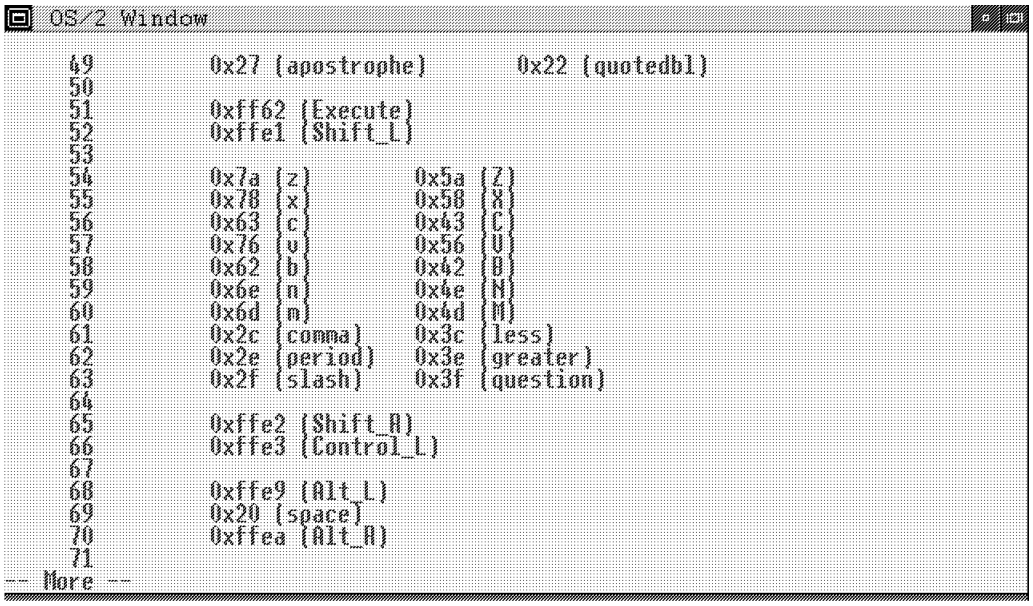
3. Issue the following command from the OS/2 command prompt:

```
XMODMAP MAP
```

4. Issue the following command again to check that the keysym for keycode 51 has in fact changed to Execute:

```
XMODMAP -pk | more
```

In Figure 73 on page 130 you can see that keycode 51 has in fact been changed to EXECUTE.



```
OS/2 Window
     49         0x27 (apostrophe)       0x22 (quotedbl)
     50
     51         0xff62 (Execute)
     52         0xffe1 (Shift_L)
     53
     54         0x7a (z)        0x5a (Z)
     55         0x78 (x)        0x58 (X)
     56         0x63 (c)        0x43 (C)
     57         0x76 (v)        0x56 (V)
     58         0x62 (b)        0x42 (B)
     59         0x6e (n)        0x4e (N)
     60         0x6d (m)        0x4d (M)
     61         0x2c (comma)    0x3c (less)
     62         0x2e (period)   0x3e (greater)
     63         0x2f (slash)    0x3f (question)
     64
     65         0xffe2 (Shift_R)
     66         0xffe3 (Control_L)
     67
     68         0xffe9 (Alt_L)
     69         0x20 (space)
     70         0xffea (Alt_R)
     71
-- More --
```

*Figure 73. Issuing the XMODMAP -pk Command with Keycode 51 Set to EXECUTE*

To satisfy the requirements of the different countries we recommend the following solution:

1. Add the entry:

   ```
   SET LANG=XX_XX
   ```

   in your CONFIG.SYS file. The variable XX_XX means the country you want to specify for your keyboard layout. We used SET LANG=GR_SW. For each country-specific keyboard there is a directory X11\DEFAULTS\XMODMAP\XX_XX, which contains the keyboard file.

2. Start the PMX server by invoking the BIN\XINIT.CMD command. Make sure that you have rebooted the system after adding the language entry in the CONFIG.SYS file.

To start some X clients each time you start the PMX server, add the following entry at the bottom of the BIN\XINIT.CMD command file:

```
rexec RS60001 -l stephan -p stephan aixterm -display PAULB:0 -fn rom11 &
```

where PAULB is the PMX server's host name.

On this X client we used the Swiss German keyboard layout.

If you want to change the keyboard layout you can do it at any time (after you have started PMX), but these changes only take effect for the clients you will start after these specific changes. One exception is when you change the button codes of your pointing device. To reverse the button codes that get generated so that the primary button is pressed using the index finger of the left hand can be done on a three-button pointer as follows:

1. Show the current pointer map with the command:

```
xmodmap -pp
```

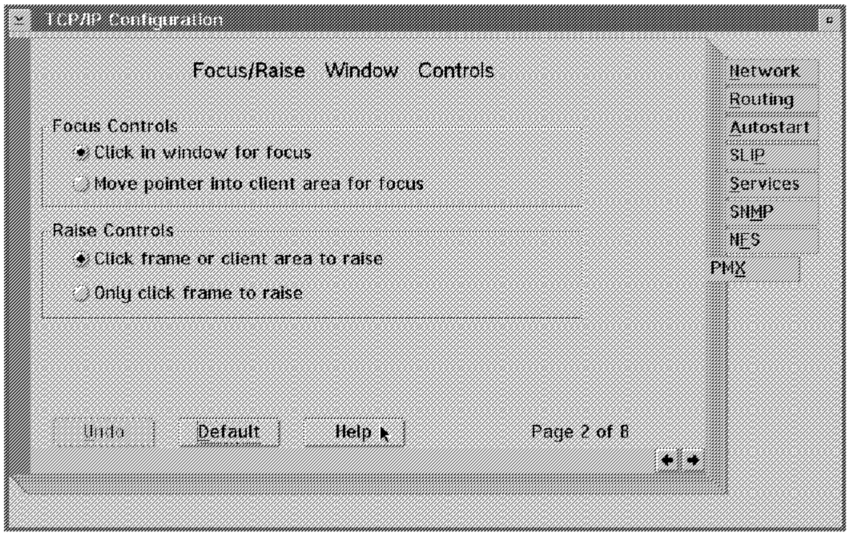
2. Enter the command:

```
xmodmap -e ″pointer = 3 2 1″
```

3. Show the current pointer map with the command:

```
xmodmap -pp
```

## 4.2.3 Window Control



Focus/Raise Window Controls

Focus Controls
- (●) Click in window for focus
- ( ) Move pointer into client area for focus

Raise Controls
- (●) Click frame or client area to raise
- ( ) Only click frame to raise

*Figure 74. Window Controls*

Figure 74 shows the Window Controls page. Use this page to specify what user actions should give X Window System client applications focus or raise these applications to the top.

- Focus Control.

  Selecting the **Click in window for focus** option tells the server to transfer focus from one client window to another only by clicking in the window. This is the normal behavior for Presentation Manager windows and is the default behavior for the server.

  If you select the **Move pointer into client area for focus** option it is not necessary to click on a client window; moving the pointer is sufficient.

- Raise Control

  With Raise Control you can determine whether you can raise the window by clicking on the frame or by clicking either on the frame or the application area. The last behavior is usual for Presentation Manager.

You can also define these user actions manually. For more information about the following parameters refer to *X Window System Server Guide*, Chapter 2.

- explicitfocus

- implicitfocus
- clickclienttoraise
- clickframetoraise

### 4.2.4 Controlling X Client Access to OS/2

In order for an X client application on a remote platform to display its output at the OS/2 X server, the remote host must be authorized at the OS/2 X server. There are three ways to provide this authorization:

1. Edit the file C:\TCPIP\ETC\X0HOSTS directly to add host authorization entries (assuming that you have installed the X Window System using the default path). An example of an entry for a AIX host called RS60007 is illustrated in Figure 75.

```
RS60007
```

*Figure 75. An Example Entry in C:\TCPIP\ETC\X0HOSTS*

There needs to be an entry for the host name in the file C:\TCPIP\ETC\HOSTS, in order to resolve a host name, such as RS60007, into a valid IP address. An example entry for RS60007 in C:\TCPIP\ETC\HOSTS is illustrated in Figure 76.

```
9.67.38.75 RS60007
```

*Figure 76. An Example Entry in C:\TCPIP\ETC\HOSTS*

2. Use the configuration notebook utility or PMX configuration option to add entries to C:\TCPIP\ETC\X0HOSTS.

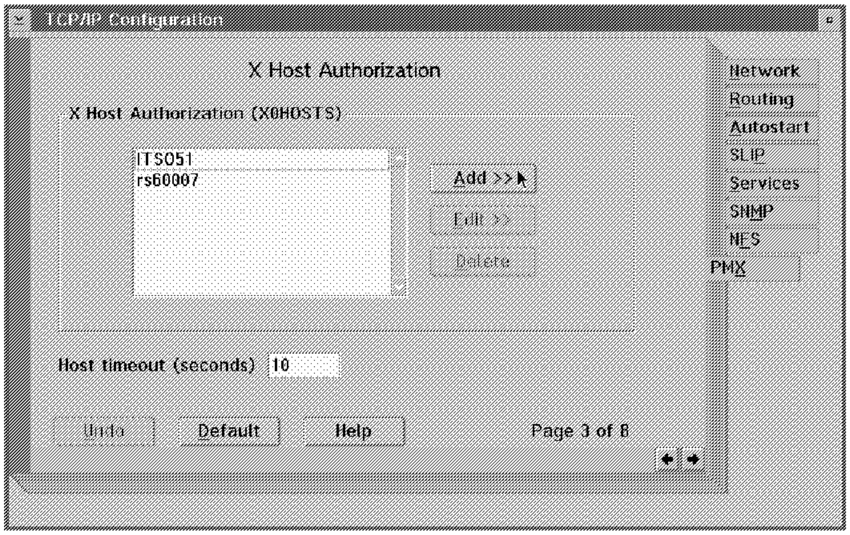


*Figure 77. X Host Authorization*

On the X Hosts Authorization window, shown in Figure 77, you can define which TCP/IP hosts are allowed to use the X Window System server to display applications. The names are stored in the X0hosts file. Use only nicknames, which can be resolved either through a Domain Name Server or the local etc\hosts file, no IP addresses.

3. Use the provided X utility XHOST. Examples of using XHOST follow:

   a. To allow all X client platforms to access the X server enter:

      ```
      XHOST +
      ```

      This turns the access control off.

   b. To only allow those X client platforms listed in C:\TCPIP\ETC\X0HOSTS to access the X server enter:

      ```
      XHOST -
      ```

   c. To authorize a particular host to access the X server enter:

      ```
      XHOST + hostname
      ```

      where hostname corresponds to an entry in C:\TCPIP\ETC\HOSTS and can be resolved into a valid IP address.

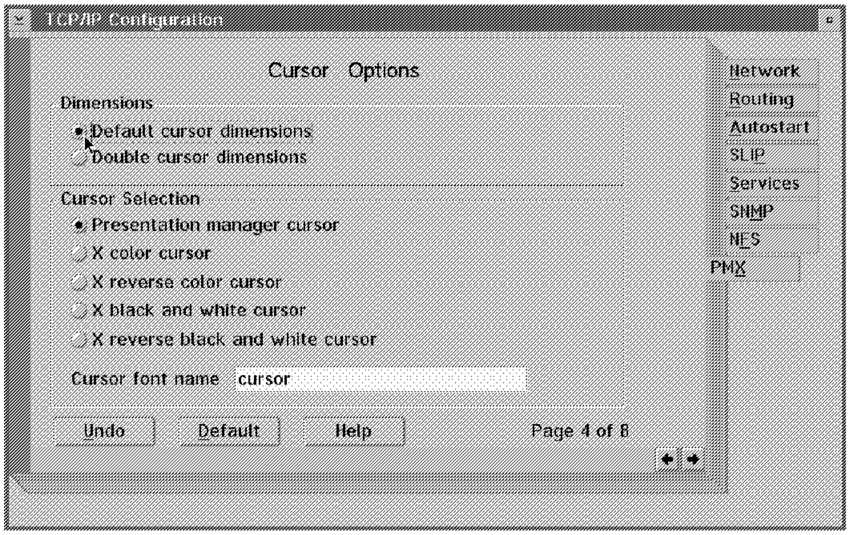
   d. To refuse access to a particular host enter:

      ```
      XHOST - hostname
      ```

      where hostname corresponds to an entry in C:\TCPIP\ETC\HOSTS and can be resolved into a valid IP address.

Note that XHOST authorization is only valid while PMX is active. XHOST provides for dynamically changing authorization while PMX is active. All XHOST authorization is lost when PMX is stopped and restarted. Authorization provided by C:\TCPIP\ETC\X0HOSTS, however, is permanent.

## 4.2.5 Cursor Options


TCP/IP Configuration

Cursor Options

Dimensions
[● Default cursor dimensions]
○ Double cursor dimensions

Cursor Selection
[● Presentation manager cursor]
○ X color cursor
○ X reverse color cursor
○ X black and white cursor
○ X reverse black and white cursor

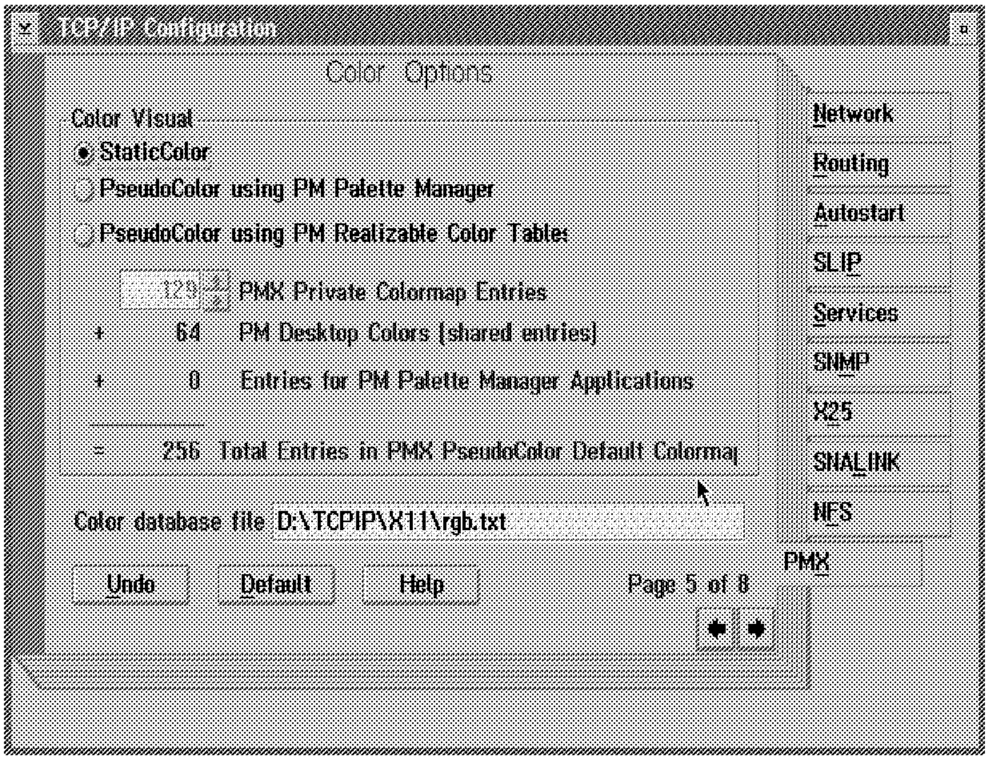
Cursor font name: cursor

*Figure 78. Cursor Options*

In the Cursor Options window shown in Figure 78 you define the setting for your cursor. You can change size or color. The Presentation Manager cursor is limited to default size. For information on defining the cursor manually, see the following parameters in Chapter 2 of the *X Window System Server Guide*:

- lc
- pmcursor
- colorcursor
- bwcursor
- reversecolorcursor
- reversebwcursor
- fc

## 4.2.6 Customizing OS/2 Colors



Color Visual
[✓ StaticColor]
PseudoColor using PM Palette Manager
PseudoColor using PM Realizable Color Table:

129 PMX Private Colormap Entries
+ 64 PM Desktop Colors (shared entries)
+ 0 Entries for PM Palette Manager Applications
= 256 Total Entries in PMX PseudoColor Default Colorma

Color database file D:\TCPIP\X11\rgb.txt

Page 5 of 8

*Figure 79. Color Options*

Figure 79 shows the Color page. Use this page to define how PMX manages colors in X Window System client applications. You can choose what kind of colors are used (static, pseudo or palette manager).

PMX supports the following visual classes:

**StaticColor** The content of the StaticColor table cannot be changed. There is only one such table available, and it contains the colors that are contained in the default physical color table of the display. If the screen is VGA it has 16 colors. If the screen is 8514/A, XGA, SVGA or Image Adapter it has 256 colors. If the X client application asks for a color that is not defined in the colormap, PMX will substitute the closest matching color. The physical color table is unchanged.

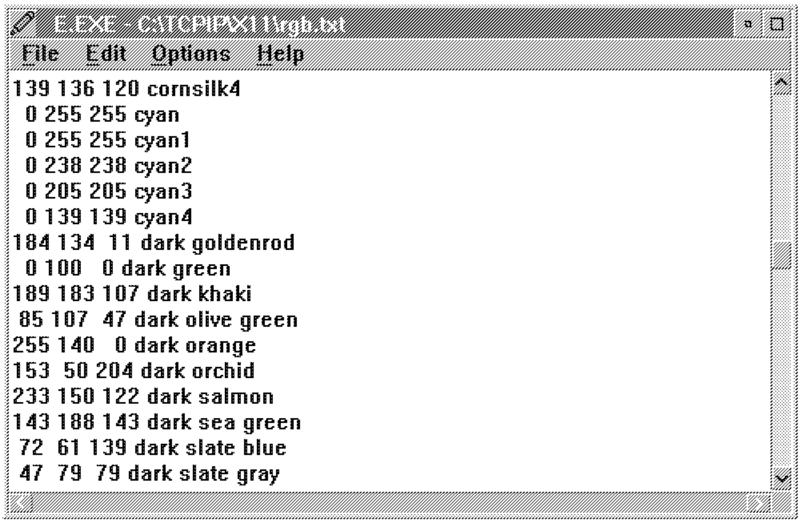
**PseudoColor** This type is supported for the 8514/A, XGA, SVGA and Image Adapter. For this type of colormap, the X client can allocate colors in the shared default PseudoColor table until it has no more room for colors, or the X client can create new private PseudoColor colormaps whose contents the application may control.

**Note:** At the time of writing, the latest PMX CSD (UN60006) implements Pseudocolor in two different ways. The first, and original, way uses PM Realizable Color tables. The second, and new, way uses the PM Palette Manager. By using the PM Palette Manager, PMX causes much less color corruption to other PM application windows. Please see the online documentation and the help items on the Color Options notebook page for more details on choosing between these two PseudoColor implementations.

