SCO

SCO OpenServer™ Graphical Environment Guide

SCO OpenServer™

SCO OpenServer™
Graphical Environment Guide

Document Version: 5.0
1 May 1995

The SCO software that accompanies this publication is commercial computer software and, together with any related documentation, is subject to the restrictions on US Government use as set forth below. If this procurement is for a DOD agency, the following DFAR Restricted Rights Legend applies:

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013. Contractor/Manufacturer is The Santa Cruz Operation, Inc., 400 Encinal Street, Santa Cruz, CA 95060.

If this procurement is for a civilian government agency, this FAR Restricted Rights Legend applies:

RESTRICTED RIGHTS LEGEND: This computer software is submitted with restricted rights under Government Contract No. _____ (and Subcontract No. _____, if appropriate). It may not be used, reproduced, or disclosed by the Government except as provided in paragraph (g)(3)(i) of FAR Clause 52.227-14 alt III or as otherwise expressly stated in the contract. Contractor/Manufacturer is The Santa Cruz Operation, Inc., 400 Encinal Street, Santa Cruz, CA 95060.

About this book 1

Chapter 1
Overview of the Graphical Environment 11

Chapter 2
Configuring the Graphical Environment from the Desktop 23

Chapter 3

Customizing startup of the Graphical Environment 43

Chapter 4
Running remote programs 65

Chapter 5
Understanding resources 79

Chapter 6

Changing colors 99

Chapter 7

Changing fonts 125

About this book

This book provides the information you need to customize and administer SCO OpenServer™ Graphical Environment sessions. It includes information on configuring the X Window System™ server, the SCO® Panner™ window manager, the Desktop, and other X clients.

You will find the information you need more quickly if you are familiar with:

- "How this book is organized" (this page)
- "Related documentation" (page 5)
- "Typographical conventions" (page 8)

Although we try to present information in the most useful way, you are the ultimate judge of how well we succeed. Please let us know how we can improve this book (page 9).

How this book is organized

This section describes the chapters presented in this book, as well as information on how the chapters are structured.

This book contains the following chapters:

- Chapter 1, "Overview of the Graphical Environment" (page 11) presents an introduction to software that composes the SCO OpenServer Graphical Environment and how you can customize it. It also provides an overview of many of the files you will use to configure the Graphical Environment.
- Chapter 2, "Configuring the Graphical Environment from the Desktop" (page 23) discusses the various aspects of the Graphical Environment you can configure directly from the Desktop, including colors, fonts, and window background patterns.

- Chapter 3, "Customizing startup of the Graphical Environment" (page 43) explains how to use **scologin**, the **startx** script, and **scosession** to manage your X server sessions. It also includes instructions for configuring **scologin** to manage multiple displays, including X terminals, and information on how to start a Graphical Environment session on an X terminal.
- Chapter 4, "Running remote programs" (page 65) explains how to run remote clients during an Graphical Environment session. This chapter includes information on X security issues related to remote clients accessing your display.
- Chapter 5, "Understanding resources" (page 79) provides an overview of some basic concepts in customizing the appearance and behavior of clients, including how to define resource specifications and how to use the resource database. This chapter is a useful reference for other chapters in this guide that discuss the specifics of setting different types of resources.
- Chapter 6, "Changing colors" (page 99) discusses how to change the colors using resources and how to use the **scocolor** client for administrative purposes.
- Chapter 7, "Changing fonts" (page 125) discusses how to specify the fonts that are used during Graphical Environment sessions, including how to set font resources and how to define font aliases.
- Chapter 8, "Configuring window size and location" (page 159) covers how to specify a window's geometry, including the size and location of the Desktop if you choose not to use it as your Root window.
- Chapter 9, "Changing cursor appearance" (page 173) explains how to change the cursor appearance on the Desktop, in **scoterm** windows, and on the Root window.
- Chapter 10, "Configuring mouse behavior" (page 195) describes how to modify the functionality of your mouse so it accommodates left- or right-handed use. This chapter also discusses modifying the rate of speed at which the mouse cursor moves on your screen and the time allowed between clicks when a user performs a double-click operation.
- Chapter 11, "Configuring the keyboard for the server" (page 209) provides information on how to configure the X server to accommodate different keyboards and how to modify the keyboard layout to suit personal tastes.
- Chapter 12, "Customizing the window manager" (page 219) provides an overview of the window manager configuration file, including a list of the window manager functions that you can specify in this file, and describes how to switch the SCO Panner window manager from the default **pmwm** mode to **mwm**. This chapter is useful as a reference when using the following chapters that cover specific aspects of window manager configuration.

- Chapter 13, "Customizing window manager menus" (page 235) explains how to create and modify window manager menus, including the **Window** and **Root** menus.
- Chapter 14, "Configuring window manager button bindings" (page 253) describes how to customize the results of pressing various mouse button combinations in different contexts.
- Chapter 15, "Configuring window manager key bindings" (page 269) describes how to customize the results of pressing a key or sequence of keys in various contexts.
- Chapter 16, "Customizing the Desktop with rules" (page 285) provides an overview of the rules and rule files that control the behavior of the Desktop.
- Chapter 17, "Using Desktop modules" (page 301) describes how to customize the Desktop for all users without having to edit the system rule file.
- Chapter 18, "Defining Desktop user types" (page 305) describes how to customize the Desktop for groups of users.
- Chapter 19, "Defining Desktop triggers" (page 309) describes how to perform an action specific to a particular icon or window.
- Chapter 20, "Creating objects for the Desktop" (page 315) explains how to create objects for accessing applications from the Desktop, either using the object builder (**objbld**) or manually. On the surface, objects are impossible to distinguish from Desktop icons, but the actions performed by an object are implemented differently than they are for icons.
- Chapter 21, "Configuring icons" (page 327) describes how to configure the actions and appearance of Desktop icons, using icon rules.
- Chapter 22, "Configuring Desktop windows" (page 333) describes how to configure the appearance and behavior of the main Desktop and other desktop windows.
- Chapter 23, "Configuring directory windows" (page 337) describes how to configure the characteristics of directory windows, including the results of dropping an icon in a directory window.
- Chapter 24, "Configuring Desktop menus" (page 341) describes how to create and modify desktop and directory menus.
- Chapter 25, "Writing Deskshell commands" (page 349) describes the Deskshell script language, including the conventions and syntax for coding the scripts. These scripts are used in the various rule files and object scripts.
- Chapter 26, "Mapping mouse triggers for the Desktop" (page 371) describes how to configure the actions that result when mouse actions such as clicking and dragging are performed on the Desktop.