It is unusual for Presentation Manager applications to modify physical colors on the screen. It is much more common for X applications to do so. PMX attempts to strike a balance between these two contrary behaviors. Some X applications attempt to allocate private colorcells or create private PseudoColor colormaps tables, without checking whether this is supported. As a result, a variety of error messages are generated from the application. If you encounter this situation and your machine has a display adapter for which PM supports Realizable Color Tables or Palette Manager, you may be able to run the application after starting PMX with the -pseudocolor or -palettemgr command line option. For more information on the colormap support see the help function in the configuration panel. For information on defining colormap behavior from the command line, see the following parameters in Chapter 2 of the *X Window System Server Guide*:

- staticcolor
- pseudocolor
- palettemgr
- co

The color name database under OS/2 is found in the file C:\TCPIP\X11\RGB.TXT (assuming that you have installed X Windows using the default path). The format of this file is illustrated by the example entries in Figure 80 on page 136.



```
139 136 120 cornsilk4
  0 255 255 cyan
  0 255 255 cyan1
  0 238 238 cyan2
  0 205 205 cyan3
  0 139 139 cyan4
184 134  11 dark goldenrod
  0 100   0 dark green
189 183 107 dark khaki
 85 107  47 dark olive green
255 140   0 dark orange
153  50 204 dark orchid
233 150 122 dark salmon
143 188 143 dark sea green
 72  61 139 dark slate blue
 47  79  79 dark slate gray
```

*Figure 80. Example of the Contents of C:\TCPIP\X11\RGB.TXT*

Each entry represents a red-green-blue (RGB) value that is associated with a color name. This file is read into memory when the OS/2 window manager PMX is started and is kept sorted by color name. Based on that information PMX may add PseudoColor as a second type of visual that is available.

The RGB.TXT file does not represent the color table. The RGB.TXT file is the color database. In that file you will find color names and the associated RGB values. When an application requests the X server to load a cell of the color table with a new color, the color name is translated to the appropriate RGB value stored in the RGB.TXT file.

There is a difference between the color database in OS/2 and in AIX. Within an AIX environment the RGB.TXT file is used as input for a small database. There is no need to recompile the color database in OS/2.

It is possible to customize the color map by editing RGB.TXT and either changing RGB values associated with a particular color name or adding new color names and RGB values to create new color entries. The following are examples that can be used as a guide to changing or adding to the PMX colormap.

#### 4.2.6.1 Changing a Color

In the following scenario we set the colors for the X client application in an application resource file at the VM host. We then modified the RBG.TXT colormap at the OS/2 X server to change the colors at our workstation.

1. We coded the VM application resource file X DISPLAY as illustrated in Figure 30 on page 62 so that we knew exactly which colors were being specified from the client.

```
XClock*hands: red
XClock*foreground: white
XClock*background: blue
```

*Figure 81. Example of X Display under VM to Set Colors for XCLOCK*

2. After having identified our OS/2 X server as the target display, we started XCLOCK from the CMS prompt. An X window appeared at the OS/2 server displaying a clock with a blue background, white second marks, and red hands. Remember that these colors are specified by the X client.

3. As a user at an OS/2 X server, you have the ability to change the colors that are specified by the X client. This is done by editing the RBG.TXT file and changing the RGB value associated with the color name specified by the client application.

   We changed our red, white, and blue clock into a green and gold clock by making the changes as illustrated in Figure 82.

RGB values for red, blue and white clock:

```
0 0 255 blue
255 0 0 red
255 255 255 white
```

RGB values for green and gold clock:

```
0 255 0 blue
255 215 0 red
255 215 0 white
```

*Figure 82. Changing RGB.TXT*

   Note that the color name blue now has the same RGB value as green, and red and white have the same color value as gold. So the names blue and red are only keywords; they themselves don′t define the color. The colors displayed on the window depends from the RGB values.

4. These changes were implemented by the following steps:

   a. Closing down the XCLOCK window.
   b. Editing RGB.TXT to make the changes as illustrated in Figure 82.
   c. Stopping PMX.
   d. Starting PMX which forces a load of the new colormap with our changes for red, white and blue.
   e. Starting XCLOCK from the CMS command line.

5. An X window then appeared at the OS/2 server displaying a clock with a green face on a gold background.

### 4.2.6.2 Adding a New Color

There may be situations where an X client application specifies a color name that is not in the OS/2 colormap. When this happens it is possible to add a color name with associated RGB values so that the user is able to have the desired color displayed at the OS/2 workstation.

For example, consider the MVS application resource file *userid*.X.DEFAULTS in Figure 83 that specifies the colors deepsea and sunset for the X client application XLOGO.

```
XLogo*foreground: deepsea
XLogo*background: sunset
```

*Figure 83. Example of userid.X.DEFAULTS under MVS to Set Colors for XLOGO*

If you look in RGB.TXT on your OS/2 machine you will not find RGB values specified for deepsea and sunset. Let′s assume that we want to see blue at the OS/2 workstation whenever deepsea is specified and red for sunset. Use the following steps as a guide to adding new colors to RGB.TXT:

1. After having identified our OS/2 X server as the target display, we started XLOGO from the TSO prompt. An X window appeared at the OS/2 server displaying a default black X with a white background. This is because the colors deepsea and sunset could not be found in RGB.TXT.
2. We edited RGB.TXT to add two new entries as illustrated in Figure 84.

```
0 0 255 deepsea
255 0 0 sunset
```

*Figure 84. Adding New Colors to RGB.TXT*

3. We stopped and then restarted PMX to force a load of the new colormap with our new colors deepsea and sunset.
4. We then invoked XLOGO again, which opened a window at the X server which displayed a blue (deepsea) X on a red (sunset) background.

In the directory D:\tcpip\X11 you can find alternative RGB databases:

- OLD-RGB.TXT

  The version that was shipped in previous releases; this was originally ″tuned″ for the Digital VT240 series terminals.

- RAVELING.TXT

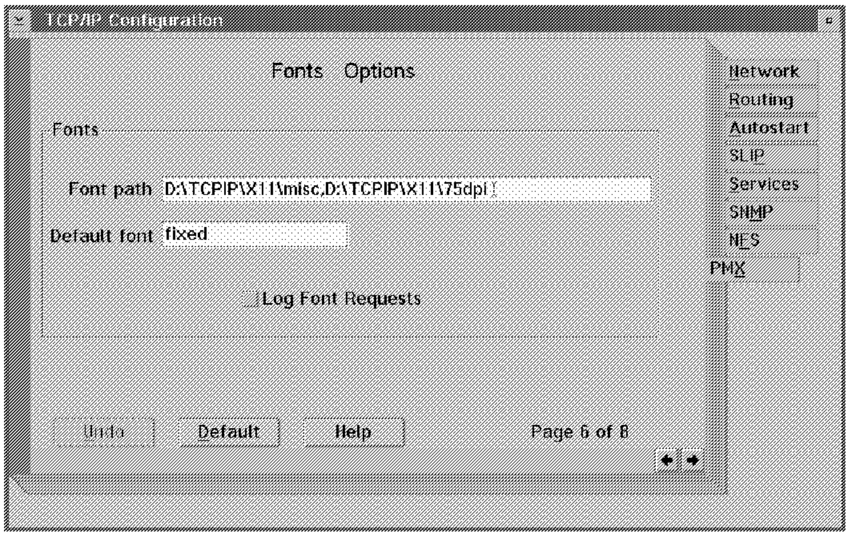
  Lots of new colors, tuned by Paul Raveling at ISI for the HP monitor; see below.

- thomas.txt

  A version of the older database that was tuned by John Thomas at Tektronix to match a box of Crayola crayons; see below.

The authors of the color databases made some comments about these colors: read the readme.rgbin for more detailed descriptions of the colors.

### 4.2.7 OS/2 X Fonts



TCP/IP Configuration

Fonts Options

Fonts

Font path D:\TCPIP\X11\misc,D:\TCPIP\X11\75dpi

Default font fixed

Log Font Requests

*Figure 85. Fonts Option*

Figure 85 shows the Fonts Options window. Use this window to specify the directory containing the fonts you want PMX to use in X Window System client applications. PMX searches for the FONTS.DIR and FONTS.ALI in each directory in the font path. These files map font names to the font names in that directory. Use the LOG fonts option to trace the font request made by X applications.

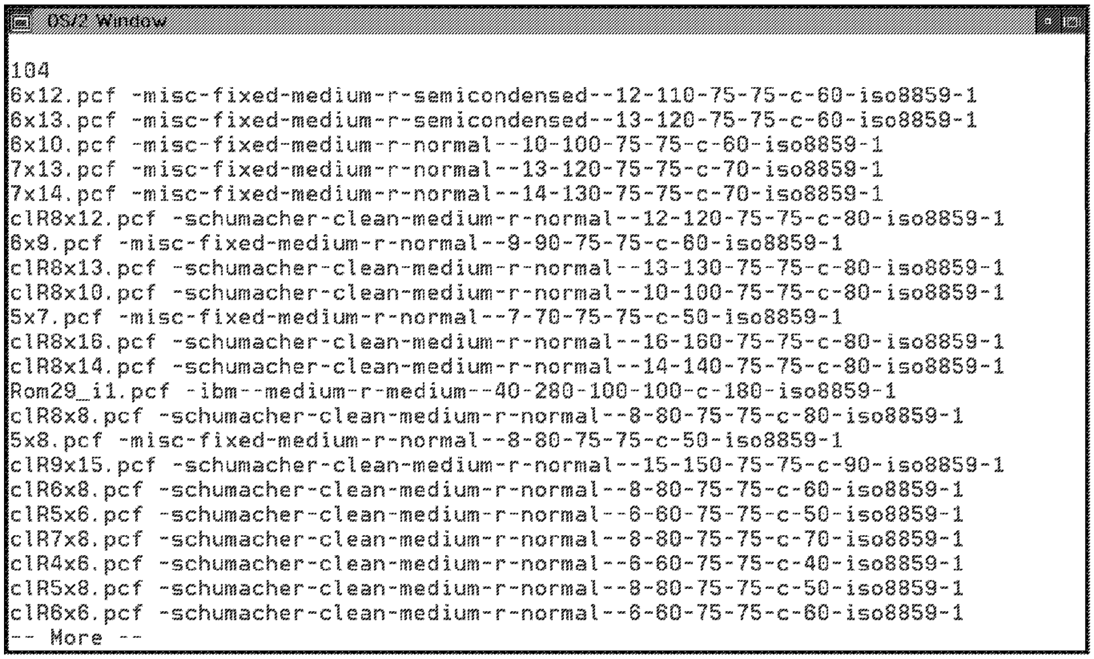
To define fonts manually use the following parameters, which are documented in Chapter 2 of the *X Window System Server Guide*.

- fp
- fn

The OS/2 X Window System is supplied with numerous X fonts, including fonts supplied from the MIT X11R5 X Window System and those from the IBM AIXwindows Environment/6000 product.

Each font is contained in a file with the extension PCF. They are located by default in the subdirectories C:\TCPIP\X11\MISC and C:\TCPIP\X11\75DPI (assuming that you have installed the X Window System using the default path). In our test environment they are in the directory D:\TCPIP\X11\MISC and D:\TCPIP\75DPI. You can change these subdirectories or add additional font libraries using the `-fp` option when invoking PMX.

An X client application specifies one or more fonts when sending a request to PMX to display output. This request may come in the form of a full font name or an accepted alias. For example, an X client application may request the font `-misc-fixed-medium-r-semicondensed--12-110-75-75-c-60-iso8859-1`. PMX accepts this X font name and if possible resolves it to the name of one of the font files. This is done through using a file called FONTS.DIR which contains the PMX font file name and the full font name. There must be a FONTS.DIR file in each subdirectory that PMX uses as a fonts library. An example of some of the entries in FONTS.DIR is illustrated in Figure 86 on page 140.



```
104
6x12.pcf -misc-fixed-medium-r-semicondensed--12-110-75-75-c-60-iso8859-1
6x13.pcf -misc-fixed-medium-r-semicondensed--13-120-75-75-c-60-iso8859-1
6x10.pcf -misc-fixed-medium-r-normal--10-100-75-75-c-60-iso8859-1
7x13.pcf -misc-fixed-medium-r-normal--13-120-75-75-c-70-iso8859-1
7x14.pcf -misc-fixed-medium-r-normal--14-130-75-75-c-70-iso8859-1
clR8x12.pcf -schumacher-clean-medium-r-normal--12-120-75-75-c-80-iso8859-1
6x9.pcf -misc-fixed-medium-r-normal--9-90-75-75-c-60-iso8859-1
clR8x13.pcf -schumacher-clean-medium-r-normal--13-130-75-75-c-80-iso8859-1
clR8x10.pcf -schumacher-clean-medium-r-normal--10-100-75-75-c-80-iso8859-1
5x7.pcf -misc-fixed-medium-r-normal--7-70-75-75-c-50-iso8859-1
clR8x16.pcf -schumacher-clean-medium-r-normal--16-160-75-75-c-80-iso8859-1
clR8x14.pcf -schumacher-clean-medium-r-normal--14-140-75-75-c-80-iso8859-1
Rom29_i1.pcf -ibm--medium-r-medium--40-280-100-100-c-180-iso8859-1
clR8x8.pcf -schumacher-clean-medium-r-normal--8-80-75-75-c-80-iso8859-1
5x8.pcf -misc-fixed-medium-r-normal--8-80-75-75-c-50-iso8859-1
clR9x15.pcf -schumacher-clean-medium-r-normal--15-150-75-75-c-90-iso8859-1
clR6x8.pcf -schumacher-clean-medium-r-normal--8-80-75-75-c-60-iso8859-1
clR5x6.pcf -schumacher-clean-medium-r-normal--6-60-75-75-c-50-iso8859-1
clR7x8.pcf -schumacher-clean-medium-r-normal--8-80-75-75-c-70-iso8859-1
clR4x6.pcf -schumacher-clean-medium-r-normal--6-60-75-75-c-40-iso8859-1
clR5x8.pcf -schumacher-clean-medium-r-normal--8-80-75-75-c-50-iso8859-1
clR6x6.pcf -schumacher-clean-medium-r-normal--6-60-75-75-c-60-iso8859-1
-- More --
```

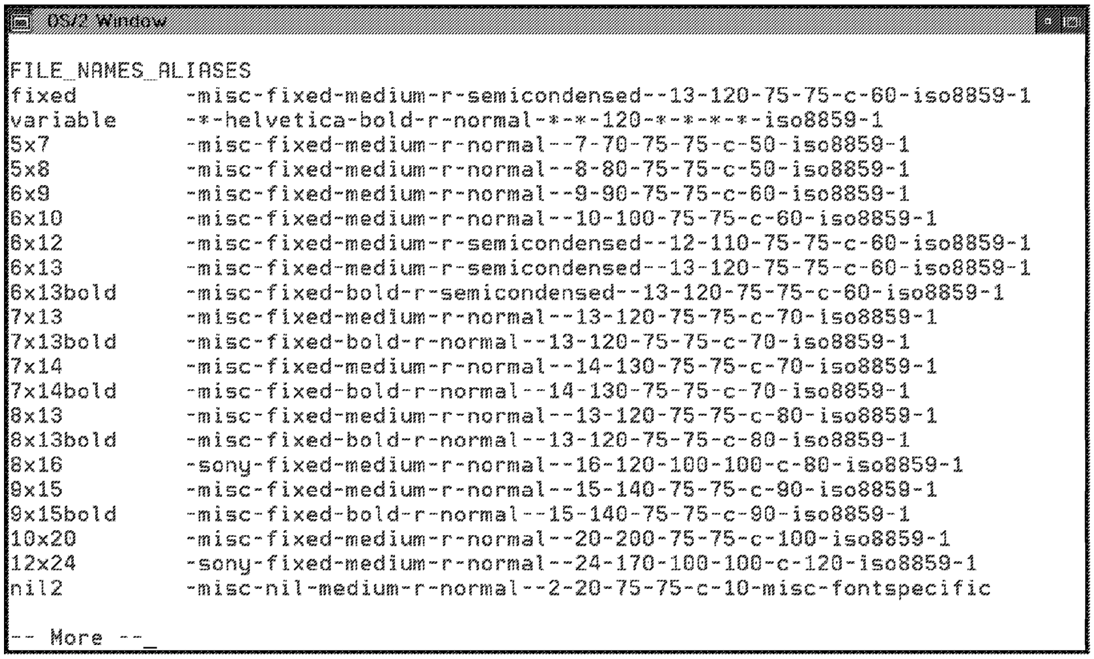
*Figure 86. Example of the Contents of D:\TCPIP\X11\MISC\FONTS.DIR*

Notice that the font `-misc-fixed-medium-r-semicondensed--12-110-75-75-c-60-iso8859-1` maps onto the file 6X12.PCF.

**Note:**

> The SNF fonts used in Version 1.2 are no longer valid. Version 2.0 is using PCF fonts. The background is the change from X11R4 to X11R5: in X11R4 the SNF (Server Normal Fonts) fonts have been used while X11R5 has switched to PCF (Portable Compiled Font). The advantage of PCF fonts compared to SNF fonts is the capability to transfer compiled font files, which eliminates compile action on each X Window server and gives the capability of using a font server.
>
> Sources from fonts are usually in BDF format (Bitmap Distribution Format), if you want to use fonts which are not already on you local X-Window server you can either get those BDF Files via FTP and compile them with the bdftopcf utility or use a font server. Both methods are described in the next chapters.

An X client application may use an alias to request a font. For example, an application may request the font `6x12`. PMX uses the file called FONTS.ALI to map the alias on to a full font name which is then mapped onto a PMX font file name using FONTS.DIR. The file FONTS.ALI needs to be in at least one of the font subdirectories accessed by PMX but it is not necessary to have a FONTS.ALI in each subdirectory. An example of some of the entries in FONTS.ALI is illustrated in Figure 87 on page 141.



```
FILE_NAMES_ALIASES
fixed        -misc-fixed-medium-r-semicondensed--13-120-75-75-c-60-iso8859-1
variable     -*-helvetica-bold-r-normal-*-*-120-*-*-*-*-iso8859-1
5x7          -misc-fixed-medium-r-normal--7-70-75-75-c-50-iso8859-1
5x8          -misc-fixed-medium-r-normal--8-80-75-75-c-50-iso8859-1
6x9          -misc-fixed-medium-r-normal--9-90-75-75-c-60-iso8859-1
6x10         -misc-fixed-medium-r-normal--10-100-75-75-c-60-iso8859-1
6x12         -misc-fixed-medium-r-semicondensed--12-110-75-75-c-60-iso8859-1
6x13         -misc-fixed-medium-r-semicondensed--13-120-75-75-c-60-iso8859-1
6x13bold     -misc-fixed-bold-r-semicondensed--13-120-75-75-c-60-iso8859-1
7x13         -misc-fixed-medium-r-normal--13-120-75-75-c-70-iso8859-1
7x13bold     -misc-fixed-bold-r-normal--13-120-75-75-c-70-iso8859-1
7x14         -misc-fixed-medium-r-normal--14-130-75-75-c-70-iso8859-1
7x14bold     -misc-fixed-bold-r-normal--14-130-75-75-c-70-iso8859-1
8x13         -misc-fixed-medium-r-normal--13-120-75-75-c-80-iso8859-1
8x13bold     -misc-fixed-bold-r-normal--13-120-75-75-c-80-iso8859-1
8x16         -sony-fixed-medium-r-normal--16-120-100-100-c-80-iso8859-1
9x15         -misc-fixed-medium-r-normal--15-140-75-75-c-90-iso8859-1
9x15bold     -misc-fixed-bold-r-normal--15-140-75-75-c-90-iso8859-1
10x20        -misc-fixed-medium-r-normal--20-200-75-75-c-100-iso8859-1
12x24        -sony-fixed-medium-r-normal--24-170-100-100-c-120-iso8859-1
nil2         -misc-nil-medium-r-normal--2-20-75-75-c-10-misc-fontspecific

-- More --_
```

*Figure 87. Example of the Contents of D:\TCPIP\X11\MISC\FONTS.ALI*

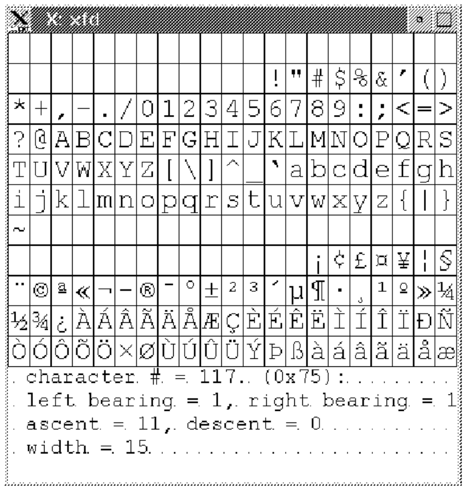
Notice that the alias 6x12 corresponds to the font
-misc-fixed-medium-r-semicondensed--12-110-75-75-c-60-iso8859-1.

It is through using FONTS.DIR and FONTS.ALI that PMX is able to use OS/2 file structures and names for font libraries while still complying with X Window System naming conventions for X fonts.

Each font file (for example, 6X12.PCF) contains object code. PMX provides a utility that allows you to look at the contents of a font file. This utility is called XFD and can be invoked from the OS/2 command prompt as follows:

```
OS2 D:\>xfd -display itso51:0 -bf courr18 -geometry 400x400  -v courr24
```

The XFD utility creates a window in which the characters of a font are displayed. In Figure 88 on page 142 you can see the result of the above command.



```
character # = 117 (0x75):
left bearing = 1, right bearing = 1
ascent = 11, descent = 0
width = 15
```

*Figure 88. Examining courr24.PCF Using PMX Utility XDF*

The XFD utility displays the font named courr24. You can use any font which is in a file available in the XPATH. The first character displayed at the top left is the character number of 0.

All characters in the font might not fit in the window of characters. Clicking mouse button 2 on the window displays the next window of characters. Clicking mouse button 1 on the window displays the previous window of characters.

Clicking buttons 1 and 2 simultaneously (or the middle button if available) on a character displays the character′s number in both decimal and hexadecimal at the bottom of the window. When you select verbose mode (-v option) additional information about particular characters appears. This information includes its width, left bearing, right bearing, ascent and descent.

The option -bf fontname (here courr18) specifies the font for messages at the bottom of the window.

### 4.2.7.1 Adding New Fonts

The OS/2 X Window System provides the ability to add new fonts as required. You would use this facility when the OS/2 X Window System did not support a particular font that was required for a particular application. If this was the case you would need to obtain the source of a font from another X server platform and then compile it for PMX. Usually X font source files have the file extension BDF.

We acquired the source of the X font `-ibm--medium-i-medium--20-14-100-100-m-90-ibm-pc850`, which has the alias `ITL14` and as a source file has the name ITL14.BDF. Although PMX provides this font, we used the source to test out compiling a font under OS/2.

Use the following as a guide to compiling a font for the OS/2 X Window System.

1. Copy the font source file into an OS/2 subdirectory. We created the directory D:\TCPIP\X11\TESTFONT into which we copied ITL14.BDF.
2. Change to the directory D:\TCPIP\X11\TESTFONT.
3. Use the OS/2 X Window system utility BDFTOPCF to compile the font source. Issue the following command from the OS/2 command prompt:

```
BDFTOPCF ITL14.BDF -o ITL14.PCF
```

4. You must now create the file FONTS.DIR to enable PMX to map the requested font name onto the font file. To do this use the PMX MKFONTDR utility as follows:

   ```
   MKFONTDR D:\TCPIP\X11\TESTFONT
   ```

   **Note:** If you have the source BDF file and the PCF file in the same subdirectory the MKFONTDR command fails. Move the source BDF files into another subdirectory.

   This will build a file called FONTS.DIR within the directory specified which is the directory that holds the font that you have just compiled. The entry in FONTS.DIR for ITL14.SNF as created by the MKFONTDR utility is shown in Figure 89. MKFONTDR obtains the information to create the entry in FONTS.DIR from the fonts file ITL14.PCF.

```
ITL14.SNF     itl14
```

*Figure 89. The Entry in FILES.DIR Created by MKFONTDR for ITL14.SNF*

   If we had compiled ITL14.SNF into a directory that already had a FONTS.DIR then MKFONTDR would have simply updated FONTS.DIR by adding an entry for ITL14.SNF.

5. Add the new font to the fontpath dynamically with the command XSET:

   ```
   XSET fp D:\TCPIP\X11\MISC,D:\TCPIP\X11\75DPI,D:\TCPIP\X11\TESTFONT
   ```

6. Update the fontpath in the TCP/IP Configuration Notebook or PMX Configuration utility for permanent update.

**Note:** At the time of writing, the latest PMX CSD (UN60006) contains a handy ALLFONTS.CMD Rexx utility that will compile all the .BDF font files in the current directory into .PCF format and then will run MKFONTDR to update the FONTS.DIR file. Just enter `allfonts` in a directory containing downloaded .BDF files.

#### 4.2.7.2 Using a Font Server

The font server (fs) program acts as an intermediary between your X server and the fonts resident on the system where the font server is running. If you want to use fonts which are not on your system you can either port them (as previously described) or use a font server which provides those fonts. If a font desired by an application is not available on the PMX server you receive the following message:

```
OS2 C:\>xfd -geometry 800x500 -bf courr12 -v xtrek
xfd: error: Unable to open font xtrek!
```

To build up a font server on an AIX system and use it with PMX, follow these steps:

1. Set up the config data set on the AIX system. This is a sample which resides in the subdirectory /usr/lib/X11/fs:

```
# @(#)29      1.3  com/XTOP/fonts/server/config.cpp, xfont,
#             r5gos325, 9325325 5/26/93 11:25:42
#
# COMPONENT_NAME:
#
# FUNCTIONS:
#
# ORIGINS: 16,27
#
# (C) COPYRIGHT International Business Machines Corp. 1992, 1993
# All Rights Reserved
# Licensed Materials - Property of IBM
#
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
#
#
# font server configuration file

clone-self = on
use-syslog = off

catalogue = /usr/lib/X11/fonts/,/usr/lib/X11/fonts/misc/,
/usr/lib/X11/fonts/JP/,/usr/lib/X11/fonts/75dpi/,
/usr/lib/X11/fonts/100dpi/,
/usr/lib/X11/fonts/Type1/,/usr/lib/X11/fonts/Speedo/,
/usr/lib/X11/fonts/i18n/,/usr/lib/X11/fonts/oldx11/,
/usr/lib/X11/fonts/oldx10/,
/usr/lib/X11/fonts/bmug/,/usr/lib/X11/fonts/info-mac/

error-file = /usr/lib/X11/fs/fs-errors

# in decipoints
default-point-size = 120
default-resolutions = 75,75,100,100

# This is normally 7000 but it is not a requirement. It is actually the
# first free port available.
port = 7500
```

*Figure 90. Config Data Set of AIX Font Server*

2. To start the font server issue the command `fs &`

3. Add the font server to your local font path:

   PMX can access one or more font servers. Everywhere a directory can be specified in a font path, you can specify a font server instead. The font server specification has the form `:tcp/hostname:portnumber`. For example, the following command points to an AIX font server:

   ```
   OS2 C:\>xset fp+ tcp/rs60001:7500
   ```

   **Remark!**

   Usually the port number for the font server is documented as 7000; AIX uses 7500!

4. Check whether the font server definitions have been added with the XSET q(uery) command:

```
OS2 C:\>xset q
Keyboard Control:
  auto repeat:  on    key click percent:  0    LED mask:  00000000
  auto repeating keys:  00ffffffffffffff
                        ffffffffffffffff
                        ffffffffffffffff
                        ffffffffffffffff
  bell percent:  50    bell pitch:  400    bell duration:  100
Pointer Control:
  acceleration:  2/1    threshold:  4
Screen Saver:
  prefer blanking:  yes    allow exposures:  yes
  timeout:  600    cycle:  600
Colors:
  default colormap:  0x21    BlackPixel:  0    WhitePixel:  255
Font Path:
  d:/tcpip/x11/misc,d:/tcpip/x11/75dpi,tcp/rs60001:7500
```

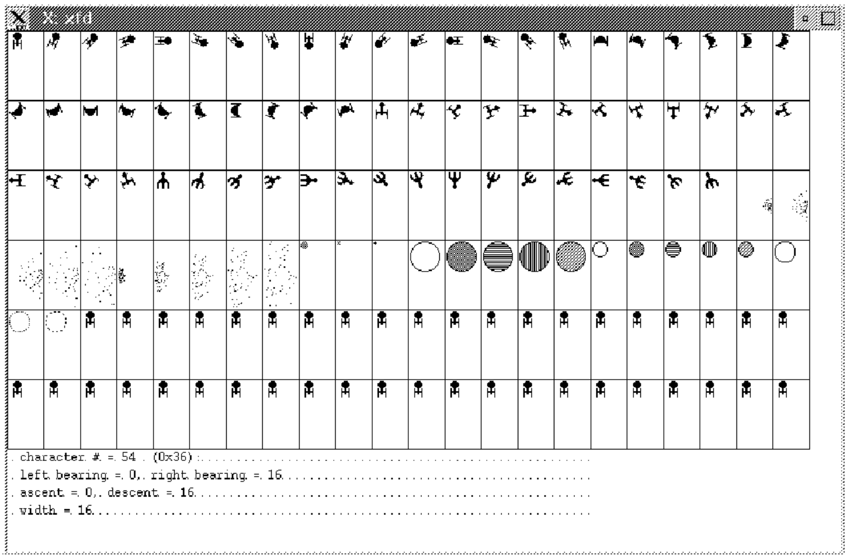
*Figure 91. Xset Command*

Note that the font server specifications have been added to the font path.

5. Restart the application:

   `OS2 C:\>xfd -geometry 800x500 -bf courr12 -v xtrek`

   The result is shown in Figure 92.



```
character # = 54  (0x36)
left bearing = 0, right bearing = 16
ascent = 0, descent = 16
width = 16
```

*Figure 92. XFD Output*

**Note:**

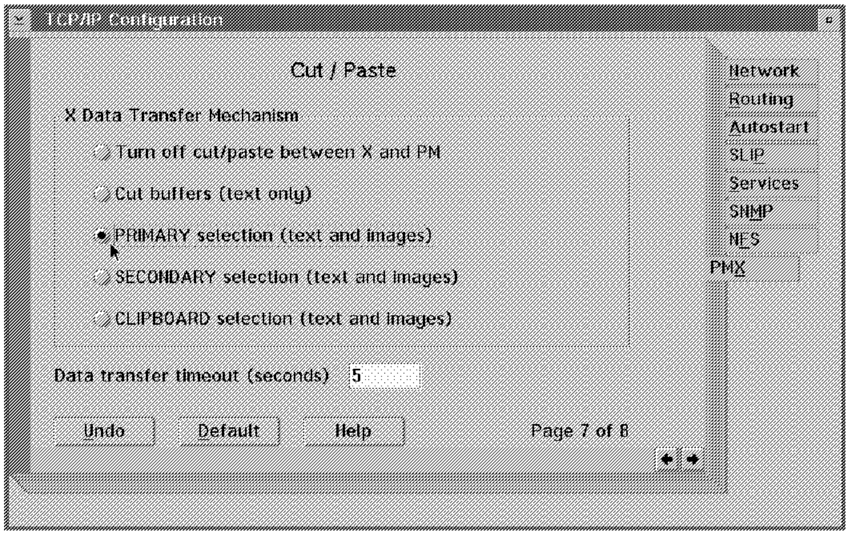
To check if a specific font is available on a X Window server (here PMX); use the xlsfonts utility. If the font is available its name is echoed. You can also use patterns such as courr* to check fontnames.

```
OS2 C:\>xlsfonts -display itso51:0 -fn xtrek
xtrek
```

*Figure 93. Xlsfonts Utility*

## 4.2.8 Using the PM Clipboard with PMX

Figure 94 shows the Cut/Paste page. Use this page to specify how to manage cut and paste operations. Cutting and pasting is a convenient way for users to transfer data between X Window System client applications, Presentation Manager, and Win-OS/2 applications.



Cut / Paste

X Data Transfer Mechanism

- Turn off cut/paste between X and PM
- Cut buffers (text only)
- [✓ PRIMARY selection (text and images)]
- SECONDARY selection (text and images)
- CLIPBOARD selection (text and images)

Data transfer timeout (seconds) 5

*Figure 94. Cut and Paste*

OS/2′s Presentation Manager and the X Window System both provide a way to share data among applications using cutting, copying, and pasting. The two systems handle these operations differently, however. PMX provides facilities for sharing data between applications running under PM and those running as clients of PMX.

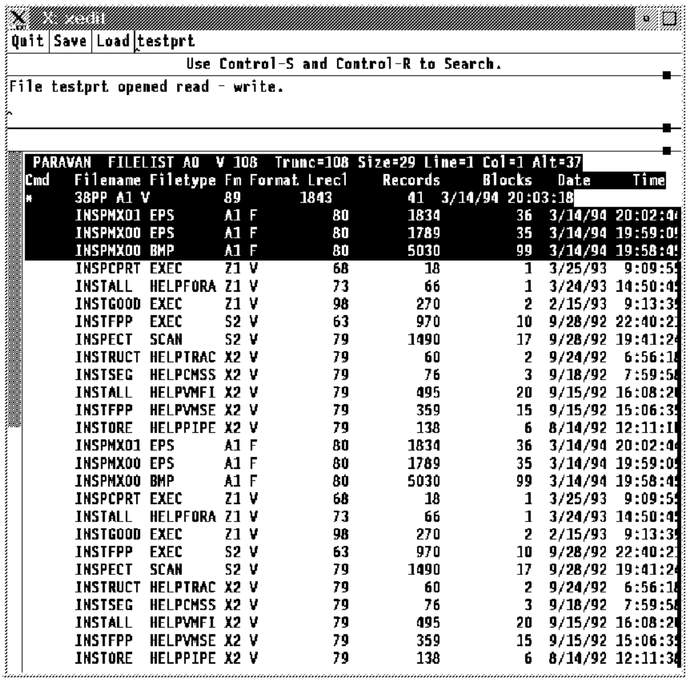
The options are described in detail in the *X Window System Server Guide*. An example of how to use the cut buffer is described here:

### 4.2.8.1 Cut Buffer within PMX

When the user copies or cuts data from an X Window System client application, the application transfers the data to the cut buffer and has no further responsibility for the data. When the user pastes the data into another X Window System client application, that application gets the data from the cut buffer.

PMX, itself, always treats cut and paste data this way. PMX Window System always cuts data from an X Window System client into CUT_BUFFER0. Any subsequent cut operations overlay the previous cut, with the cut data always going into CUT_BUFFER0. PMX Window System always pastes data to an X Window System client from CUT_BUFFER0. Any subsequent paste operations continue to get the same data from CUT_BUFFER0 until the data is changed by a subsequent cut.

To use the cut buffers as a means of cutting and pasting between X and OS/2 applications, select the cut buffers (text only) option from the PMX configuration notebook (Figure 94). Then close the notebook and select **Apply** or **Save and Apply** to apply the setting to PMX. Now start the text-related applications in X, PM, and WIN-OS/2 environments. To see which X Window System clients support cut-and-paste operations using cut buffers, check the manual pages for the client of interest. Two clients that use this method are XTERM and XEDIT. You can simply cut and paste among all three types of applications assuming the clients support it. Using XTERM, for example, you can use mouse button 1 to select the desired text part. See Figure 95 for an example.