- Appendix A, "OSF/Motif window manager resources" (page 377) provides a list and description of the resources you can use to configure the window manager.
- Appendix B, "Desktop resources" (page 403) provides a list and description of the resources you can use to configure the Desktop.
- Appendix C, "Deskshell command summary" (page 415) provides an alphabetical list of all Deskshell commands.

How to use the chapters in this book

Most of the chapters in this book describe how to perform a general task, such as changing colors, or creating new window manager menus. Other chapters are more reference-oriented, providing overviews of how to use major configuration components of the Graphical Environment, such as X resources and Desktop rules.

In general, the task-oriented chapters consist of several sections:

- Background information, such as important concepts and terms, is presented first.
- The next sections cover the procedures that relate to performing the task covered by the chapter. (In many cases, not all of the procedures covered in a chapter are necessary to perform a task.) For example, the chapter that covers the task of changing fonts includes several related procedures, including how to preview available fonts on your system, how to create font aliases, and how to implement a font resource specification for a client.

  The procedure sections are designed to accommodate both new and experienced users. Each procedure begins with a list of steps that provides the essential information for completing the procedure. Following the list of steps are several subsections, one for each step required by the procedure. These subsections provide details on performing the steps, including explanations of command or file syntax. If you are unclear about the goal or desired outcome of performing a particular step, refer to that step's subsection for more information.
- Finally, most chapters provide an example section. These examples describe realistic scenarios and tie together many of the concepts and procedures described throughout the chapter.

It is not intended that you read this manual sequentially. However, it is recommended that you familiarize yourself with the following reference chapters before attempting to configure the Graphical Environment: Chapter 1, "Overview of the Graphical Environment" (page 11); Chapter 5, "Understanding resources" (page 79); Chapter 16, "Customizing the Desktop with rules" (page 285); and Chapter 12, "Customizing the window manager" (page 219). You can use the task-oriented chapters on an as-needed basis.

Related documentation

SCO OpenServer systems include comprehensive documentation. Depending on which SCO OpenServer system you have, the following books are available in online and/or printed form. Access online books by double-clicking on the Desktop **Help** icon. Additional printed versions of the books are also available. The Desktop and most SCO OpenServer programs and utilities are linked to extensive context-sensitive help, which in turn is linked to relevant sections in the online versions of the following books. See "Getting help" in the *SCO OpenServer Handbook.*

NOTE When you upgrade or supplement your SCO OpenServer software, you might also install online documentation that is more current than the printed books that came with the original system. For the most up-to-date information, check the online documentation.

Release Notes
contain important late-breaking information about installation, hardware requirements, and known limitations. The *Release Notes* also highlight the new features added for this release.

SCO OpenServer Handbook
provides the information needed to get your SCO OpenServer system up and running, including installation and configuration instructions, and introductions to the Desktop, online documentation, system administration, and troubleshooting.

Graphical Environment help
provides online context-sensitive help for Calendar, Edit, the Desktop, Help, Mail, Paint, the SCO Panner window manager, and the UNIX® command-line window.

Graphical Environment Reference
contains the manual pages for the X server (section X), the Desktop, and X clients from SCO and MIT (section XC).

Guide to Gateways for LAN Servers
describes how to set up SCO® Gateway for NetWare® and LAN Manager Client software on an SCO OpenServer system to access printers, filesystems, and other services provided by servers running Novell® NetWare® and by servers running LAN Manager over DOS, OS/2®, or UNIX systems. This book contains the manual pages for LAN Manager Client commands (section LMC).

Mail and Messaging Guide
describes how to configure and administer your mail system. Topics include **sendmail,** MMDF, **SCO Shell Mail, mailx,** and the Post Office Protocol (POP) server.

Networking Guide
provides information on configuring and administering TCP/IP, NFS®, and IPX/SPX™ software to provide networked and distributed functionality, including system and network management, applications support, and file, name, and time services.

Networking Reference
contains the command, file, protocol, and utility manual pages for the IPX/SPX (section PADM), NFS (sections NADM, NC, and NF), and TCP/IP (sections ADMN, ADMP, SFF, and TC) networking software.

Operating System Administrator's Reference
contains the manual pages for system administration commands and utilities (section ADM), system file formats (section F), hardware-specific information (section HW), miscellaneous commands (section M), and SCO Visual Tcl™ commands (section TCL).

Operating System Tutorial
provides a basic introduction to the SCO OpenServer operating system. This book can also be used as a refresher course or a quick-reference guide. Each chapter is a self-contained lesson designed to give hands-on experience using the SCO OpenServer operating system.

Operating System User's Guide
provides an introduction to SCO OpenServer command-line utilities, the SCO Shell utilities, working with files and directories, editing files with the **vi** editor, transferring files to disks and tape, using DOS disks and files in the SCO OpenServer environment, managing processes, shell programming, regular expressions, **awk**, and **sed**.

Operating System User's Reference
contains the manual pages for user-accessible operating system commands and utilities (section C).

PC-Interface Guide
describes how to set up PC-Interface™ software on an SCO OpenServer system to provide print, file, and terminal emulation services to computers running PC-Interface client software under DOS or Microsoft® Windows™.

Performance Guide
describes performance tuning for uniprocessor, multiprocessor, and networked systems, including those with TCP/IP, NFS, and X clients. This book discusses how the various subsystems function, possible performance constraints due to hardware limitations, and optimizing system configuration for various uses. Concepts and strategies are illustrated with case studies.

SCO Merge User's Guide
: describes how to use and configure an SCO® Merge™ system. Topics include installing Windows, installing DOS and Windows applications, using DOS with the SCO OpenServer operating system, configuring hardware and software resources, and using SCO Merge in an international environment.

SCO Wabi User's Guide
: describes how to use SCO® Wabi™ software to run Windows 3.1 applications on the SCO OpenServer operating system. Topics include installing the SCO Wabi software, setting up drives, configuring ports, managing printing operations, and installing and running applications.

System Administration Guide
: describes configuration and maintenance of the base operating system, including account, filesystem, printer, backup, security, UUCP, and virtual disk management.

The SCO OpenServer Development System includes extensive documentation of application development issues and tools.

Many other useful publications about SCO systems by independent authors are available from technical bookstores.

For further reading

It is beyond the scope of this book to explain all the details of the industry-standard X Window System and OSF/Motif® window manager software. A number of fine books and articles are available commercially for customers who need more information.

The standard source of information about the X Window System is the *X Window System* set published by O'Reilly & Associates, Inc. The information in this book is particularly related to Volume 3 of the O'Reilly set (*X Window System User's Guide*, OSF/Motif Edition).

Typographical conventions

This publication presents commands, filenames, keystrokes, and other special elements as shown here:

Example:	Used for:
lp or **lp**(C)	commands, device drivers, programs, and utilities (names, icons, or windows); the letter in parentheses indicates the reference manual section in which the command, driver, program, or utility is documented
/new/client.list	files, directories, and desktops (names, icons, or windows)
root	system, network, or user names
filename	placeholders (replace with appropriate name or value)
⟨Esc⟩	keyboard keys
`Exit program?`	system output (prompts, messages)
yes or **yes**	user input
"Description"	field names or column headings (on screen or in database)
Cancel	button names
Edit	menu names
Copy	menu items
File ⇨ Find ⇨ Text	sequences of menus and menu items
open or **open**(S)	library routines, system calls, kernel functions, C keywords; the letter in parentheses indicates the reference manual section in which the file is documented
$HOME	environment or shell variables
SIGHUP	named constants or signals
"adm3a"	data values
employees	database names
orders	database tables
`buf`	C program structures
`b_b.errno`	structure members

How can we improve this book?

What did you find particularly helpful in this book? Are there mistakes in this book? Could it be organized more usefully? Did we leave out information you need or include unnecessary material? If so, please tell us.

To help us implement your suggestions, include relevant details, such as book title, section name, page number, and system component. We would appreciate information on how to contact you in case we need additional explanation.

To contact us, use the card at the back of the *SCO OpenServer Handbook* or write to us at:

Technical Publications
Attn: CFT
The Santa Cruz Operation, Inc.
PO Box 1900
Santa Cruz, California 95061-9969
USA

or e-mail us at:

techpubs@sco.com or ... *uunet!sco!techpubs*

Thank you.

Chapter 1

Overview of the Graphical Environment

This chapter provides an introduction to the various components that work together to create the SCO OpenServer™ Graphical Environment. An overview of this nature is important in understanding how to administer and customize the various aspects of the Graphical Environment.

Understanding servers and clients

Before you begin administering and customizing the Graphical Environment, you should familiarize yourself with the concepts of X servers and X clients.

The "X server" is the software that controls a workstation's or X terminal's hardware, such as its physical display, the keyboard, and mouse or other pointer device. It relays messages between X clients and the hardware on which the clients run. When you use the mouse, keyboard, or a drawing pad to interact with a client, the client decides how to respond to the input (for example, redraw a window, open a menu, and so forth). The client then sends a message to the server, asking for the appropriate action to take place. In response, the server interacts with the system's graphics adapter, whereby the appropriate output is displayed on your screen.

It is the server that actually creates windows and draws images and text in them, in response to requests from client programs. The X server does not initiate actions itself; it only performs actions that are requested by client processes.

The SCO OpenServer Graphical Environment uses the industry-standard X Window System™ server.

An "X client" is a program or application that is written specifically for the X Window System, using X programming tools. These programs are called clients because they act as customers of the X server, asking the server to perform particular actions on their behalf. Clients cannot affect a window or display directly; they can only send a request to the X server to do what they require.

Two major clients of the Graphical Environment are the SCO® Panner™ window manager (an enhanced version of the OSF/Motif® window manager) and the Desktop.

The Graphical Environment uses the client-server architecture because it allows each client to be hardware-independent. This way, clients are more easily ported to different hardware and operating system platforms, and users can access clients that reside on other machines.

Components of the Graphical Environment

The Graphical Environment is created through the combination of several different software components. It is important to understand the relationship between these components before you begin configuring your environment.

The Graphical Environment is comprised of the following components:

- MIT's X Window System, an industry-standard software system that provides the X server for the SCO OpenServer Graphical Environment
- the SCO Panner window manager, an enhanced version of the OSF/Motif window manager, a client that determines the "look and feel" of the windows. The window manager (**pmwm**) mediates communication between the X server and other X clients, managing running programs, and controlling the location and appearance of client windows (particularly window borders), the contents of window menus, and the actions of mouse and key clicks in relationship to the windows.
- the Desktop, a client that provides a graphical interface for running utilities and applications, moving through directories, and manipulating icons. The Desktop (**xdt3**) controls the appearance of desktop icons and directories, the contents of desktop and directory menus, and the effects of mouse clicks and drags on desktop icons, directories, and menus.
- SCO-provided X clients, such as **scomail**, **scocalendar**, **scocolor**, and **scoterm**, that allow you to perform many user and administrative tasks graphically
- additional, industry-standard X clients, such as **xclock**, **xlsfonts**, and **xmodmap**, that provide many other user and administrative functions

When configuring or customizing the system, the parts of the Graphical Environment with which you need to be concerned can be combined into the following categories: X server characteristics, X client characteristics, window manager characteristics, and Desktop characteristics.

See also:

- "Customizing the Graphical Environment" (this page)

Customizing the Graphical Environment

Customizing the Graphical Environment requires that you modify the behavior of one or more of the X clients that work together to create the environment you see when you run the system. As a user, you probably think of the Graphical Environment as a single program, but to customize the environment you must remember that the Graphical Environment is comprised of several different programs, each of which control different aspects of the Graphical Environment's functionality.

The following three major components of the Graphical Environment can be customized:

- **X resources:**
  These resources are data that define appearance and behavior of X clients. These data include definitions for window size, window color, fonts, and so forth. X resources are specified on a client basis, although they can be defined to apply to all clients.

  Because the window manager client manages every client window that runs under the graphical environment, it has an especially large number of resources that you can specify. The Desktop client also has a large number of resources that you can configure.

- **SCO Panner window manager characteristics:**
  In addition to the window manager resources mentioned above, you can customize characteristics such as menus and button actions on windows.

- **Desktop characteristics:**
  In addition to the Desktop resources mentioned above, you can configure Desktop-specific features such as icon behavior, menu content, and mouse actions used on the Desktop. These characteristics are defined in "rule files" that may apply to one directory, one user, or the entire system.