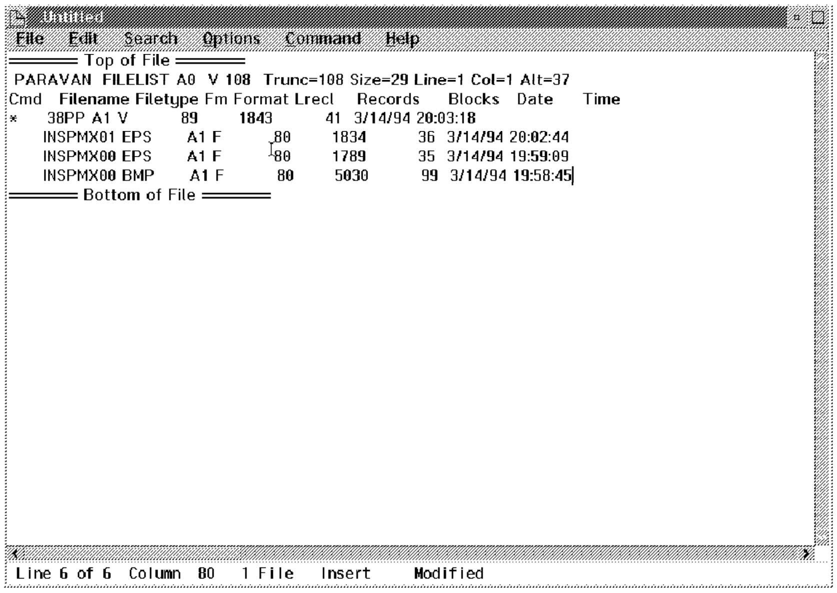


```
testprt
                    Use Control-S and Control-R to Search.
File testprt opened read - write.

 PARAVAN  FILELIST A0  V 108  Trunc=108 Size=29 Line=1 Col=1 Alt=37
Cmd    Filename Filetype Fm Format Lrecl     Records      Blocks   Date      Time
*      38PP A1 V         89        1843            41  3/14/94 20:03:18
       INSPMX01 EPS      A1 F          80       1834          36  3/14/94 20:02:4
       INSPMX00 EPS      A1 F          80       1789          35  3/14/94 19:59:0
       INSPMX00 BMP      A1 F          80       5030          99  3/14/94 19:58:4
       INSPCPRT EXEC     Z1 V          68         18           1  3/25/93  9:09:5
       INSTALL  HELPFORA Z1 V          73         66           1  3/24/93 14:50:4
       INSTGOOD EXEC     Z1 V          98        270           2  2/15/93  9:13:3
       INSTFPP  EXEC     S2 V          63        970          10  9/28/92 22:40:2
       INSPECT  SCAN     S2 V          79       1490          17  9/28/92 19:41:2
       INSTRUCT HELPTRAC X2 V          79         60           2  9/24/92  6:56:1
       INSTSEG  HELPCMSS X2 V          79         76           3  9/18/92  7:59:5
       INSTALL  HELPVMFI X2 V          79        495          20  9/15/92 16:08:2
       INSTFPP  HELPVMSE X2 V          79        359          15  9/15/92 15:06:3
       INSTORE  HELPPIPE X2 V          79        138           6  8/14/92 12:11:1
       INSPMX01 EPS      A1 F          80       1834          36  3/14/94 20:02:4
       INSPMX00 EPS      A1 F          80       1789          35  3/14/94 19:59:0
       INSPMX00 BMP      A1 F          80       5030          99  3/14/94 19:58:4
       INSPCPRT EXEC     Z1 V          68         18           1  3/25/93  9:09:5
       INSTALL  HELPFORA Z1 V          73         66           1  3/24/93 14:50:4
       INSTGOOD EXEC     Z1 V          98        270           2  2/15/93  9:13:3
       INSTFPP  EXEC     S2 V          63        970          10  9/28/92 22:40:2
       INSPECT  SCAN     S2 V          79       1490          17  9/28/92 19:41:2
       INSTRUCT HELPTRAC X2 V          79         60           2  9/24/92  6:56:1
       INSTSEG  HELPCMSS X2 V          79         76           3  9/18/92  7:59:5
       INSTALL  HELPVMFI X2 V          79        495          20  9/15/92 16:08:2
       INSTFPP  HELPVMSE X2 V          79        359          15  9/15/92 15:06:3
       INSTORE  HELPPIPE X2 V          79        138           6  8/14/92 12:11:3
```

*Figure 95. Cut in XEDIT Application*

Next go to any other X, PM, or WIN-OS/2 application (here we used the PM enhanced editor). We used the paste function to paste the contents of CUT_BUFFER0 to the application. The result is shown in Figure 96 on page 148.

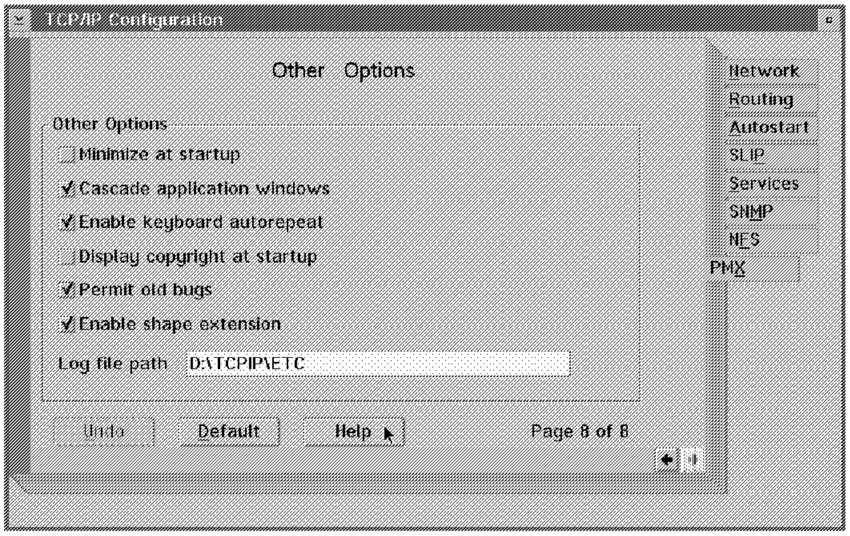


```
═══════ Top of File ═══════
 PARAVAN  FILELIST A0  V 108  Trunc=108 Size=29 Line=1 Col=1 Alt=37
Cmd   Filename Filetype Fm Format Lrecl    Records    Blocks   Date     Time
*    38PP A1 V        89      1843        41  3/14/94 20:03:18
     INSPMX01 EPS      A1 F       80      1834        36  3/14/94 20:02:44
     INSPMX00 EPS      A1 F       80      1789        35  3/14/94 19:59:09
     INSPMX00 BMP      A1 F        80      5030        99  3/14/94 19:58:45
═══════ Bottom of File ═══════

 Line 6 of 6  Column  80   1 File   Insert     Modified
```

*Figure 96. PASTE in PM Enhanced Editor*

## 4.2.9 Other Options



Other Options
- [ ] Minimize at startup
- [✓] Cascade application windows
- [✓] Enable keyboard autorepeat
- [ ] Display copyright at startup
- [✓] Permit old bugs
- [✓] Enable shape extension

Log file path D:\TCPIP\ETC

*Figure 97. Other Options*

Figure 97 shows the Other Options page. Use this page to define mouse options and other miscellaneous options defining the way PMX operates.

If the Cascade application windows option is selected, windows that are not initially positioned by user specifications will be cascaded down the screen.

Use Permit old bugs to permit certain old broken clients to function properly. Examples of clients that will be affected by this setting are X11R2 and R3 versions of XTERM.

The installed shape extension allows non-rectangular windows to be created by clients. This function is needed if you plan to display NV/6000 on the OS/2. For information on defining these options manually, see the following parameters in *X Window System Server Guide*:

- nocopyright
- nocascade
- logpath

## 4.3 DOS

In DOS we used HCL-eXceed/W for Windows and HCL-eXceed/DOS for DOS without Windows.

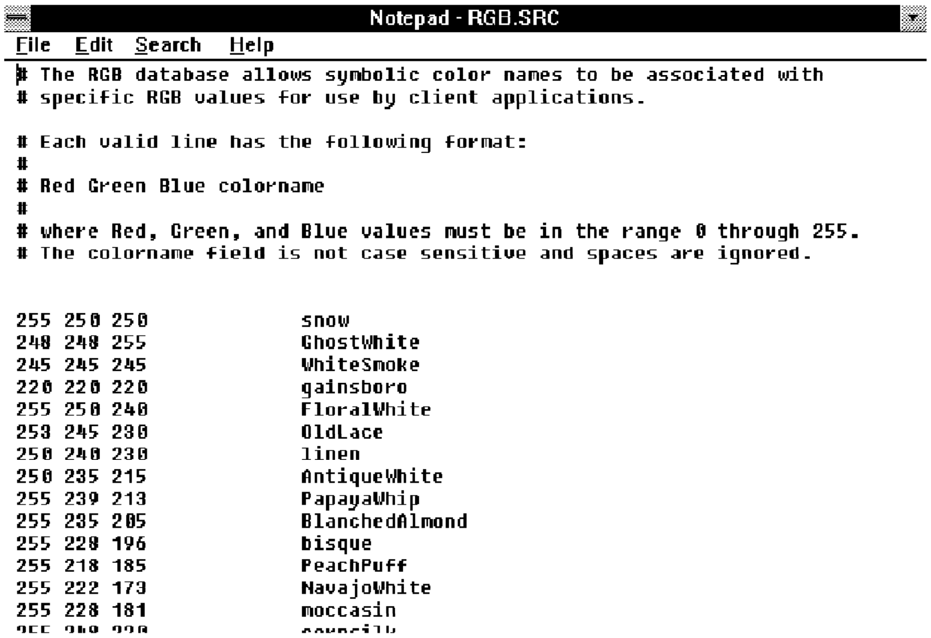
### 4.3.1 Customizing HCL-eXceed/W for Windows Version 3.3.3

The customizing of this X server implementation is very easy. Most of the customization utilities can be started with an icon and there is help text available. The customization programs are stored in the Xconfig/W program group. We assume that you have already installed the program and that an X client can connect to your server.

#### 4.3.1.1 Customizing Colors

The color database information is stored in the drive:\installation\user\rgb.src file. eXceed/W supports the PseudoColor, StaticGray, and StaticColor visuals. You can modify the color database by using a program which is represented as the COLOR icon in the Xconfig/W program group. By using this program you can change and add items to the color database file.

You can change the colors either by changing the RGB values, or you can add a color of your choice by setting the RGB values and adding a color name.



```
# The RGB database allows symbolic color names to be associated with
# specific RGB values for use by client applications.

# Each valid line has the following format:
#
# Red Green Blue colorname
#
# where Red, Green, and Blue values must be in the range 0 through 255.
# The colorname field is not case sensitive and spaces are ignored.


255 250 250             snow
248 248 255             GhostWhite
245 245 245             WhiteSmoke
220 220 220             gainsboro
255 250 240             FloralWhite
253 245 230             OldLace
250 240 230             linen
250 235 215             AntiqueWhite
255 239 213             PapayaWhip
255 235 205             BlanchedAlmond
255 228 196             bisque
255 218 185             PeachPuff
255 222 173             NavajoWhite
255 228 181             moccasin
```

Figure 98. The RGB Database

The edit menu is shown in Figure 98.

When you have changed the color database, press the compile button to rebuild. When you quit the customization save your changes. The SAVE item appears when you press the left button on the Xconfig/W menu bar. You can activate your changes without restarting your X server. This can be done by the following:

- Click on your started HCL-eXceed/W icon.
- Select the following items:

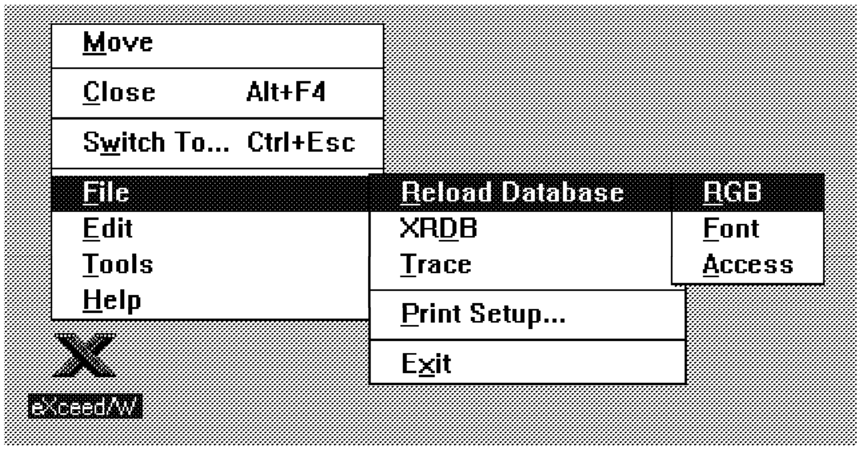
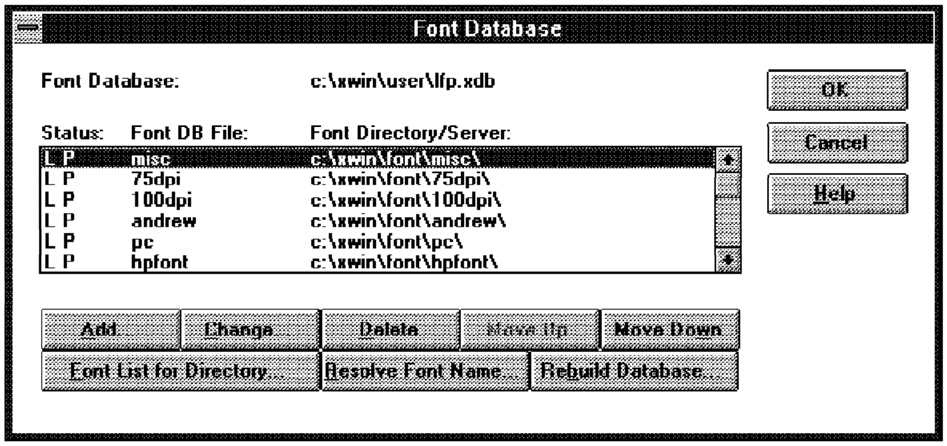


*Figure 99. Reload the RGB Database*

### 4.3.1.2 X Fonts

The customizing of the font database differs from other X server implementations. The information for the fonts (fonts.dir and fonts.ali paths) is stored in one database. You can access the eXceed/W fontbase by the FONT icon. You can compile fonts, manage aliases and customize the font directories. This integrated program is a replacement for the clients mkfontdir, bdftopcf, xdf, and xset -fp.

There is a log available that shows you useful information about the clients′ requests. When a client requests a font either with the alias name or with the XLFD structure, and the server doesn′t find an entry in its font database, an entry with the error message will be added in the log file. This is an easy way to see which alias you should add to your font database. An alias gives you the ability to set several logical names for one physical font.

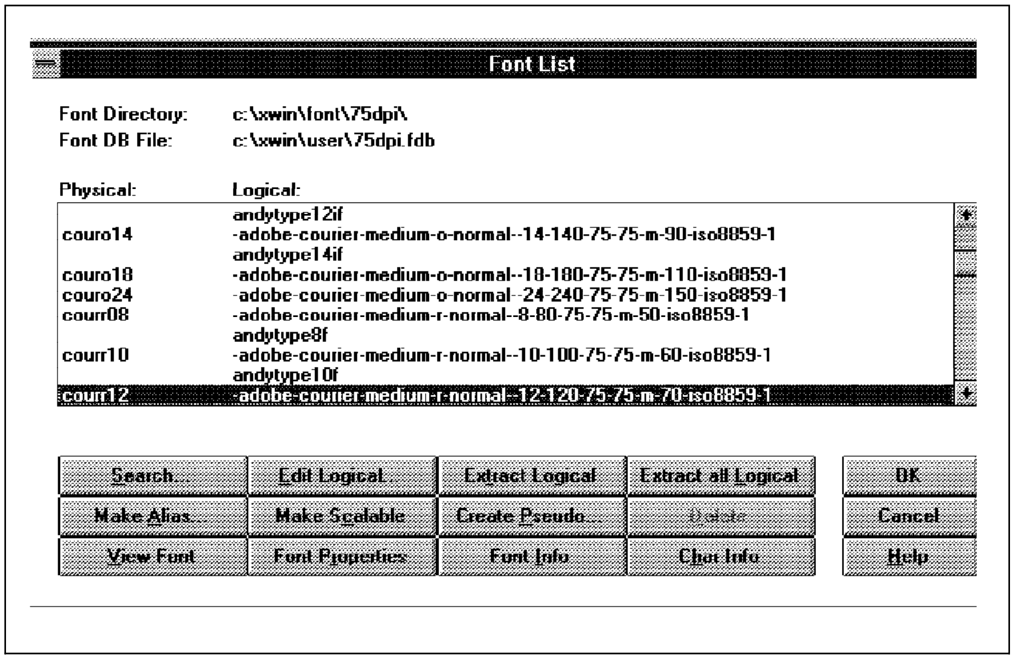


```
Font Database:          c:\xwin\user\lfp.xdb

Status:  Font DB File:  Font Directory/Server:
L P      misc           c:\xwin\font\misc\
L P      75dpi          c:\xwin\font\75dpi\
L P      100dpi         c:\xwin\font\100dpi\
L P      andrew         c:\xwin\font\andrew\
L P      pc             c:\xwin\font\pc\
L P      hpfont         c:\xwin\font\hpfont\
```

Figure 100. HCL-eXceed/W Font Database

On that window (see Figure 100), you can also call functions to display specific fonts. After you call Resolve Font Name... and use the string courr12 as a search pattern, the window in Figure 101 appears and lists the fonts.



```
Font Directory:   c:\xwin\font\75dpi\
Font DB File:     c:\xwin\user\75dpi.fdb

Physical:         Logical:
                  andytype12if
couro14           -adobe-courier-medium-o-normal--14-140-75-75-m-90-iso8859-1
                  andytype14if
couro18           -adobe-courier-medium-o-normal--18-180-75-75-m-110-iso8859-1
couro24           -adobe-courier-medium-o-normal--24-240-75-75-m-150-iso8859-1
courr08           -adobe-courier-medium-r-normal--8-80-75-75-m-50-iso8859-1
                  andytype8f
courr10           -adobe-courier-medium-r-normal--10-100-75-75-m-60-iso8859-1
                  andytype10f
courr12           -adobe-courier-medium-r-normal--12-120-75-75-m-70-iso8859-1
```

Figure 101. HCL-eXceed/W Font List

To reload the font database without restarting the X server, click on the started HCL-eXceed/W server icon and reload the database with the following menu:

```
File →Reload → Database → Font
```

### 4.3.1.3 Remapping the Keyboard

There is no xmodmap client available. However, you can change the layout of your keyboard. There are some country-specific keyboards available that you can select, and you can change the layout of your keymap with an integrated tool that is similar to the xmodmap command. But it is not possible to have more than one keyboard mapping on your display at the same time, because the X server reads the keyboard you defined at startup time.

When you work with aixterm you will see very soon that the keyboard mapping does not correspond to the IBM PS/2 keyboard layout. When working with an aixterm the Backspace key works only by pressing the Shift modifier, and it is not possible to print any decimal number through the numeric keypad.

We changed the keyboard in the following way:

1. Create a copy of the keyboard file in the drive:\installation\user directory. We named it test.kbd.
2. Start Windows with the command:

   ```
   win xport
   ```

3. Activate the INPUT icon in the Xconfig/W group and select your copy of the keyboard file. Press the Edit button. The editor starts and you see the configuration file of your keyboard.
4. We made the changes as illustrated in Figure 102.

```
compose NumLock+91 255.183 .... to  compose Numlock+91 55 .....
compose NumLock+92 255.180 .... to  compose NumLock+92 52 .....
compose NumLock+93 255.177 .... to  compose NumLock+93 49 .....
compose NumLock+96 255.184 .... to  compose NumLock+96 56 .....
compose NumLock+97 255.181 .... to  compose NumLock+97 53 .....
compose NumLock+98 255.178 .... to  compose NumLock+98 50 .....
compose NumLock+99 255.176 .... to  compose NumLock+99 48 .....
compose NumLock+101 255.185 ....to  compose NumLock+101 57 .....
compose NumLock+102 255.182 ....to  compose NumLock+102 54 .....
compose NumLock+103 255.179 ....to  compose NumLock+103 51 .....
compose NumLock+104 255.174 ....to  compose NumLock+104 46 .....
15 255.255 255.8                to  15 255.2 255.255
```

*Figure 102. Changing the Keyboard Mapping for HCL-eXceed/W*

5. Save your changes with the editor title bar.
6. Compile your keyboard database.
7. Save your settings with the Xconfig/W title bar.
8. Stop your X server.
9. Restart your X server.