See also:

- "Graphical Environment configuration files" (page 14)

Graphical Environment configuration files

The major components of the Graphical Environment have their own configuration files. These files can be modified to implement the customizations to the Graphical Environment that you desire.

There are at least two versions of each of these configuration files on the system: one that resides in the user's **$HOME** directory and affects only that user, and one that provides default specifications for the entire system. Table 1-1, "Files used to customize the Graphical Environment" summarizes the configuration files that are used by the Graphical Environment.

Table 1-1 Files used to customize the Graphical Environment

System file	User file
X server:	
/usr/lib/X11/sys.startxrc	**$HOME**/.startxrc
X resources for clients:	
/usr/lib/X11/app-defaults/***client***	**$HOME**/.Xdefaults-***hostname***
/usr/lib/X11/sco/startup/***client***	**$HOME**/***client***
Window manager:	
/usr/lib/X11/system.pmwmrc	**$HOME**/.pmwmrc
/usr/lib/X11/system.mwmrc	**$HOME**/.mwmrc
Desktop:	
/usr/lib/X11/IXI/XDesktop/rules/system/xdtsysinfo	**$HOME**/.xdtuserinfo
/usr/lib/X11/IXI/XDesktop/rules/modules/***module***	**$HOME**/.xdt_dir/modules/***module***
/usr/lib/X11/IXI/XDesktop/rules/SCO.user/Main.dt	**$HOME**/Main.dt
/usr/lib/X11/IXI/XDesktop/rules/SCO.user/***ll_TT***.prf	**$HOME**/.xdt_dir/installed.prf
/usr/lib/X11/IXI/XDesktop/rules/SCO.user/Rule.dr	
directory/.xdtdir/***ll_TT***	**$HOME**/***directory***/.xdtdir/***ll_TT***

Note the following:

- ***client*** is the class name of the X client to which the resource specifications in the file apply
- ***hostname*** is the name of the system on which the client is running
- ***module*** is the name of the module file, and can include "auto" modules, with the *.auto* suffix, and "loop" modules, prefixed with *Loop_*

- *ll_TT* represents language and locale, where *ll* is a two-character code for the language (as defined by the ISO 639 standard) and *TT* is a two-character code for the territory (as defined by the ISO 3166 standard). By default, "en_US" is used.

- *directory* is the name of the directory in which the rule file exists.

- the desktop defines the **$XDTHOME** environment variable to be */usr/lib/X11/IXI/XDesktop* and the **$XDTUSERHOME** environment variable to be **$HOME/***.xdt_dir*. You can use these variables to aid in locating the desktop files listed in this table.

See also:

- "The .startxrc file" (this page)
- "The .Xdefaults-hostname file" (page 16)
- "The pmwmrc and .mwmrc files" (page 16)
- "Desktop rule files" (page 17)

The .startxrc file

The **startx** runtime configuration file specifies the clients that the X server should automatically run when a Graphical Environment session is started.

The system-wide version of this file is named *sys.startxrc* and is located in */usr/lib/X11*. The user's version of this file is **$HOME/***.startxrc*.

The *.startxrc* file is not placed in a user's home directory by default. If you want your own personal copy of this file, copy it from the system-wide file and rename it appropriately.

Because the Graphical Environment uses the **scosession** client to manage all user sessions, **scosession** is the only client that is invoked by the default system *sys.startxrc* file. See Chapter 3, "Customizing startup of the Graphical Environment" (page 43), for more information on this configuration file.

NOTE It is strongly recommended that you use the **scosession** client to configure the clients you want to run during a Graphical Environment session, instead of modifying the *.startxrc* file. The **scosession** client is responsible for starting all of the X daemon processes. For more information, see Chapter 3, "Customizing startup of the Graphical Environment" (page 43).

The .Xdefaults-hostname file

The *.Xdefaults-**hostname*** file, located in **$HOME**, defines resource specifications for clients invoked by an individual user. ***hostname*** refers to the name of the machine on which the clients are running.

If you run clients on multiple hosts, you need to create an *.Xdefaults-**hostname*** file on each host.

The **$HOME**/*.Xdefaults-**hostname*** file is not created by default. If you want to specify your own individual resources, you must first create this file.

The *.Xdefaults-**hostname*** file overrides default resource specifications that are defined in client-specific resource files, located in */usr/lib/X11/app-defaults*. These files take the name ***client***, where the client's class name is used. For example, the *ScoHelp* file contains resource specifications for the **scohelp** client, and the *ScoTerm* file contains resource specifications for the **scoterm** client.

Finally, individual users can create client-specific resource files that only control the clients they run without affecting other users on the system. As in the */usr/lib/X11/app-defaults* directory, these files use a client's class name as their filename; however, they are located in the user's home directory.

For example, if you want to define several unique Desktop resources and store them in your own personal client-specific resource file, you would name the file **$HOME**/*XDesktop3*.

See also:

- See Chapter 5, "Understanding resources" (page 79) for more information on the *.Xdefaults-**hostname*** file, client-specific resource files, and specifying resources

The pmwmrc and .mwmrc files

The contents and functionality of window manager menus (including the **Window** menu and the **Root** window), the behavior of the window manager when you press a mouse button with the pointer focused in a window or somewhere on a window border, and the keyboard keys that act as "accelerator keys" and what happens when you press an accelerator key are all defined in the window manager configuration file.

The name of the window manager configuration file depends on the mode of the SCO Panner window manager that you are using: the default **pmwm** mode, which provides the enhanced functionality of the panner, or **mwm** mode, which provides standard OSF/Motif functionality.

The system-wide version of the configuration file is */usr/lib/X11/system.pmwmrc* (for **pmwm** mode) and */usr/lib/X11/system.mwmrc* (for **mwm** mode). The user's version of this file is **$HOME/***.pmwmrc* or **$HOME/***.mwmrc*.

The local configuration file is not placed in a user's home directory by default. If you want your own personal copy of this file, copy it from the system-wide file and rename it appropriately.

NOTE If a window manager configuration file exists in a user's home directory, it completely replaces the system file; the window manager never reads the system file after it locates a local configuration file. Make sure you copy the entire system file to your home directory, or you will lose critical window manager functionality.

See also:

- Chapter 12, "Customizing the window manager" (page 219)

Desktop rule files

Rule files (including modules and user types) define characteristics of the Desktop beyond those defined with resource specifications. Depending on its location, a rule file may apply to files in a specific directory, to operations done by an individual user, or to any operation on the system that is not defined elsewhere.

Rather than create one rule file that is always in effect, a user can create separate rule files for different environments. For instance, a user might want to define one environment for programming, another for text-editing sessions, and yet another for administrative tasks such as reading mail and scheduling the calendar. Another user might want one environment for sending invoices, one for recording receipts, and another for generating summary reports. Each environment could display different data and executable files, use different icons for the files, include different desktop and directory menus, or define different actions for double-clicking or dragging a mouse button.

See also:

- Chapter 16, "Customizing the Desktop with rules" (page 285)
- Chapter 17, "Using Desktop modules" (page 301)
- Chapter 18, "Defining Desktop user types" (page 305)

Guidelines for configuring the Graphical Environment

You should observe the following guidelines whenever you modify the Graphical Environment configuration:

- It is recommended that you first copy the original version of a configuration file to another name (for example, *.pmwmrc.old*). This gives you an easily accessible backup copy that can be restored if you make a serious error or decide you want to return to the default configuration.

- Superuser (*root*) privileges are necessary to modify system configuration files. Users, however, can modify their personal configuration files that are located in their home directories.

- All Graphical Environment configuration files provide at least one version of the file that can be customized by users and another version of the file that defines the defaults for the system.

- It is a matter of local policy whether the administrator modifies users' configuration files or whether users are responsible for maintaining their own files. If the administrator is responsible for maintaining many users' files and wants to keep them all identical, it may be useful to link files with the standard UNIX® system **ln**(C) command. Linking configuration files allows the administrator to edit one file and effectively change the contents of several users' files. If you link configuration files, change the permissions so that users cannot edit them. However, if individual users are customizing their own environments, configuration files should not be linked.

- The names that are used for the various configuration files can be changed through resources, specified in an *.Xdefaults-**hostname*** file. This guide refers to the default pathnames and filenames for these files. It is recommended that you do not change these names.

Looking at the Graphical Environment

To clarify which configuration files you should use to customize the various aspects of the Graphical Environment, it is useful to look at an active Graphical Environment session. Figure 1-1, "Configurable Graphical Environment characteristics - View 1" (this page) and Figure 1-2, "Configurable Graphical Environment characteristics - View 2" (page 21) show possible screen displays, with annotations that discuss each characteristic and how it is customized.

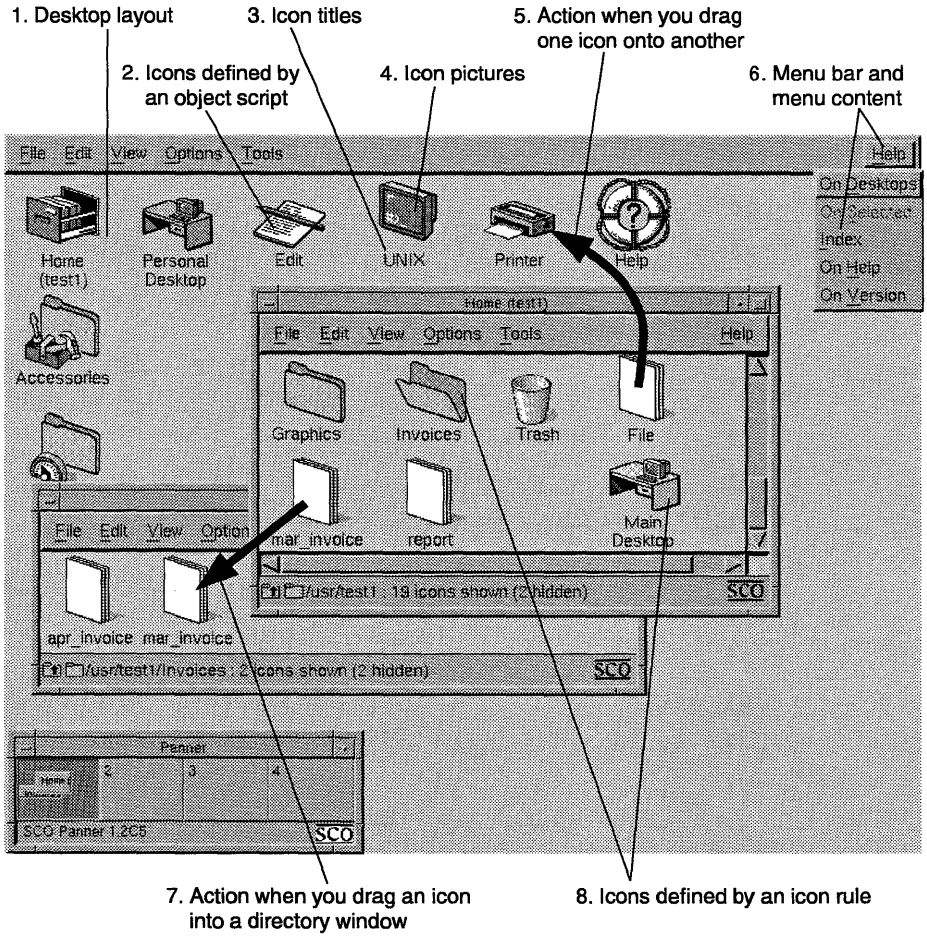

Figure 1-1 Configurable Graphical Environment characteristics - View 1

Figure 1-1, "Configurable Graphical Environment characteristics - View 1" (page 19) illustrates a number of configurable characteristics, most of which are controlled by the Desktop's various rules.

- The **desktop_layout** section of a rule file designates the icons that are displayed on the Desktop and the order in which they are displayed (**#1**). The **locked_on_desktop** section of a rule file specifies icons that cannot be removed from the Desktop, such as the icon for your home directory. See Chapter 16, "Customizing the Desktop with rules" (page 285) and Chapter 22, "Configuring Desktop windows" (page 333) for more information.

- Objects (**#2**) determine the pictures that are used for object icons (**#4**), the label displayed for the icon (**#3**), the action when you click or double-click on an icon, and the action when you drag one icon onto another (**#5**). See Chapter 20, "Creating objects for the Desktop" (page 315) for more information.

  Files and directories on your system are represented through icons that are created with **icon_rules** (**#8**). Like objects, the **icon_rules** section of a rule file determines an icon's picture, title, and behavior. See Chapter 21, "Configuring icons" (page 327) for more information.

- You can also use **icon_rules** in rule files to define the action when you drag an icon into a directory window (**#7**). See Chapter 23, "Configuring directory windows" (page 337) for more information.