To modify or create a keyboard file, you will also require some or all of the following information:

- The number of the PC key you want to define. You can find the number associated with any physical key on your keyboard by running ShowKey from the eXceed/W Program Group. You will be prompted to press any key.

- The number of the X keysym you want to keystroke to send. A keysym is a key code that corresponds to a specific symbol supported by the X protocol.

  Keysyms and their corresponding codes are listed in the file KEYSYMS.SYM located in your \home\INFO directory.

#### 4.3.1.4 Controlling X Client Access

When you start the X server after installation, every host is permitted to display windows on your screen. To control the access of the hosts you can edit an access file. Activate the ACCESS icon and edit the file. If there is no entry in the file, every host is allowed to display a window on your server. Add the hosts you want to allow access to (one host on each line), then save and compile the file.

The eXceed/W icon allows you to dynamically reread the RGB, font, and access databases. The changes will take effect at once and you don′t have to restart the X server. Only when you change the keyboard do you have to restart the server.

## 4.3.2 Customizing HCL-eXceed/DOS for DOS

The customization of HCL-eXceed Plus is very easy and the configuration tool provides help text. The configuration program is called xconfigp and it can be invoked when the X server is running by pressing the Alt-Esc and then the Enter keys. This key combination starts a DOS shell. To quit the DOS shell enter `exit` at the command prompt.

#### 4.3.2.1 Customizing Colors

The RGB database file is called drive:\install\source\rgb.src. It exists in a source format and in a compiled version. The compiled version is the file drive:\install\work\rgb.xdb. HCL-eXceed Plus uses the compiled version.

HCL-eXceed Plus supports three visual classes: PseudoColor, GrayScale, and StaticColor. To change the color definition you can perform the following steps:

1. Start with the xconfigp command.
2. Press F6 (RGB database).
3. The next screen is the definition file for the color database. You can edit this either by changing an existing entry or by adding or deleting a line. The syntax is as follows:

   ```
   Red Green Blue [=colornumber] colorname
   ```

   Red, Green, Blue must be a value between 0 and 255. The colornumber option is used when the visual class is set to StaticColor. The colors with this colornumber entry will be loaded into the colormap. colorname is the name of the color which is used to indicate the color for an X client.
4. Press F10 to compile the database (Compile/Exit).
5. Press F10 to quit the customize menu (Compile/Save).

**Note:** To reload the RGB database while the server is running, press Alt-Esc and then the letter R. See Table 7 on page 52.

### 4.3.2.2 X Fonts

The font customization can be started with the xconfigp command. Let′s have a look at the font database:

1. Start the xconfigp program.
2. Select the font database with F8. Perform one of the following activities:
   a. Add and delete font paths.
   b. Search for specific fonts and display or edit them.
   c. Change the search order through the fontpaths by changing the in order.

   To display or change a font or its characteristics press F7 (or Alt + F7) and enter a search pattern (Font name or Alias). A list of all fonts, which include this search pattern is listed. Now you can view the font or change font definitions.
3. By pressing F10 you will see the main menu and when you press the F10 (Exit/Save), you quit the customization.

   **Note:** To reload the font database while the server is running, press Alt-Esc and then the letter F.

### 4.3.2.3 Remapping the Keyboard

HCL-eXceed Plus offers several predefined keyboard files. They are stored in the directory drive:\install\source directory. To select a new keyboard file or change an existing one, execute the following:

1. Start the xconfigp command at the command prompt. If you have already started HCL-eXceed Plus, enter Alt-Esc, and then Esc again. This will end your HCL-eXceed Plus session because you have to restart your X server when you have made keyboard mapping changes.
2. Select INPUT SETTINGS with F2.
3. Choose a keyboard on the line KEYBOARD CLASS (Cursor left).
4. Press F2 (Compile/Edit).

   You can make the same changes as described in the customization of HCL-eXceed/W for Windows.
5. Press F10 (Compile/Exit).
6. Press Esc (Main Menu) and then F10 (Exit/Save).
7. To activate the changes you must restart the server with the exceedp command.

### 4.3.2.4 Controlling X Client Access

The access file is called drive:\install\source\access.src. It exists in the source format and in a compiled version. The compiled version is the file drive:\install\work\access.xdb. HCL-eXceed Plus uses the compiled version. To set the access permissions perform the following steps:

1. Start the xconfigp program at the DOS command prompt.
2. Press F7 (Access Database).
3. Edit the file. If empty all hosts are allowed to connect to your server.
4. Press F10 (Compile/Exit).
5. Press F10 (Exit/Save).

**Note:** To reload the access database while the server is running, press Alt-Esc and then the letter H.

# Chapter 5. Multivendor Interoperability

One of the key advantages of the X Window System is that it is device and vendor independent. X client applications can run on one platform and use a remote system as the X server to display output. As long as that remote system complies with the X Window System standards, it should not matter what the hardware and operating system is on the X server.

We had an HP 9000 Model 340 running HP-UX** Release 6.5 available at the ITSO Center in Raleigh with which we were able to demonstrate this fundamental principle. Please refer to the network configuration in Figure 7 on page 25.

The following section describes X Window System interoperability between IBM systems and the Hewlett-Packard** platform.

## 5.1 Hewlett-Packard

The Hewlett-Packard workstation with which we tested X Window System interoperability was an HP 9000 Model 340 running HP-UX Release 6.5, which is the Hewlett-Packard implementation of UNIX. The Hewlett-Packard X Window System was at Version A.01.

The X Window System was installed as a component of the HP-UX operating system.

### 5.1.1 Hewlett-Packard as a Client

We performed the following steps to start an X application on the HP 9000 and use an OS/2 X server as the target display:

1. We authorized the HP 9000 at the OS/2 X server by doing the following:
   a. We added an entry in C:\TCPIP\ETC\HOSTS for the HP 9000 which has an internet address of 9.67.32.66. We used the host name HPAIX. The entry is illustrated in Figure 103.

```
9.67.32.66 HPAIX
```

*Figure 103. The Entry in C:\TCPIP\ETC\HOSTS for the HP 9000*

   b. We issued the following command to add the HP 9000 to the X client authorization file:

```
xhost + HPAIX
```

2. We started the X Window System on the HP 9000 by issuing the following command at the HP-UX prompt:

```
/usr/bin/x11start
```

   Note that we have included the full directory path for the x11start command. You can find the directory path for a file under HP-UX by issuing the following command:

```
    find / -name filename -print
```

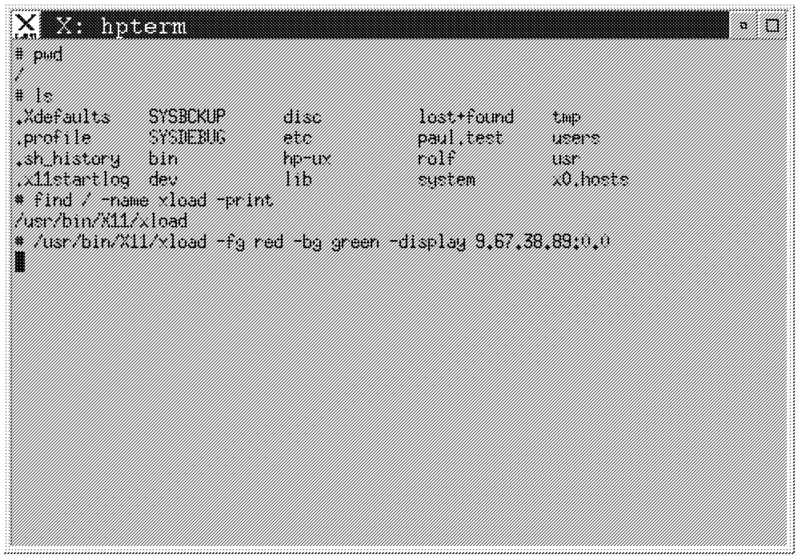
where filename is the file that you wish to locate.

3. We started the X client application hpterm and directed the output to our OS/2 X server with the following command at the HP-UX prompt:

```
    /usr/bin/X11/hpterm -display 9.67.38.89:0.0
```

Note that we have included the full directory path for the hpterm command.

This opened up an HP 9000 command window on the OS/2 X server at which we could enter HP-UX commands as if we were a user at a local HP 9000 screen. An example of the window is illustrated in Figure 104. We did not encounter any significant keyboard mapping problems that prevented effective use of hpterm at the OS/2 X server.



```
# pwd
/
# ls
.Xdefaults    SYSBCKUP    disc     lost+found   tmp
.profile      SYSDEBUG    etc      paul.test    users
.sh_history   bin         hp-ux    rolf         usr
.x11startlog  dev         lib      system       x0.hosts
# find / -name xload -print
/usr/bin/X11/xload
# /usr/bin/X11/xload -fg red -bg green -display 9.67.38.89:0.0
```

*Figure 104. An hpterm Window Displayed at an OS/2 X Server*

## 5.1.1.1 Modifying Hewlett-Packard X Client Application Resources

If you look closely at the commands that we issued from the hpterm window illustrated in Figure 104 you will notice that we actually started up a second X client application called xload and displayed the output at our OS/2 X server. xload displays a graphical picture of the HP 9000 system load. An example of xload window at our OS/2 X server is illustrated in Figure 106 on page 159.

Notice too that we specified some runtime options that defined the values for two xload application resources. In the command in Figure 104 we specified the foreground color as red and the background color as green.

Another way to set application resources for X clients on the HP 9000 is through the use of an application resource file. We did the following to set up an application resource file for the xload program.

1. We found a sample application resource file called /usr/lib/X11/sys.Xdefaults. We copied this file to our home directory by issuing the following command:

```
cp /usr/lib/X11/sys.Xdefaults .Xdefaults
```

We now had an application resource file .Xdefaults in our home directory.

2. In this file there are a number of sample resource definitions for several X client applications available on the HP 9000. Note that they are all commented out with an **!** in the first column.

   Using the vi editor, we added the lines as shown in Figure 105 for xload, ensuring that we removed the **!** from the first column.

```
XLoad*foreground:       red
XLoad*background:       green
XLoad*Font:             8x13
XLoad*geometry:         500x100+5+5
```

*Figure 105. Sample HP 9000 Application Resource File Definitions for xload*

3. We started xload with the following command at an HP-UX command prompt:

```
/usr/bin/X11/xload -display 9.67.38.89:0.0&
```

   The resulting window is illustrated in Figure 106. The area under the graph is red and the background is green. The font used for the test is 8x13, which is a font that is available on the OS/2 X server. The window has the dimensions 500 pixels wide by 100 pixels high and is placed 5 pixels in the X direction and 5 pixels in the Y direction from the upper left-hand corner of the display.

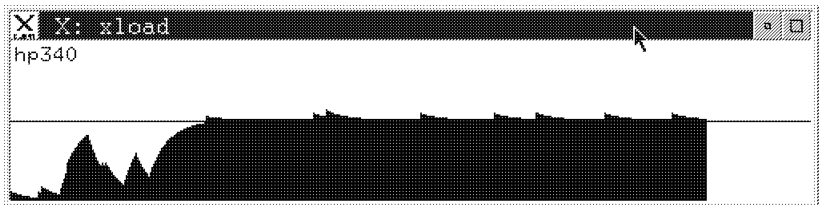


*Figure 106. An HP 9000 xload Window Displayed at an OS/2 X Server*

   In order to generate some load on the HP 9000 for the display illustrated in Figure 106 we wrote a little program with an endless loop in it.

### 5.1.2 Hewlett-Packard as an X Server

We tested the HP 9000 as an X server by running some X client applications on MVS and AIX/6000. Since the X client support on VM is virtually identical to that of MVS, we have not documented a case for VM as the X client platform.

In order to use the HP 9000 as an X server we had to first authorize both the MVS client and AIX/6000 client hosts. This was done as follows:

1. We added the MVS host name MVS18, and the AIX host name RS60001 to the HP 9000 X client authorization file /etc/X0.hosts using the vi editor. The entries we added for MVS18 and RS60001 are shown in Figure 107 on page 160.

```
mvs18
rs60001
```

*Figure 107. Entries in /etc/X0.hosts on the HP 9000*

2. In order to resolve the host names in /etc/X0.hosts to valid IP addresses we had to put entries for MVS18 and RS60001 in the HP 9000 /etc/hosts file. We did this using the vi editor. The entries are illustrated in Figure 108.

```
9.67.32.85    mvs18
9.67.38.135   rs60001
```

*Figure 108. Entries in /etc/hosts on the HP 9000*

#### 5.1.2.1 MVS Client Application

The most common MVS X client applications that will be displayed on OEM workstations will probably be GDDM applications. We chose ADMCHART to display at the HP 9000.

We performed the following to open an X window for ADMCHART at the HP9000:

1. We activated the X Window System GDDM interface on MVS18. Please refer to 2.2.2.2, "Installation Verification" on page 29 for details on how to set up and activate the X Window System GDDM interface for MVS.
2. We changed the entry in the data set userid.XWINDOWS.DISPLAY to identify the HP 9000 as the target display. Note that userid was the user ID of the TSO user under which we started ADMCHART. Refer to 2.2.1, "Installation Verification for the MVS X Window System API" on page 26 for further information on the data set userid.XWINDOWS.DISPLAY. The entry is illustrated in Figure 109.

```
9.67.32.66:0.0
```

*Figure 109. Entry for the HP 9000 Target Display in userid.XWINDOWS.DISPLAY*

3. In order to interact with ADMCHART you need to use a key that has the keysym set to EXECUTE. This is the key you would use when you normally press Enter. On the HP 9000 there is a utility called xprkbd that shows the relationship between keycodes and keysyms. We invoked xprkbd with the following command on the HP 9000:

   ```
   /usr/bin/X11/xprkbd | more
   ```

   From the resulting output we were able to determine that the key with keycode hexadecimal 57 was set to the keysym name EXECUTE.
4. In order to determine which key on the HP 9000 keyboard corresponded to keycode hexadecimal 57, we used the MVS X client application KEYCODE. From the TSO command prompt we typed:

   ```
   KEYCODE
   ```

Since we had earlier set the HP 9000 as the target display, the KEYCODE window was opened at the HP 9000. An example of the KEYCODE window is illustrated in Figure 44 on page 78. By using the KEYCODE display we were able to determine which of the HP 9000 keys corresponded to keycode 57.

5. We started up ADMCHART from the TSO command prompt and a GDDM display window was opened at the HP 9000.

***Remapping the HP 9000 Keyboard:*** The HP 9000 keyboard has only eight program function keys. ADMCHART requires a keyboard with twelve program function keys. This meant that we had to remap the HP 9000 keyboard to provide the program function keys 9 to 12. We performed the following steps to accomplish this task:

1. We chose four keys on the HP 9000 keyboard that we were not using, and to which we could map the keysym names F9, F10, F11, and F12. We used the MVS X client application KEYCODE to determine the keycodes for our chosen four HP 9000 keys. They had the keycodes hexadecimal 2D, 29, 2B, and 2F.
2. We invoked the xprkbd utility to check the current keysym settings for the keycodes 2D, 29, 2B, and 2F. This was done with the following command:

   ```
   /usr/bin/X11/xprkbd | more
   ```

3. The xmodmap utility under HP-UX allows you to remap the HP 9000 keyboard. Using the vi editor we created an input file for xmodmap in our home directory called gddmkeys. The contents of this file is illustrated in Figure 110. We executed xmodmap using the following command:

   ```
   /usr/bin/X11/xmodmap gddmkeys
   ```

   **Note:** You must execute xprkbd and xmodmap from the display and keyboard for which you want the keys remapped. Remember that the HP 9000 is a multi-user system and keeps a keyboard map for each keyboard.

```
keycode 0x2d = F9
keycode 0x29 = F10
keycode 0x2b = F11
keycode 0x2f = F12
```

*Figure 110. Input File for HP 9000 xmodmap Utility*

4. Using xprkbd, we confirmed that the keys with keycodes 2D, 29, 2B, and 2F had actually been set to F9, F10, F11, and F12 respectively.
5. We restarted ADMCHART from the TSO command prompt and a GDDM display window was opened at the HP 9000. Once we had created mappings for the function program keys 9 through 12 we found we had no further problems using ADMCHART with the HP 9000 as the X server.

#### 5.1.2.2 AIX/6000 Client Application

A typical AIX X client application that will be displayed on OEM workstations might be aixterm. We performed the following steps to open an X window for aixterm at the HP 9000:

1. From the HP 9000 we used Telnet to establish a session with the RISC System/6000. This was done with by issuing the following command at the HP 9000 command prompt:

   ```
   telnet rs60001
   ```

   We were then able to enter the AIX/6000 root user ID and password at the AIX/6000 login prompt.

2. At the AIX/6000 prompt we started the AIX X client application aixterm with the following command:

   ```
   aixterm -display 9.67.32.66:0.0
   ```

   An aixterm X window was opened at the HP 9000. We were able to use this window as if we were at the RISC System/6000 console.