- The **menu** section of a rule file defines the contents of desktop and directory menu bars and the contents of the menus on these menu bars (**#6**). Note that window manager menus are defined in the window manager configuration file rather than in rule files; see Figure 1-2, "Configurable Graphical Environment characteristics - View 2" (page 21). See Chapter 24, "Configuring Desktop menus" (page 341) for more information.

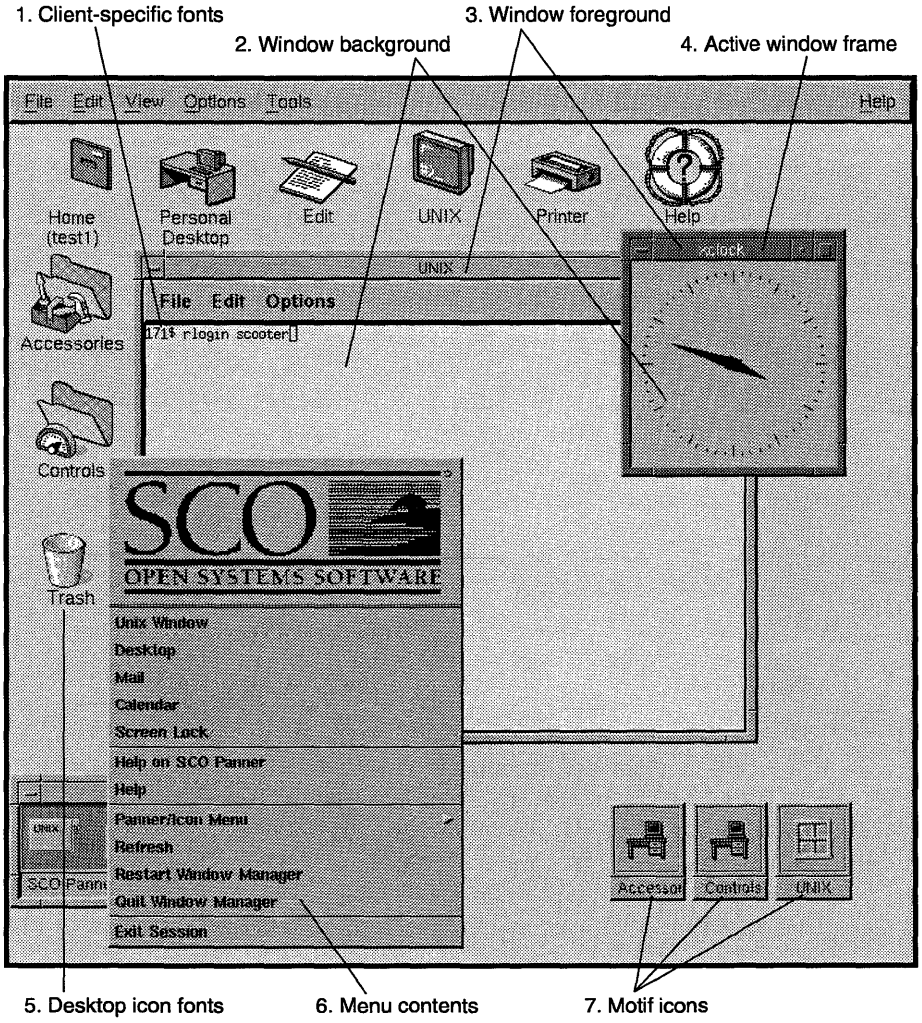

Figure 1-2 Configurable Graphical Environment characteristics - View 2

Figure 1-2, "Configurable Graphical Environment characteristics - View 2" (page 21) illustrates configurable characteristics that are controlled by the window manager or by X resources.

- The color of active window frames (**#4**), and of window foregrounds (**#3**) and window backgrounds (**#2**) are controlled by X resources, specified in system-wide resource files or in local *.Xdefaults-**hostname*** files. See Chapter 6, "Changing colors" (page 99) for more information.
- The font used for display text for clients is also controlled by X resources (**#1**). See Chapter 7, "Changing fonts" (page 125), for more information.
- The font used to display icon titles is defined by a Desktop resource (**#5**). See Appendix B, "Desktop resources" (page 403) for more information.
- The window manager configuration file defines the contents of window manager menus (**#6**). Local window manager configuration files can override these menus. Note that Desktop menus are defined in rule files rather than in the window manager configuration file; see Figure 1-1, "Configurable Graphical Environment characteristics - View 1" (page 19). See Chapter 13, "Customizing window manager menus" (page 235) for more information.
- When a client is iconified, the picture that is used for the client's window manager icon is defined either by the client or by the window manager, depending on X resource specifications you make (**#7**). See Appendix A, "OSF/Motif window manager resources" (page 377) for more information.
- Keyboard actions and mouse actions are controlled by X resources and the window manager configuration file. For more information, see: Chapter 14, "Configuring window manager button bindings" (page 253), Chapter 15, "Configuring window manager key bindings" (page 269), Chapter 11, "Configuring the keyboard for the server" (page 209). and Chapter 26, "Mapping mouse triggers for the Desktop" (page 371),

Chapter 2

Configuring the Graphical Environment from the Desktop

Many of the customizations that you may want to make to the Graphical Environment can be done graphically, using various tools provided in the **Preferences Editor**.

Specifically, this chapter describes:

Using the Preferences Editor

You can easily tailor the Graphical Environment to suit your needs using the **Preferences Editor**. To run the **Preferences Editor**, double-click on the **Preferences Editor** icon, located in the *Controls* window.

In the **Preferences Editor** window, double-click on the icon that corresponds to the part of the Graphical Environment's appearance or behavior that you want to configure.

NOTE If you only want to change aspects of a particular desktop or directory window, use the **Options** menu in that window's menu bar.

Some of the icons in the **Preferences Editor** window launch applications that control various aspects of the Graphical Environment, while others display dialog boxes from which you can make configuration selections.