# Appendix A. MVS C/370 Catalogued Procedures

## A.1 C/370 Compiler Catalogued Procedure EDCC

```
//*********************************************************/
//* EDCC   *                                              */
//***********      C/370 -- MVS                           */
//*          VERSION 2 RELEASE 1 MODIFICATION 0           */
//*                                                       */
//* 5688-187 (C) COPYRIGHT IBM CORP. 1988, 1991           */
//*                                                       */
//*                                                       */
//* EDCC    --- THIS CATALOGUED PROCEDURE COMPILES        */
//*             A C PROGRAM                               */
//*                                                       */
//*********************************************************/
//*                                                       */
//*USING THIS CATALOGUED PROCEDURE:                       */
//*-------------------------------                        */
//* WHEN USING THIS CATALOGUED PROCEDURE, THE USER MUST   */
//* SPECIFY THE QUALIFIED INPUT FILE NAME (SOURCE PROGRAM).*/
//*                                                       */
//* THE MINIMUM A USER CAN CODE TO INVOKE THIS PROCEDURE  */
//* IS:                                                   */
//*                                                       */
//*  //USERID   JOB...                                    */
//*  //STEPNAME EXEC EDCC,                                */
//*             INFILE='QUALIFIED DATA SET NAME'          */
//* DEFAULT VALUES WILL APPLY TO ALL REMAINING PARAMETERS. */
//*                                                       */
//*********************************************************/
//*                                                       */
//*SETTING UP THIS CATALOGUED PROCEDURE:                  */
//*-------------------------------------                  */
//*THE FOLLOWING SYMBOLIC PARAMETERS ARE USED IN THIS     */
//*CATALOGUED PROCEDURE AND MUST BE CUSTOMIZED WHEN IT IS */
//*INSTALLED. THE MEANINGS OF THE PARAMETERS ARE AS       */
//*FOLLOWS:                                               */
//*                                                       */
//*********************************************************/
//*                                                       */
//*SETTING UP THIS CATALOGUED PROCEDURE:                  */
//*-------------------------------------                  */
//*THE FOLLOWING SYMBOLIC PARAMETERS ARE USED IN THIS     */
//*CATALOGUED PROCEDURE AND MUST BE CUSTOMIZED WHEN IT IS */
//*INSTALLED. THE MEANINGS OF THE PARAMETERS ARE AS       */
//*FOLLOWS:                                               */
//*                                                       */
//*                                                       */
//* &CREGSIZ  - COMPILER REGION SIZE                      */
//* &INFILE   - INPUT DATA SET                            */
//* &OUTFILE  - OUTPUT DATA SET                           */
//* &OUTDCB   - DCB FOR OUTPUT FILE                       */
//* &CPARM    - COMPILER OPTIONS                          */
//*                                                       */
//* &VSCCHD   - PREFIX FOR SYSTEM FILES                   */
//* &COMHD    - PREFIX FOR COMMON LIBRARY                    */
//* &CVER     - VERSION OF COMPILER                          */
//* &COMVER   - VERSION OF COMMON LIBRARY                    */
//* &COMPL    - C COMPILER MODULES                           */
//* &EDCMSGS  - C COMPILER MESSAGES                          */
//* &LANG     - MESSAGE LANGUAGE                             */
//* &COMLINK  - COMMON DYNAMIC RUNTIME LIBRARY               */
//* &CLINK    - C DYNAMIC LIBRARY                            */
//* &EDCHDRS  - C SYSTEM HEADERS                             */
//* &WORKDA   - UNIT TYPE FOR WORK FILES                     */
//* &WRKSPC   - SPACE ALLOCATED FOR WORK FILES               */
//* &DCB80    - DCB FOR LRECL OF 80                          */
//* &DCB3200  - DCB FOR LRECL OF 3200                        */
//* &SYSOUT1  - COMPILER OUTPUT                              */
//* &SYSOUT6  - COMPILER LISTING                             */
//*                                                          */
//* THE SYMBOLIC PARAMETERS DIRECTLY FOLLOW THIS COMMENT     */
//*BLOCK AND MOST LIKELY REQUIRE CUSTOMIZATION BY YOUR       */
//*INSTALLATION.                                             */
//*                                                          */
//***********************************************************/
//EDCC PROC CREGSIZ='1536K',
//   INFILE=,
// OUTFILE='&&LOADSET,DISP=(MOD,PASS),UNIT=VIO,SPACE=(80,(250,100))',
//   CPARM=,
//*-------------------------------------------------------------
//   VSCCHD='EDC.',
//   COMHD='PLI.',
//   CVER='V2R1M0.',
//   COMVER='V2R3M0.',
//   COMPL='SEDCCOMP',
//   EDCMSGS='SEDCMSGS',
//   LANG='EDCMSGE',
//   COMLINK='SIBMLINK',
//   CLINK='SEDCLINK',
//   EDCHDRS='SEDCHDRS',
//*-------------------------------------------------------------
//   WORKDA='VIO',
//   WRKSPC='(32000,(30,30))',
//   OUTDCB='(RECFM=FB,LRECL=80,BLKSIZE=3200)',
//   DCB80='(RECFM=FB,LRECL=80,BLKSIZE=3200)',
//   DCB3200='(RECFM=FB,LRECL=3200,BLKSIZE=12800)',
//*------------------------------------------------------------------
//   SYSOUT1='*',
//   SYSOUT6='*'
//*------------------------------------------------------------------
//*  COMPILE STEP:
//*-------------------------------------------------------------------
//COMPILE EXEC PGM=EDCCOMP,PARM=(,
//   '&CPARM'),REGION=&CREGSIZ
//STEPLIB   DD DSN=&VSCCHD&CVER&CLINK,DISP=SHR
//          DD DSN=&COMHD&COMVER&COMLINK,DISP=SHR
//          DD DSN=&VSCCHD&CVER&COMPL,DISP=SHR
//SYSMSGS   DD DSN=&VSCCHD&CVER&EDCMSGS(&LANG),DISP=SHR
//SYSIN     DD DSN=&INFILE,DISP=SHR
//SYSLIB    DD DSN=&VSCCHD&CVER&EDCHDRS,DISP=SHR
//SYSLIN    DD DSN=&OUTFILE,DCB=&OUTDCB
//SYSPRINT  DD SYSOUT=&SYSOUT1
//SYSCPRT   DD SYSOUT=&SYSOUT6
```

```
//SYSUT1    DD DSN=&&SYSUT1,UNIT=&WORKDA,DISP=(NEW,DELETE),
//   SPACE=&WRKSPC,DCB=&DCB80
//SYSUT4    DD DSN=&&SYSUT4,UNIT=&WORKDA,DISP=(NEW,DELETE),
//   SPACE=&WRKSPC,DCB=&DCB80
//SYSUT5    DD DSN=&&SYSUT5,UNIT=&WORKDA,DISP=(NEW,DELETE),
//   SPACE=&WRKSPC,DCB=&DCB3200
//SYSUT6    DD DSN=&&SYSUT6,UNIT=&WORKDA,DISP=(NEW,DELETE),
//   SPACE=&WRKSPC,DCB=&DCB3200
//SYSUT7    DD DSN=&&SYSUT7,UNIT=&WORKDA,DISP=(NEW,DELETE),
//   SPACE=&WRKSPC,DCB=&DCB3200
//SYSUT8    DD DSN=&&SYSUT8,UNIT=&WORKDA,DISP=(NEW,DELETE),
//   SPACE=&WRKSPC,DCB=&DCB3200
//SYSUT9    DD DSN=&&SYSUT9,UNIT=&WORKDA,DISP=(NEW,DELETE),
//   SPACE=&WRKSPC,DCB=(RECFM=VB,LRECL=137,BLKSIZE=882)
//SYSUT10   DD SYSOUT=&SYSOUT6
//*
```

## A.2 C/370 Linkage Editor Catalogued Procedure EDCL

```
//* ***************************************************************/
//* EDCL    *                                                     */
//* *********      C/370 -- MVS                                   */
//*         VERSION 2,  RELEASE 1,  MODIFICATION 0                */
//*                                                               */
//* 5688-188 (C) COPYRIGHT IBM CORP. 1988, 1991.                  */
//*                                                               */
//*                                                               */
//* EDCL    --- THIS CATALOGUED PROCEDURE LINK-EDITS              */
//*             A PROGRAM TO THE C LIBRARIES                      */
//*                                                               */
//* ***************************************************************/
//*                                                               */
//* SETTING UP THIS CATALOGUED PROCEDURE:                         */
//* -------------------------------------                         */
//* THE FOLLOWING SYMBOLIC PARAMETERS ARE USED IN THIS            */
//* CATALOGUED PROCEDURE AND MUST BE CUSTOMIZED WHEN IT IS        */
//* INSTALLED. THE MEANINGS OF THE PARAMETERS ARE AS              */
//* FOLLOWS:                                                      */
//*                                                               */
//* &SYSOUT4  - LINKAGE EDITOR DIAGNOSTIC OUTPUT                  */
//*                                                               */
//* &LPARM    - LINKAGE EDITOR OPTIONS                            */
//* &VSCCHD   - PREFIX FOR SYSTEM FILES                           */
//* &COMHD    - PREFIX FOR COMMON LIBRARY                         */
//* &WORKDA   - UNIT TYPE FOR WORK FILES                          */
//* &COMVER   - VERSION OF COMMON LIBRARY                         */
//* &CVER     - VERSION OF C LIBRARY      1                       */
//* &CBASE    - C-LIBRARY STUBS                                   */
//* &COMBASE  - COMMON LIBRARY STUBS                              */
//* &LKMOD    - DATASET FOR LOAD MODULE                           */
//* &LKDISP   - DISPOSITION FOR LOAD MODULE                       */
//*                                                               */
//* ***************************************************************/
//EDCL    PROC LPARM='AMODE=31,MAP',
//   INFILE=,
// OUTFILE='&&GSET(GO),DISP=(MOD,PASS),UNIT=VIO,SPACE=(512,(50,20,1))',
//   VSCCHD='EDC.',
//   CVER='V2R1M0.',
//   CBASE='SEDCBASE',
//   COMHD='PLI.',
//   COMVER='V2R3M0.',
//   COMBASE='SIBMBASE',
//   WORKDA=VIO,
//   WRKSPC='(32000,(30,30))',
//   LKDISP='(NEW,PASS)',
//   SYSOUT='*'
//*-------------------------------------------------------------
//* LINKEDIT STEP:
//*-------------------------------------------------------------
//LKED    EXEC PGM=IEWL,PARM='&LPARM'
//SYSLIB    DD DSN=&VSCCHD&CVER&CBASE,DISP=(SHR,PASS)
//          DD DSN=&COMHD&COMVER&COMBASE,DISP=(SHR,PASS)
//SYSLIN    DD DSN=&INFILE,DISP=SHR
//SYSLMOD   DD DSN=&OUTFILE
//SYSPRINT  DD SYSOUT=&SYSOUT
//SYSUT1    DD DSN=&&SYSUT1,UNIT=&WORKDA,DISP=&LKDISP,SPACE=&WRKSPC
```

# Appendix B. Supplied Application Resource File Definitions for XCALC

```
XCalc*Cursor:                    hand2
XCalc.toplevel.icon_name:        Calculator
XCalc*iconPixmap:                xcalc.bm
XCalc*Font:                      *-helvetica-medium-r-normal--*-100-*-*-*-*-*-*
XCalc*bevel.screen.LCD.Font:     *-helvetica-bold-r-normal--*-120-*-*-*-*-*-*

XCalc*bevel.Background:                  black
XCalc*bevel.horizDistance:                       25
XCalc*bevel.vertDistance:                        6

XCalc*bevel.screen.horizDistance:                6
XCalc*bevel.screen.vertDistance:                 2
XCalc*bevel.screen.defaultDistance:              0

XCalc*bevel.screen.Label.horizDistance:          0
XCalc*bevel.screen.Label.vertDistance:           0
XCalc*bevel.screen.Label.internalHeight:         1
XCalc*bevel.screen.Label.internalWidth:          1

XCalc*bevel.screen.LCD.Label:            88888888888
XCalc*bevel.screen.LCD.fromHoriz:        M
XCalc*bevel.screen.LCD.horizDistance:            0
XCalc*bevel.screen.LCD.vertDistance:             0

XCalc*bevel.screen.INV.fromVert:         M
XCalc*bevel.screen.INV.vertDistance:             4

XCalc*bevel.screen.DEG.fromHoriz:        INV
XCalc*bevel.screen.DEG.fromVert:         LCD
XCalc*bevel.screen.DEG.horizDistance:            1

XCalc*bevel.screen.RAD.fromHoriz:        DEG
XCalc*bevel.screen.RAD.fromVert:         LCD

XCalc*bevel.screen.GRAD.fromHoriz:       RAD
XCalc*bevel.screen.GRAD.fromVert:        LCD

XCalc*bevel.screen.P.Label:              ()
XCalc*bevel.screen.P.fromHoriz:          GRAD
XCalc*bevel.screen.P.fromVert:           LCD
XCalc*bevel.screen.P.horizDistance:              2

XCalc*Command.horizDistance:                     2
XCalc*Command.vertDistance:                      4
XCalc*Command.width:                             32
XCalc*Command.height:                            15
XCalc*Command.internalWidth:                     1
XCalc*Command.ShapeStyle:                Oval

XCalc.ti.Geometry:                               171x252
XCalc.ti.bevel.screen.LCD.width:                 100
XCalc.ti.bevel.screen.LCD.ShapeStyle:    rectangle

XCalc.ti.bevel.screen.LCD.Translations: #replace\n\
Ctrl<Key>c:quit()\n\
Ctrl<Key>h:clear()\n\
None<Key>0:digit(0)\n\
None<Key>1:digit(1)\n\
None<Key>2:digit(2)\n\
None<Key>3:digit(3)\n\
None<Key>4:digit(4)\n\
None<Key>5:digit(5)\n\
None<Key>6:digit(6)\n\
None<Key>7:digit(7)\n\
None<Key>8:digit(8)\n\
None<Key>9:digit(9)\n\
<Key>KP_0:digit(0)\n\
<Key>KP_1:digit(1)\n\
<Key>KP_2:digit(2)\n\
<Key>KP_3:digit(3)\n\
<Key>KP_4:digit(4)\n\
<Key>KP_5:digit(5)\n\
<Key>KP_6:digit(6)\n\
<Key>KP_7:digit(7)\n\
<Key>KP_8:digit(8)\n\
<Key>KP_9:digit(9)\n\
<Key>KP_Enter:equal()\n\
<Key>KP_Equal:equal()\n\
<Key>KP_Multiply:multiply()\n\
<Key>KP_Add:add()\n\
<Key>KP_Subtract:subtract()\n\
<Key>KP_Decimal:decimal()\n\
<Key>KP_Divide:divide()\n\
<Key>KP_Tab:equal()\n\
<Key>Clear:clear()\n\
<Key>.:decimal()\n\
<Key>+:add()\n\
<Key>-:subtract()\n\
<Key>*:multiply()\n\
<Key>/:divide()\n\
<Key>(:leftParen()\n\
<Key>):rightParen()\n\
<Key>!:factorial()\n\
<Key>e:e()\n\
<Key>:power()\n\
<Key>p:pi()\n\
<Key>i:inverse()\n\
<Key>s:sine()\n\
<Key>c:cosine()\n\
<Key>t:tangent()\n\
<Key>d:degree()\n\
<Key>l:naturalLog()\n\
<Key>=:equal()\n\
<Key>n:negate()\n\
<Key>r:squareRoot()\n\
<Key>space:clear()\n\
<Key>q:quit()\n\
<Key>Delete:clear()\n\
<Key>BackSpace:clear()\n\
<Btn1Down>,<Btn1Up>:toggle()selection()\n
```

```
XCalc.ti.button1.Label:         1/x
XCalc.ti.button1.Translations:          #override<Btn1Up>:reciprocal()unset()
XCalc.ti.button2.Label          x2
XCalc.ti.button2.Translations:          #override<Btn1Up>:square()unset()
XCalc.ti.button3.Label:         SQRT
XCalc.ti.button3.Translations:          #override<Btn1Up>:squareRoot()unset()
XCalc.ti.button4.Label:         CE/C
XCalc.ti.button4.Translations:          #override<Btn1Up>:clear()unset()
XCalc.ti.button5.Label:         AC
XCalc.ti.button5.Translations:          #override<Btn1Up>:off()unset()\n\
                                        <Btn3Up>:quit()

XCalc.ti.button6.Label:         INV
XCalc.ti.button6.Translations:          #override<Btn1Up>:inverse()unset()
XCalc.ti.button7.Label:         sin
XCalc.ti.button7.Translations:          #override<Btn1Up>:sine()unset()
XCalc.ti.button8.Label:         cos
XCalc.ti.button8.Translations:          #override<Btn1Up>:cosine()unset()
XCalc.ti.button9.Label:         tan
XCalc.ti.button9.Translations:          #override<Btn1Up>:tangent()unset()
XCalc.ti.button10.Label:                DRG
XCalc.ti.button10.Translations:         #override<Btn1Up>:degree()unset()

XCalc.ti.button11.Label:                e
XCalc.ti.button11.Translations: #override<Btn1Up>:e()unset()
XCalc.ti.button12.Label:                EE
XCalc.ti.button12.Translations: #override<Btn1Up>:scientific()unset()
XCalc.ti.button13.Label:                log
XCalc.ti.button13.Translations: #override<Btn1Up>:logarithm()unset()
XCalc.ti.button14.Label:                ln
XCalc.ti.button14.Translations: #override<Btn1Up>:naturalLog()unset()
XCalc.ti.button15.Label:                yx
XCalc.ti.button15.Translations: #override<Btn1Up>:power()unset()

XCalc.ti.button16.Label:                PI
XCalc.ti.button16.Translations: #override<Btn1Up>:pi()unset()
XCalc.ti.button17.Label:                x!
XCalc.ti.button17.Translations: #override<Btn1Up>:factorial()unset()
XCalc.ti.button18.Label:                (
XCalc.ti.button18.Translations: #override<Btn1Up>:leftParen()unset()
XCalc.ti.button19.Label:                )
XCalc.ti.button19.Translations: #override<Btn1Up>:rightParen()unset()
XCalc.ti.button20.Label:                /
XCalc.ti.button20.Translations: #override<Btn1Up>:divide()unset()

XCalc.ti.button21.Label:                STO
XCalc.ti.button21.Translations: #override<Btn1Up>:store()unset()
XCalc.ti.button22.Label:                7
XCalc.ti.button22.Translations: #override<Btn1Up>:digit(7)unset()
XCalc.ti.button23.Label:                8
XCalc.ti.button23.Translations: #override<Btn1Up>:digit(8)unset()
XCalc.ti.button24.Label:                9
XCalc.ti.button24.Translations: #override<Btn1Up>:digit(9)unset()
XCalc.ti.button25.Label:                *
XCalc.ti.button25.Translations: #override<Btn1Up>:multiply()unset()

XCalc.ti.button26.Label:                RCL
XCalc.ti.button26.Translations: #override<Btn1Up>:recall()unset()
XCalc.ti.button27.Label:                4
XCalc.ti.button27.Translations: #override<Btn1Up>:digit(4)unset()
XCalc.ti.button28.Label:                5
XCalc.ti.button28.Translations: #override<Btn1Up>:digit(5)unset()
XCalc.ti.button29.Label:                6
XCalc.ti.button29.Translations: #override<Btn1Up>:digit(6)unset()
XCalc.ti.button30.Label:                -
XCalc.ti.button30.Translations: #override<Btn1Up>:subtract()unset()

XCalc.ti.button31.Label:                SUM
XCalc.ti.button31.Translations: #override<Btn1Up>:sum()unset()
XCalc.ti.button32.Label:                1
XCalc.ti.button32.Translations: #override<Btn1Up>:digit(1)unset()
XCalc.ti.button33.Label:                2
XCalc.ti.button33.Translations: #override<Btn1Up>:digit(2)unset()
XCalc.ti.button34.Label:                3
XCalc.ti.button34.Translations: #override<Btn1Up>:digit(3)unset()
XCalc.ti.button35.Label:                +
XCalc.ti.button35.Translations: #override<Btn1Up>:add()unset()

XCalc.ti.button36.Label:                EXC
XCalc.ti.button36.Translations: #override<Btn1Up>:exchange()unset()
XCalc.ti.button37.Label:                0
XCalc.ti.button37.Translations: #override<Btn1Up>:digit(0)unset()
XCalc.ti.button38.Label:                .
XCalc.ti.button38.Translations: #override<Btn1Up>:decimal()unset()
XCalc.ti.button39.Label:                +/-
XCalc.ti.button39.Translations: #override<Btn1Up>:negate()unset()
XCalc.ti.button40.Label:                =
XCalc.ti.button40.Translations: #override<Btn1Up>:equal()unset()


XCalc.ti.button1.horizDistance: 4
XCalc.ti.button1.fromVert:              bevel
XCalc.ti.button2.fromHoriz:             button1
XCalc.ti.button2.fromVert:              bevel
XCalc.ti.button3.fromHoriz:             button2
XCalc.ti.button3.fromVert:              bevel
XCalc.ti.button4.fromHoriz:             button3
XCalc.ti.button4.fromVert:              bevel
XCalc.ti.button5.fromHoriz:             button4
XCalc.ti.button5.fromVert:              bevel

XCalc.ti.button6.horizDistance: 4
XCalc.ti.button6.fromVert:              button1
XCalc.ti.button7.fromHoriz:             button6
XCalc.ti.button7.fromVert:              button2
XCalc.ti.button8.fromHoriz:             button7
XCalc.ti.button8.fromVert:              button3
XCalc.ti.button9.fromHoriz:             button8
XCalc.ti.button9.fromVert:              button4
XCalc.ti.button10.fromHoriz:            button9
XCalc.ti.button10.fromVert:             button5

XCalc.ti.button11.horizDistance:        4
XCalc.ti.button11.fromVert:             button6
XCalc.ti.button12.fromHoriz:            button11
XCalc.ti.button12.fromVert:             button7
XCalc.ti.button13.fromHoriz:            button12
XCalc.ti.button13.fromVert:             button8
XCalc.ti.button14.fromHoriz:          button13
XCalc.ti.button14.fromVert:           button9
XCalc.ti.button15.fromHoriz:          button14
XCalc.ti.button15.fromVert:           button10

XCalc.ti.button16.horizDistance:      4
XCalc.ti.button16.fromVert:           button11
XCalc.ti.button17.fromHoriz:          button16
XCalc.ti.button17.fromVert:           button12
XCalc.ti.button18.fromHoriz:          button17
XCalc.ti.button18.fromVert:           button13
XCalc.ti.button19.fromHoriz:          button18
XCalc.ti.button19.fromVert:           button14
XCalc.ti.button20.fromHoriz:          button19
XCalc.ti.button20.fromVert:           button15

XCalc.ti.button21.horizDistance:      4
XCalc.ti.button21.fromVert:           button16
XCalc.ti.button22.fromHoriz:          button21
XCalc.ti.button22.fromVert:           button17
XCalc.ti.button23.fromHoriz:          button22
XCalc.ti.button23.fromVert:           button18
XCalc.ti.button24.fromHoriz:          button23
XCalc.ti.button24.fromVert:           button19
XCalc.ti.button25.fromHoriz:          button24
XCalc.ti.button25.fromVert:           button20

XCalc.ti.button26.horizDistance:      4
XCalc.ti.button26.fromVert:           button21
XCalc.ti.button27.fromHoriz:          button26
XCalc.ti.button27.fromVert:           button22
XCalc.ti.button28.fromHoriz:          button27
XCalc.ti.button28.fromVert:           button23
XCalc.ti.button29.fromHoriz:          button28
XCalc.ti.button29.fromVert:           button24
XCalc.ti.button30.fromHoriz:          button29
XCalc.ti.button30.fromVert:           button25

XCalc.ti.button31.horizDistance:      4
XCalc.ti.button31.fromVert:           button26
XCalc.ti.button32.fromHoriz:          button31
XCalc.ti.button32.fromVert:           button27
XCalc.ti.button33.fromHoriz:          button32
XCalc.ti.button33.fromVert:           button28
XCalc.ti.button34.fromHoriz:          button33
XCalc.ti.button34.fromVert:           button29
XCalc.ti.button35.fromHoriz:          button34
XCalc.ti.button35.fromVert:           button30

XCalc.ti.button36.horizDistance:      4
XCalc.ti.button36.fromVert:           button31
XCalc.ti.button37.fromHoriz:          button36
XCalc.ti.button37.fromVert:           button32
XCalc.ti.button38.fromHoriz:          button37
XCalc.ti.button38.fromVert:           button33
XCalc.ti.button39.fromHoriz:          button38
XCalc.ti.button39.fromVert:           button34
XCalc.ti.button40.fromHoriz:          button39
XCalc.ti.button40.fromVert:           button35
XCalc.hp.Geometry:                      336x164
XCalc.hp.bevel.screen.LCD.width:        180



XCalc.hp.bevel.screen.LCD.Translations: #replace\n\
                                Ctrl<Key>c:quit()\n\
                                Ctrl<Key>h:back()\n\
                                None<Key>0:digit(0)\n\
                                None<Key>1:digit(1)\n\
                                None<Key>2:digit(2)\n\
                                None<Key>3:digit(3)\n\
                                None<Key>4:digit(4)\n\
                                None<Key>5:digit(5)\n\
                                None<Key>6:digit(6)\n\
                                None<Key>7:digit(7)\n\
                                None<Key>8:digit(8)\n\
                                None<Key>9:digit(9)\n\
                                <Key>KP_0:digit(0)\n\
                                <Key>KP_1:digit(1)\n\
                                <Key>KP_2:digit(2)\n\
                                <Key>KP_3:digit(3)\n\
                                <Key>KP_4:digit(4)\n\
                                <Key>KP_5:digit(5)\n\
                                <Key>KP_6:digit(6)\n\
                                <Key>KP_7:digit(7)\n\
                                <Key>KP_8:digit(8)\n\
                                <Key>KP_9:digit(9)\n\
                                <Key>KP_Enter:enter()\n\
                                <Key>KP_Multiply:multiply()\n\
                                <Key>KP_Add:add()\n\
                                <Key>KP_Subtract:subtract()\n\
                                <Key>KP_Decimal:decimal()\n\
                                <Key>KP_Divide:divide()\n\
                                <Key>.:decimal()\n\
                                <Key>+:add()\n\
                                <Key>-:subtract()\n\
                                <Key>*:multiply()\n\
                                <Key>/:divide()\n\
                                <Key>!:factorial()\n\
                                <Key>e:e()\n\
                                <Key>:power()\n\
                                <Key>p:pi()\n\
                                <Key>i:inverse()\n\
                                <Key>s:sine()\n\
                                <Key>c:cosine()\n\
                                <Key>t:tangent()\n\
                                <Key>d:degree()\n\
                                <Key>l:naturalLog()\n\
                                <Key>n:negate()\n\
                                <Key>r:squareRoot()\n\
                                <Key>space:clear()\n\
                                <Key>q:quit()\n\
                                <Key>Delete:back()\n\
                                <Key>Return:enter()\n\
                                <Key>Linefeed:enter()\n\
                                <Key>x:XexchangeY()\n\
                                 <Key>BackSpace:back()\n\
                                 <Btn1Down>,<Btn1Up>:toggle()selection()\n



XCalc.hp.button1.Label:         SQRT
XCalc.hp.button1.Translations:          #override<Btn1Up>:squareRoot()unset()
XCalc.hp.button2.Label:         ex
XCalc.hp.button2.Translations:          #override<Btn1Up>:epower()unset()
XCalc.hp.button3.Label:         10x
XCalc.hp.button3.Translations:          #override<Btn1Up>:tenpower()unset()
XCalc.hp.button4.Label:         yx
XCalc.hp.button4.Translations:          #override<Btn1Up>:power()unset()
XCalc.hp.button5.Label:         1/x
XCalc.hp.button5.Translations:          #override<Btn1Up>:reciprocal()unset()
XCalc.hp.button6.Label:         CHS
XCalc.hp.button6.Translations:          #override<Btn1Up>:negate()unset()
XCalc.hp.button7.Label:         7
XCalc.hp.button7.Translations:          #override<Btn1Up>:digit(7)unset()
XCalc.hp.button8.Label:         8
XCalc.hp.button8.Translations:          #override<Btn1Up>:digit(8)unset()
XCalc.hp.button9.Label:         9
XCalc.hp.button9.Translations:          #override<Btn1Up>:digit(9)unset()
XCalc.hp.button10.Label:                /
XCalc.hp.button10.Translations: #override<Btn1Up>:divide()unset()

XCalc.hp.button11.Label:                x!
XCalc.hp.button11.Translations: #override<Btn1Up>:factorial()unset()
XCalc.hp.button12.Label:                PI
XCalc.hp.button12.Translations: #override<Btn1Up>:pi()unset()
XCalc.hp.button13.Label:                sin
XCalc.hp.button13.Translations: #override<Btn1Up>:sine()unset()
XCalc.hp.button14.Label:                cos
XCalc.hp.button14.Translations: #override<Btn1Up>:cosine()unset()
XCalc.hp.button15.Label:                tan
XCalc.hp.button15.Translations: #override<Btn1Up>:tangent()unset()
XCalc.hp.button16.Label:                EEX
XCalc.hp.button16.Translations: #override<Btn1Up>:scientific()unset()
XCalc.hp.button17.Label:                4
XCalc.hp.button17.Translations: #override<Btn1Up>:digit(4)unset()
XCalc.hp.button18.Label:                5
XCalc.hp.button18.Translations: #override<Btn1Up>:digit(5)unset()
XCalc.hp.button19.Label:                6
XCalc.hp.button19.Translations: #override<Btn1Up>:digit(6)unset()
XCalc.hp.button20.Label:                *
XCalc.hp.button20.Translations: #override<Btn1Up>:multiply()unset()

XCalc.hp.button21.Label:
XCalc.hp.button22.Label:
XCalc.hp.button23.Label:                Rv
XCalc.hp.button23.Translations: #override<Btn1Up>:roll()unset()
XCalc.hp.button24.Label:                x:y
XCalc.hp.button24.Translations: #override<Btn1Up>:XexchangeY()unset()
XCalc.hp.button25.Label:                <-
XCalc.hp.button25.Translations: #override<Btn1Up>:back()unset()
XCalc.hp.button26.Label:                ENTR
XCalc.hp.button26.Translations: #override<Btn1Up>:enter()unset()
XCalc.hp.button27.Label:                1
XCalc.hp.button27.Translations: #override<Btn1Up>:digit(1)unset()
XCalc.hp.button28.Label:                2
XCalc.hp.button28.Translations: #override<Btn1Up>:digit(2)unset()
XCalc.hp.button29.Label:                3
XCalc.hp.button29.Translations: #override<Btn1Up>:digit(3)unset()
XCalc.hp.button30.Label:                -
XCalc.hp.button30.Translations: #override<Btn1Up>:subtract()unset()

XCalc.hp.button31.Label:                ON
XCalc.hp.button31.Translations: #override<Btn1Up>:off()unset()\n\
                                                 <Btn3Up>:quit()
XCalc.hp.button32.Label:                DRG
XCalc.hp.button32.Translations: #override<Btn1Up>:degree()unset()
XCalc.hp.button33.Label:                INV
XCalc.hp.button33.Translations: #override<Btn1Up>:inverse()unset()
XCalc.hp.button34.Label:                STO
XCalc.hp.button34.Translations: #override<Btn1Up>:store()unset()
XCalc.hp.button35.Label:                RCL
XCalc.hp.button35.Translations: #override<Btn1Up>:recall()unset()
XCalc.hp.button36.Label:                0
XCalc.hp.button36.Translations: #override<Btn1Up>:digit(0)unset()
XCalc.hp.button37.Label:                .
XCalc.hp.button37.Translations: #override<Btn1Up>:decimal()unset()
XCalc.hp.button38.Label:                SUM
XCalc.hp.button38.Translations: #override<Btn1Up>:sum()unset()
XCalc.hp.button39.Label:                +
XCalc.hp.button39.Translations: #override<Btn1Up>:add()unset()


XCalc.hp.button1.horizDistance: 4
XCalc.hp.button1.fromVert:              bevel
XCalc.hp.button2.fromHoriz:             button1
XCalc.hp.button2.fromVert:              bevel
XCalc.hp.button3.fromHoriz:             button2
XCalc.hp.button3.fromVert:              bevel
XCalc.hp.button4.fromHoriz:             button3
XCalc.hp.button4.fromVert:              bevel
XCalc.hp.button5.fromHoriz:             button4
XCalc.hp.button5.fromVert:              bevel
XCalc.hp.button6.fromHoriz:             button5
XCalc.hp.button6.fromVert:              bevel
XCalc.hp.button7.fromHoriz:             button6
XCalc.hp.button7.fromVert:              bevel
XCalc.hp.button8.fromHoriz:             button7
XCalc.hp.button8.fromVert:              bevel
XCalc.hp.button9.fromHoriz:             button8
XCalc.hp.button9.fromVert:              bevel
XCalc.hp.button10.fromHoriz:            button9
XCalc.hp.button10.fromVert:             bevel

XCalc.hp.button11.horizDistance:        4
XCalc.hp.button11.fromVert:             button1
XCalc.hp.button12.fromHoriz:            button11
XCalc.hp.button12.fromVert:             button2
XCalc.hp.button13.fromHoriz:            button12
XCalc.hp.button13.fromVert:             button3
XCalc.hp.button14.fromHoriz:            button13
XCalc.hp.button14.fromVert:             button4
XCalc.hp.button15.fromHoriz:            button14
XCalc.hp.button15.fromVert:             button5
XCalc.hp.button16.fromHoriz:           button15
XCalc.hp.button16.fromVert:            button6
XCalc.hp.button17.fromHoriz:           button16
XCalc.hp.button17.fromVert:            button7
XCalc.hp.button18.fromHoriz:           button17
XCalc.hp.button18.fromVert:            button8
XCalc.hp.button19.fromHoriz:           button18
XCalc.hp.button19.fromVert:            button9
XCalc.hp.button20.fromHoriz:           button19
XCalc.hp.button20.fromVert:            button10

XCalc.hp.button21.horizDistance:       4
XCalc.hp.button21.fromVert:            button11
XCalc.hp.button22.fromHoriz:           button21
XCalc.hp.button22.fromVert:            button12
XCalc.hp.button23.fromHoriz:           button22
XCalc.hp.button23.fromVert:            button13
XCalc.hp.button24.fromHoriz:           button23
XCalc.hp.button24.fromVert:            button14
XCalc.hp.button25.fromHoriz:           button24
XCalc.hp.button25.fromVert:            button15
XCalc.hp.button26.fromHoriz:           button25
XCalc.hp.button26.fromVert:            button16
XCalc.hp.button26.height:              36
XCalc.hp.button27.fromHoriz:           button26
XCalc.hp.button27.fromVert:            button17
XCalc.hp.button28.fromHoriz:           button27
XCalc.hp.button28.fromVert:            button18
XCalc.hp.button29.fromHoriz:           button28
XCalc.hp.button29.fromVert:            button19
XCalc.hp.button30.fromHoriz:           button29
XCalc.hp.button30.fromVert:            button20

XCalc.hp.button31.horizDistance:       4
XCalc.hp.button31.fromVert:            button21
XCalc.hp.button32.fromHoriz:           button31
XCalc.hp.button32.fromVert:            button22
XCalc.hp.button33.fromHoriz:           button32
XCalc.hp.button33.fromVert:            button23
XCalc.hp.button34.fromHoriz:           button33
XCalc.hp.button34.fromVert:            button24
XCalc.hp.button35.fromHoriz:           button34
XCalc.hp.button35.fromVert:            button25
XCalc.hp.button36.fromHoriz:           button26
XCalc.hp.button36.fromVert:            button27
XCalc.hp.button37.fromHoriz:           button36
XCalc.hp.button37.fromVert:            button28
XCalc.hp.button38.fromHoriz:           button37
XCalc.hp.button38.fromVert:            button29
XCalc.hp.button39.fromHoriz:           button38
XCalc.hp.button39.fromVert:            button30
```

# Appendix C. Information on Zapping the VM GXDEMOx Programs

If you made a copy of the existing named GDDM shared segment with a different name for use by the X Window System GDDM interface modules, you must zap the load modules for those products that will be used with the new GDDM shared segment for the X Window System GDDM interface. This must also be done for the supplied demonstration programs GMDEMO1, GMDEMO2, GMDEMO3, GMDEMO4, GMDEMO4a, GMDEMO5, and GMDEMO6. This is to point each load module to the name of the new GDDM shared segment that must be accessed in order to use the X Window System GDDM interface.

You can use the following steps as a guide to zapping the GXDEMOx load modules:

1. On our VM system we have a production GDDM shared segment called ADMXA230. When we installed the X Window System GDDM interface we reinstalled a copy of this GDDM shared segment and called it GDDMXD.

   Each GXDEMOx MODULE points to ADMXA230. In order to run each module with the X Window System GDDM interface we zapped each GXDEMOx MODULE to point to GDDMXD.

2. You can zap a module using the ZAP command under VM. This command can either accept input from the CMS command line or from an input file called *filename* ZAP where *filename* can be any name you choose. We recommend using an input file because it minimizes the chance of a typing error.

   Create a ZAP input file for each of the GXDEMOx modules on your A disk and call them:

   ```
   GMDEMO1 ZAP
   GMDEMO2 ZAP
   GMDEMO3 ZAP
   GMDEMO4 ZAP
   GMDEMO4A ZAP
   GMDEMO5 ZAP
   GMDEMO6 ZAP
   ```

   The zap input file for GXDEMO1 is illustrated in Figure 111 on page 178. The contents of the zap input files for the other modules will be the same except for the module name.

   In each zap input file, you must name the module you are going to zap (one of GXDEMO1 to GXDEMO6), name the section of the module to be zapped (in this case it will be ADMASSNV for each module), verify the contents of the bytes you are going to replace, and specify the new byte values that will replace the bytes that were verified. The starting location of the bytes to be replaced is specified by an offset from the beginning of the ADMASSNV.

   In the ZAP input file in Figure 111 on page 178 we verify that the contents of the 8 bytes at offset 0000 is C1C4D4E7C1F2F3F0, which is the byte representation of ADMXA230. We then replace these bytes, again at offset 0000 with C7C4C4D4E7C44040, which is the byte representation of GDDMXD.

```
NAME GXDEMO1 ADMASSNV
VER 0000 C1C4D4E7C1F2F3F0
REP 0000 C7C4C4D4E7C44040
DUMP GXDEMO1 ADMASSNV
```

*Figure 111. Contents of the ZAP Input File GXDEMO1 ZAP*

3. Execute ZAP by typing the following command at the CMS command line:

   ```
   ZAP MODULE (INPUT GXDEMOn
   ```

   where GXDEMOn is the zap input file name.