See also:

- "Using the Preferences Editor dialog boxes" (this page)
- "Preference categories" (this page)
- "Using the Preferences Library" (page 25)

Using the Preferences Editor dialog boxes

There are several kinds of **Preferences Editor** dialog boxes, including:

- the Font Selector, for changing fonts in desktop and directory windows
- the Pattern Selector, for changing the background in desktop and directory windows
- list boxes, which provide alternatives for preferences. Click the down arrow to display the list of alternatives and select the option you want.
- text or number entry boxes, into which you can enter the value you want

Preference categories

You can configure the following aspects of the Graphical Environment with the **Preferences Editor**:

- how you start and exit (page 26) the Graphical Environment
- the colors (page 27) used through the Graphical Environment

- the fonts (page 30) and background patterns (page 31) used for specific desktop and directory windows, including the main Desktop and the Trash and Treeview desktops, and for icons, dialog boxes, and pop-up menus
- mouse characteristics, (page 33) including the double-click rate, speed of the pointer, and whether the mouse is used right- or left-handed
- the key click volume and auto repeat for your keyboard (page 35)
- the pitch and duration of your system bell (page 35)
- who has access to your display (page 36)
- Main Desktop behavior, (page 37) including how icons are displayed and whether or not the Desktop occupies the entire background
- Treeview desktop behavior, (page 37) including the display depth and the expand rate
- desktop window behavior, (page 37) including how icons are displayed and whether or not changes are automatically saved upon exit
- directory window behavior, (page 37) including how files are sorted and displayed, and whether or not existing windows are reused
- icon appearance and behavior, (page 37) including image size
- dialog box appearance and behavior, (page 37) including positioning on the screen
- the applications that are associated with various tools, (page 41) such as your default text editor, mail reader, and calendar
- the devices (page 41) that are associated with the icons in the *Devices* window

Using the Preferences Library

The preference settings of the default Graphical Environment configuration are stored in a file called *installed.prf*, located in the *.xdt_dir* directory in your home directory. This file is updated whenever you change your preferences.

If you want to store several different sets of preferences, you can use the **Preferences Library**. Double-click on the **Preferences Library** icon in the *Controls* window. Select **Archive Installed Preference As** from the **Preferences Library File** menu.

To use a particular set of preferences, you simply double-click on its icon in the **Preferences Library** window.

The following preference files are supplied in the **Preferences Library**, to give you some examples of the various changes you can make to your environment:

SCO	SCO user preferences
Small_icons	small icons are used
Large_icons	large icons are used

Changing how you start and exit the Graphical Environment

To change how you start and exit the Graphical Environment, double-click on the **Session** icon in the **Preferences Editor**, located in the *Controls* window.

You can continue working with the Desktop layout and programs in the same state as your last session by selecting:

```
Resume previous session
```

(This selection restores only the Desktop layout and running programs; it does not reopen the direcory windows, desktop windows or Treeview windows that were open when you logged out.)

Or, you can start each session with the default Desktop appearance by selecting:

```
Start a new session (my desktop)
```

To change the default appearance, first configure your Graphical Environment as desired. For example, you may want to have multiple desktops open automatically each time you log in, with some clients iconified, and special programs running. Then select:

```
Save current configuration
```

You also have the option of deciding each time you log in whether you want to use the default Desktop or the Desktop as you left it by selecting:

```
Ask each time
```

If you want to avoid logging out accidentally, select:

```
Confirm that I want to log out
```

This selection means that when you select **Exit** from the Desktop **File** menu, you will need to respond to a dialog box asking you to confirm your selection before you can completely exit.

After you have made your **Session** selections, click on **OK**.

See also:

- **scosession**(XC) manual page
- Chapter 3, "Customizing startup of the Graphical Environment" (page 43)

Changing colors with the Color control

The **Color** control lets you change the color scheme (palette) used in the Graphical Environment. The colors in the selected palette are assigned to window components, including backgrounds, foregrounds, text, and frame shadows. Palettes can be selected from a list, created, edited, and deleted. The colors that make up a palette can be selected from a list or mixed using one of two color mixing models.

To run the **Color** control, double-click on the **Color** icon in the **Preferences Editor**, located in the *Control* window.

NOTE The **Color** control requires a display that supports at least 16 colors or grayscales. If you try to run **Color** on a monochrome system, you see an error message.

To select one of the provided palettes, select it from the list. If you like the new color scheme, click on **OK**.

To exit Color without making any changes, click on **Cancel**. You are asked to verify that you really want to discard your changes.

See also:

- "Creating a new palette" (page 28)
- "Changing colors in a palette" (page 28)
- "Deleting a palette" (page 28)
- "Colors for DOS programs" (page 30)
- "Colors for grayscale monitors" (page 30)

See Chapter 6, "Changing colors" (page 99) and the **scocolor**(XC) manual page for information on how programs in the Graphical Environment support palettes, on making color changes on a client-by-client basis, and on setting colors from the UNIX command line.

Creating a new palette

In the **Color** window, click on **Add palette**. You are asked to name the new palette. (If the name field is not highlighted, click on it before typing.)

The new palette inherits the colors of the current palette. Change the colors in your new palette as desired.

See also:

- "Changing colors in a palette" (this page)
- "Deleting a palette" (this page)

Deleting a palette

In the **Color** window, select a palette from the list, then click on **Delete palette**.

If you accidentally delete a palette, click on **Cancel**. The palette is not actually deleted until you click on **OK**.

Only user-defined palettes can be deleted. The **Delete palette** button is disabled when one of the palettes supplied with your SCO system is selected.

Changing colors in a palette

In the **Color** window, select a palette name from the list. Then click on one of the color buttons to the right of the list.

In the Color Selection window that pops up, select a color from the list or mix your own color. Click on **Apply** when you have the color you want. The Color Selection window stays up so you can select other color buttons on the **Color** window to modify as well. When you are finished changing colors, click on **OK**.

NOTE The color change for a button will be lost if you click on another color button before clicking on **Apply** in the Color Selection window.

If you try to change the colors in one of the palettes supplied with your SCO system, you are prompted to provide a new palette name. Although you cannot change any of the supplied palettes, the new palette inherits the colors of the supplied palette, along with your changes.

See also:

- "Color buttons" (this page)
- "Mixing colors" (this page)
- "Creating a new palette" (page 28)
- "Deleting a palette" (page 28)

Color buttons

Each color button in the **Color** window represents one of the colors in a palette:

- The "Background" color is used for all windows on the screen, with the exception of the window in which you are working.
- The "Foreground" color is used for text.
- The "Top shadow" color is used for the outline around all window frames.
- The "Active window" and "Active top shadow" colors are used to identify the window you are working in. The "Active foreground" color is used for text in the active window.
- The "Alternate background" color is used for the Desktop background, and for scrollbar and slider bar troughs, and the icon box background, if in use.
- The "Highlight" color outlines buttons when you click on them.