# Appendix D. Standard X Client Applications

***appres:*** Prints the resources of a program, which a client would load from various sources. Resources that are commented out are not printed.

***bdftosnf:*** Converts .bdf (bitmap distribution format) font files into .snf (server natural format) font files. .snf is the format X can access and is server specific. The .bdf format is portable.

***bitmap:*** Tool for creating and editing bitmaps.

***listres:*** Lists the resources for the widgets.

***mkfontdir:*** Creates the fonts.dir and fonts.alias files. fonts.dir contains the names of all the font files stored in this directory either as file names or in the XLFD format. It is the mapping between the font file name and the XLFD description. The fonts.alias file (may be fonts.ali file) must be edited by hand. In that file the logical name (alias) can be set for specific fonts. A physical font can have more than one logical font name. The X server is searching in both files when a client requests to load a specific font. To specify a font use either the XLFD description (* as wild cards allowed) or the logical font name (alias). To select the font for an application either use the XLFD description (fonts.dir) or the alias name (fonts.alias) for that font.

***mwm:*** The Motif window manager provides all of the standard window management functions. This command will be started after the initialization of the X server. You can start the mwm on any workstation that has implemented the X server protocol and if a window manager is not running yet.

***oclock:*** Displays the time of the day in analog form. This client uses the shape extension, which supports no rectangular windows. Use it to test if your server supports the shape extension, which is part of the X11R4 version.

***resize:*** Program for use on systems that lack the ability to automatically notify processes of window size changes.

***showsnf:*** Displays the contents of font files in server natural fonts. Normally it is used to check whether the font file has been corrupted.

***xbiff:*** Displays a mailbox on the display and checks if mail has arrived. When the user gets mail the flag raises and the background color changes.

***xcalc:*** Scientific calculator that can emulate either a TI-30 or an HP-10C.

***xclipboard:*** Used to collect and display text selections that are sent to the clipboard by other clients.

***xclock:*** Displays a clock either analog or digital on the display. This client uses a rectangular window opposite to the oclock client.

***xcutsel:*** Copies the current selection (marked text with left mouse button) into a cut buffer. The user can use this selection and copy it into the cut buffer. It acts as a bridge between applications that don′t support selections. Unfortunately this client doesn′t work with the AIX/6000 InfoExplorer or with the OS/2 Presentation Manager.

***xditview:*** Client that is used to display ditroff output in a window.

***xdm:*** Provides the ″login services″ for X terminals and X servers. xdm uses the xdmcp (X display manager control program) to communicate with servers. To get an xdm session the server must have implemented the xdmcp protocol. xdm provides the log in and password, authenticates the user and runs a session which is customizable.

***xdpyinfo:*** A very useful tool for displaying information about an X server. Displays information about the color table, version, extensions, and other useful information.

***xedit:*** This client is a simple text editor for X.

***xev:*** Prints contents of X events. An event may be pressing a key on the keyboard, pressing a button or moving the mouse. It is an important tool that helps you when you want to change the mapping of a keyboard.

***xfd:*** Displays all printable characters of a font file.

***xfontsel:*** A way to display the fonts known to your X server. For each font you examine the full XLFD name is printed in the window. You can change the XLFD name, and this affects the font that you see in the window.

***xhost:*** Used to allow clients access to the X server. With this command you can control the access of the client hosts. There is only an access control for hosts and not for users. Normally the servers allow access only to programs running on the same machine or from machines listed in the file /etc/X*n*.hosts, where *n* is the display number of the server.

***xinit:*** Shell script to start the X server and some specific clients that the user can customize.

***xkill:*** Kills a client application. When the user starts the client without option the program asks the user to select a window which is to be killed. Moving the cursor to a window and pressing the button kills the selected client.

***xload:*** Displays a periodically updated histogram of the system load average (the client host).

***xlogo:*** Displays the X Windows System logo.

***xlsclients:*** Lists the client applications that are running on a display.

***xlsfonts:*** Lists all fonts or all fonts on the server that match the pattern. The font path must be set so that this client can show the fonts in the directory. The output is the XLFD format. The font path can be shown by the xset command.

***xlswins:*** Lists the window tree.

***xmag:*** Client that allows you to magnify a portion of the screen. The content of the portion will be shown magnified in another window.

***xman:*** A very nice tool to show the man pages of the X client.

***xmh:*** This is the X window interface to the mh message handling system.

***xmodmap:*** Client for displaying and changing the map of the X keyboard time.

***xpr:*** Prints an X window dump. It takes the output file produced by the xwd client as input, converts it into a PostScript file, and prints the file on the printer. The input can be scaled.

***xprop:*** Shows the properties of an X window that you have selected.

***xrdb:*** Is used to get or set the contents of the RESOURCE_MANAGER property on the root window. xrdb loads the property into the server normally at startup. In this way all the necessary information (where it is available to all clients) is stored in the server. This solves the problem which existed in previous versions of X that required you to maintain defaults files on every machine that you might use.

***xrefresh:*** Client that repaints your screen. Useful when system messages have been displayed on the screen. This client sends a refresh event to everything on the screen.

***xset:*** Important tool to show and set various user preference options of the display and keyboard. It sets the acceleration and threshold of the mouse, controls the bell volume, sets the font path, and controls the screen saver.

***xsetroot:*** Sets the color or bitmaps of a root window. This command doesn′t work on the OS/2 server for the X clients that are under control of Presentation Manager and for which there is no root window on the OS/2 display.

***xterm:*** This is a terminal emulator for the X Window System. It provides DEC VT102 and Tektronix 4014 support. AIXwindows delivers the xterm source code. AIXwindows Environment/6000 provides the aixterm client instead.

***xwd:*** Stores window images in a specially formatted window dump file. This file will be input for the xpr client. The target window is selected by clicking the mouse button in the desired window. When the bell rings twice the dump has been completed.

***xwininfo:*** Displays information about windows.

***xwud:*** Displays a file, created with the xwd command, on the screen. This command is similar to the xpr command, which prints the output on the printer.

# Appendix E.  XEDIT Subcommands

Subcommands and further information to use the XEDIT editor:

- Command Buttons
  - Quit

    Quits the current editing session. If any changes have not been saved, XEDIT displays a warning message, allowing the user to save the file.
  - Save

    If file backups are enabled (see ″Resources″), XEDIT stores a copy of the original, unedited file in <prefix>filename<suffix>, then overwrites the filename with the contents of the edit window. The filename is retrieved from the Text widget directly to the right of the Load button.
  - Load

    Loads the file named in the Text widget immediately to the right of this button and displays it in the Edit window. If the currently displayed file has been modified, a warning message will ask the user to save the changes or to press Load again.
- Editing

  The Athena Text widget is used for the three sections of this application that allow text input, namely the Message window, the Edit window, and the window to the right of the command buttons, in which a filename can be entered.

  The characters typed will go to the Text widget that the pointer is currently over. If the pointer is not over a Text widget, then the keystrokes will have no effect on the application. This is also true for the special key sequences that pop-up dialog widgets; so, for example, typing CTRL-s in the filename widget (next to the command buttons) will enable searching in that widget, not the Edit window (edit widget).

  Both the Message window and the Edit window will create a scrollbar if the text to display is too large to fit in that window. Horizontal scrolling is not allowed by default, but can be turned on through the Text widget′s resources.

  The following list summarizes the editing commands recognized by XEDIT (that is, by the Text widget).

| | |
|---|---|
| Control-a | Move to the beginning of the current line. |
| Control-b | Move backward one character. |
| Control-d | Delete the next character. |
| Control-e | Move to the end of the current line. |
| Control-f | Move forward one character. |
| Control-h or Backspace | Delete the previous character. |
| Control-j, Control-m, | New line. |

| | |
|---|---|
| Return, or LineFeed | |
| Control-k | Kill the rest of this line. (Does not kill the carriage return at the end of the line. To do so, use Control-k twice. However, be aware that the second kill overwrites the text line in the kill buffer.) |
| Control-l | Redraw the window. (Also scrolls text so the cursor is positioned in the middle of the window.) |
| Control-n | Move down to the next line. |
| Control-o | Divide this line into two lines at this point and move the cursor back up. |
| Control-p | Move up to the previous line. |
| Control-r | Search and replace backward. |
| Control-s | Search and replace forward. |
| Control-t | Transpose characters. (Swap the characters immediately before and after the cursor.) |
| Control-u | Perform next command four times. For example, the sequence Control-u, Control-n moves the cursor down four lines. |
| Control-v | Move down to the next screen of text. |
| Control-w | Kill the selected text. |
| Control-y | Insert the last killed text. (If the last killed text is a carriage return--see Control-k above--a blank line is inserted.) |
| Control-z | Scroll up the text one line. |
| Meta-< | Move to the beginning of the file. |
| Meta-> | Move to the end of the file. |
| Meta-· | Move backward one paragraph. |
| Meta-‘ | Move forward one paragraph. |
| Meta-b | Move backward one word. |
| Meta-d | Delete the next word. |
| Meta-D | Kill the next word. |
| Meta-f | Move forward one word. |
| Meta-h, Meta-Backspace, or Meta-Delete | Delete the previous word. |

| | |
|---|---|
| Meta-H, Meta-Shift-Backspace, or Meta-Shift-Delete | Kill the previous word. |
| Meta-i | Insert a file. A dialog box will appear in which you can type the desired filename. |
| Meta-k | Kill to the end of the paragraph. |
| Meta-q | Join lines to form a paragraph. |
| Meta-v | Move up to the previous screen of text. |
| Meta-y | Insert the last selected text here. This command is the equivalent of clicking the second pointer button. See Chapter 5, The xterm Terminal Emulator, for more information about text selections. |
| Meta-z | Scroll down the text one line. |
| Delete | Delete the previous character. |

Note that a translation in the application defaults file overrides the translation for the Return key for the text window in which a filename can be entered (next to the command buttons); in this window only, instead of starting a new line, Return moves the editing cursor to the end of the current line.

# Glossary

**bit plane**. On a color display, each pixel has more than one bit defined. Data in display memory can be either pixels (multiple bits per pixel) or bit planes. There is a 1-bit plane for each usable bit in the pixel.

**bitmap**. A pixmap with a depth of one bit plane.

**client**. An application program that connects to X Windows server by an InterProcess Communication (IPC) path, such as a Transmission Control Protocol (TCP) connection or a shared memory buffer. The program can be referred to as the client of the server, but it is actually the IPC path itself. Programs with multiple paths open to the server are viewed as multiple clients by the protocol.

**colorcell**. An entry in a colormap that consists of three values based on red, green, and blue intensities. The values are 16-bit, unsigned numbers. Zero represents the minimum intensity. The values are scaled by the server to match the particular display in use.

**colormap**. A set of colorcells. A pixel value indexes the colormap to produce RGB-value intensities. A colormap consists of a set of entries defining color values that, when associated with a window, is used to display the contents of the window. Depending on hardware limitations, one or more colormaps can be installed at one time, such that windows associated with those maps display correct colors. Two classes of colormaps are direct color and pseudocolor.

**connection**. The IPC path between the server and a client program. A client program typically, but not necessarily, has one connection to the server over which requests and events are sent.

**DBCS**. (Double-Byte Character Set) A set of characters in which each character is represented in 2 bytes of storage. Languages such as Japanese, Chinese, and Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets. Because each character requires 2 bytes, the typing, display, and printing of DBCS characters requires hardware and programs that support DBCS.

**DECnet**. Digital Equipment Corporation′s proprietary network architecture.

**depth**. The number of bits per pixel for a window or pixmap.

**display**. A set of one or more physical screens driven by a single X server. The DISPLAY variable tells the clients which server to connect to.

**event**. Information generated either asynchronously from a device or as the side effect of a client request. Events are grouped into types and are not sent to a client by the server unless the client has issued a specific request for information of that type. Events are usually reported relative to a window.

**font**. A set of glyphs, usually characters. The protocol does not translate or interpret character sets. The client indicates values used to access the glyph arrays. A font contains additional metric information to determine inter-glyph and inter-line spacing.

**font directory**. A directory where several font files exist including the file fonts.dir which contains entries for each font in the directory including the XLFD convention for each fonts.

**GDDM**. (Graphical Data Display Manager) A group of routines that allows pictures to be defined and displayed procedurally through function routines that correspond to graphic primitives.

**GL**. Graphics library.

**glyph**. (1) An image, usually of a character, in a font. (2) A graphic symbol whose appearance conveys information; for example, the vertical and horizontal arrows on a cursor key that indicate the direction in which they control cursor movement, the sunburst symbol on the screen illumination control of a display device.

**graPHIGS**. An implementation of PHIGS used by IBM and based on the ANSI proposed standard, Programmer′s Hierarchical Interactive Graphics System (PHIGS).

**gray scale**. A type of degenerate pseudocolor where the red, green, and blue values in any given colormap entry are equal, thus producing shades of gray.

**GUI**. (Graphical User Interface) A type of computer interface consisting of a visual metaphor of a real-world scene, often a desktop. Within that scene are icons, representing actual objects, that the user can access and manipulate with a pointing device.

**icon**. A graphic symbol, displayed on a screen, that a user can point to with a device such as a mouse in order to select a particular function or software application. Synonymous with pictogram.

**Intrinsics**. A set of management mechanisms that provides for constructing and interfacing between composite widgets, their children, and other clients. Also, provides the ability to organize a collection of widgets into an application.

**keycode**. Number (between 8 and 255) which represents a physical key on a keyboard. The keycode is fixed for each key and cannot be changed.

**keysym**. (Key symbol) The name that represents the label of a key. Keysyms are mapped to keycodes at the server.

**MBCS**. (Multi Byte Character Set) A set of characters in which each character is represented in 2 or more bytes of storage.

**modifier**. A logical keyname which represents functions that are recognized by X programs. There are 8 modifiers which can be assigned to a keycode (Shift, Lock, Control, Mod1, Mod2, Mod3, Mod4, Mod5).

**PEX**. (PHIGS Extension to X). The 3-D extension to X Windows.

**pixel value**. The number of bit planes used in a particular window or pixmap. For a window, a pixel value indexes a colormap and derives an actual color to be displayed. A pixel is an N-bit value, where N is the number of bit planes (the depth) used in a particular window or pixmap.

**pixmap**. A data representation arranged in a three-dimensional array of bits. A pixmap can be thought of as a two-dimensional array of pixels, with each pixel being a value from zero to 2 to the power of (N-1), with N as the depth (Z-axis) of the pixmap.

**plane**. When a pixmap or window is thought of as a stack of bitmaps, each bitmap is called a plane or bit plane.

**pointer**. The device attached to the cursor and tracked on the screen.

**property**. The name, type, data format, and data associated with a window. By using properties, clients and a window manager share information, such as re-size hints, program names, and icon formats. It is a general-purpose naming mechanism for clients. The protocol does not interpret properties.

**pseudocolor**. A class of colormap in which a pixel value indexes the colormap entry to produce independent red, green, and blue intensity values. That is, the colormap is viewed as an array of triples (RGB values). The RGB values can be changed dynamically.

**resource**. (1) Items such as windows, pixmaps, cursors, fonts, and colormaps are known as resources. Each has a unique identifier associated with it for naming purposes. The lifetime of a resource is bounded by the lifetime of the connection over which the resource was created. (2) A named piece of data in a widget that can be set by a client, by an application, or by user defaults.

**RGB**. Color coding in which the brightness of the additive primary colors of light, red, green, and blue, are specified as three distinct values of white light.

**RGB value**. Red, green, and blue intensity values are used to define a color. These values are always represented as 16-bit unsigned numbers with zero, the minimum intensity, and 65535, the maximum intensity. The X server scales these values to match the display hardware.

**root window**. Each screen has a root window covering it. This window cannot be reconfigured or unmapped, but otherwise performs like any other window.

**SBCS**. (Single Byte Character Set) A set of characters in which each character is represented in one byte of storage.

**screen**. A server can provide several independent screens that typically have physically independent monitors (display screens). This is the expected configuration when there is only a single keyboard and pointer shared among the screens.

**server**. Provides the basic windowing mechanism. It handles IPC connections from clients, de-multiplexes graphics requests onto screens, and multiplexes input back to clients.

**widget**. An object providing a user interface abstraction; for example, a Scrollbar widget. It is the combination of an X Windows window and its associated semantics. Logically, it is a rectangle with associated input and output semantics, although some can be input-only or output-only.

**window**. A region on the server display created by a client application. Windows can be manipulated by using a window manager.

**window manager**. A client that allows the user to move, re-size, circulate and iconize windows on the display.

**X Windows Toolkit**. A collection of basic functions for developing a variety of application environments. Toolkit functions manage Toolkit initialization, widgets, memory, events, geometry, input focus, selections, resources, translation of events, pixmaps, and errors.

# Index

# ITSO Technical Bulletin Evaluation RED000

**TCP/IP for MVS, VM, OS/2 and DOS**
**X Window System Guide**

**Publication No. GG24-3911-01**

Your feedback is very important to help us maintain the quality of ITSO Bulletins. **Please fill out this questionnaire and return it using one of the following methods:**

- Mail it to the address on the back (postage paid in U.S. only)
- Give it to an IBM marketing representative for mailing
- Fax it to: Your International Access Code + 1 914 432 8246
- Send a note to REDBOOK@VNET.IBM.COM

**Please rate on a scale of 1 to 5 the subjects below.**
**(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)**

**Overall Satisfaction** ____

| | | | |
|---|---|---|---|
| Organization of the book | ____ | Grammar/punctuation/spelling | ____ |
| Accuracy of the information | ____ | Ease of reading and understanding | ____ |
| Relevance of the information | ____ | Ease of finding information | ____ |
| Completeness of the information | ____ | Level of technical detail | ____ |
| Value of illustrations | ____ | Print quality | ____ |

**Please answer the following questions:**

a) If you are an employee of IBM or its subsidiaries:

Do you provide billable services for 20% or more of your time? Yes____ No____

Are you in a Services Organization? Yes____ No____

b) Are you working in the USA? Yes____ No____

c) Was the Bulletin published in time for your needs? Yes____ No____

d) Did this Bulletin meet your needs? Yes____ No____

If no, please explain:

What other topics would you like to see in this Bulletin?

What other Technical Bulletins would you like to see published?

**Comments/Suggestions: ( THANK YOU FOR YOUR FEEDBACK! )**

Name

Address

Company or Organization

Phone No.

**ITSO Technical Bulletin Evaluation** **RED000**
GG24-3911-01

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

**BUSINESS REPLY MAIL**

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM International Technical Support Organization
Department 985, Building 657
P.O. BOX 12195
RESEARCH TRIANGLE PARK NC
USA 27709-2195

Fold and Tape **Please do not staple** Fold and Tape

GG24-3911-01

Cut or Fold
Along Line

Printed in U.S.A.

GG24-3911-01