Bottom shadows are always black.

Mixing colors

In the Color Selection window, drag the color mix sliders left or right to compose the color you want, then click on **Apply**.

To get to the Color Selection window, click on one of the color buttons on the main **Color** window.

To change the color mixing model, click on the **Color model** button above the sliders, then click on **RGB** or **HSV** in the pop-up menu.

RGB defines a color by the amount of red, green, and blue it contains. **HSV** defines a color by its hue, saturation, and value.

See also:

- "Creating a new palette" (page 28)
- "Deleting a palette" (page 28)
- "The RGB and HSV color models" (page 101)

Colors for grayscale monitors

Color palettes are automatically mapped to grayscale monitors. Because this might not always yield optimal results, several grayscale palettes are provided.

It is not necessary to select one of the grayscale palettes unless the screen is difficult to read or looks odd.

Colors for DOS programs

Some X servers support 256 colors, others only support 16. If your machine only supports 16 colors, DOS programs running in the SCO Merge window may produce unreadable screens or distorted colors. This does not occur if the SCO Merge window is zoomed to fill the whole screen.

If you encounter this problem, use **Color** to select the "DOS Primary Colors" palette.

See also:

- "Colormaps" (page 107) for an explanation of the different ways SCO Merge and X Windows handle colors

Changing Desktop fonts

You can change the fonts that are used in the Desktop, including desktop and directory windows, dialog boxes, menus, and icons.

To change a font:

1. Double-click on the **Preferences Editor** icon in the *Controls* window.
2. You can change the font used for:
   - the main Desktop
   - desktop and directory windows, including the Trash and Treeview desktops
   - dialog boxes
   - pop-up menus
   - icon labels

   Double-click on the desired icon.
3. Click on the **Font** button in the window that appears. The Font Selector dialog box is displayed.

4. Select a font family by clicking on one of the items in the "Font family" box. The available fonts in that family appear in the "Matching font list" box. Display an example of a font by clicking on it in the "Matching font list" box.
5. To change the font weight, click on one of the buttons under "Weight". Not every font family provides all the weights listed.
6. To change the font angle, click on one of the buttons under "Angle". Not every font family provides all the angles listed.
7. To change the font point size, click on one of the selections under "Point size". Not every font family provides all the points listed.
8. After defining all your font parameters, select a font listed in the "Matching font list" box and click on **OK**.
9. Click on **OK** in the Preferences dialog box to implement the changes.
10. Restart the affected desktop or directory window to see your changes.

See also:

- Chapter 7, "Changing fonts" (page 125)

Changing the background pattern

You can change the background pattern displayed on your screen by selecting bitmap or pixmap files.

- A bitmap is a picture or other graphical image that is stored as a series of 0's and 1's. Each character represents a single dot in the image. A bitmap can include only two colors.
- A pixmap is similar to a bitmap except it can include multiple colors.

See also:

- "Selecting the background pattern" (page 32)
- "Removing background patterns" (page 32)
- "Defining the bitmap/pixmap path" (page 33)

Selecting the background pattern

To specify a background pattern:

1. Double-click on the **Preferences Editor** icon in the *Controls* window.
2. You can change the background for:
   - the main Desktop
   - desktop and directory windows, including the Trash and Treeview desktops
   - dialog boxes

   Double-click on the desired icon.
3. Click on the **Background pattern** button in the window that appears. The Pattern Selector dialog box is displayed.
4. Click on one of the bitmap or pixmap files displayed in the "Patterns" box. Or, open a new bitmap or pixmap directory by double-clicking on a directory listed in the "Directories" box, then clicking on the appropriate filename in the "Patterns" box. An example of the bitmap or pixmap appears as the current selection.

   NOTE You can change the bitmap/pixmap path to specify where bitmaps or pixmaps are stored. For more information, see "Defining the bitmap/pixmap path" (page 33).

5. Click on **OK** to choose the bitmap or pixmap.
6. Click on **OK** in the the Preferences dialog box.

Removing background patterns

To remove a pattern from a window and return to the default background:

1. Double-click on the **Preferences Editor** icon in the *Controls* window.
2. Double-click on the icon for the part of the Desktop from which you want to remove the background pattern.
3. Click on the **Background pattern** button in the window that appears. The Pattern Selector dialog box is displayed.
4. Click on **Remove pattern**.
5. Click on **OK** to choose the bitmap or pixmap.
6. Click on **OK** in the the Preferences dialog box.

Defining the bitmap/pixmap path

The bitmap/pixmap path specifies where bitmap and pixmap files are stored. To specify a path different from the default:

1. Double-click on the **Preferences Editor** icon in the *Controls* window.
2. You can change the background for:
   - the main Desktop
   - desktop and directory windows, including the Trash and Treeview desktops
   - dialog boxes

   Double-click on the desired icon.
3. Click on the **Background pattern** button in the window that appears. The Pattern Selector dialog box is displayed.
4. Type the name of the directory where the desired bitmap or pixmap files are stored in the "Directory path filter" field. At the end of the directory name, include the characters "/*" to ensure that all the files in the directory are listed in the "Patterns" box. Be sure you enter a valid bitmap/pixmap path name.
5. Click on **Filter**. The bitmap or pixmap files in the directory you specified are displayed.
6. Select the desired background pattern. See "Selecting the background pattern" (page 32).

Changing mouse characteristics

With the **Mouse** control, you can specify:

- if the mouse is used right-handed or left-handed
- the speed of double clicks
- the speed at which you can move the pointer

To start the **Mouse** control, double-click on the **Mouse** icon in the **Preferences Editor**, located in the *Controls* window.

Each of the mouse settings are described below:

Right- or left-handed mouse

By default, the left mouse button is mouse button 1 (the select button). This favors right-handed users, so left-handed users may find it easier to change mouse button 1 to the right mouse button. See also Figure 3-1, "Mouse buttons" in *Using the Desktop*.

NOTE All adjustments made to the mouse are effective immediately within the Mouse control window. This lets you try your changes as you make them, but can be confusing if you select **Left Handed** and then do not realize that you must now use the *right* mouse button as the select button instead of the left.

If you click the mouse button and nothing happens, the mouse may be configured for use with the other hand.

Double click speed

You can change the maximum amount of time allowed between the two clicks of a double-click. The slower the double-click setting, the more time is allowed between the two clicks.

You can test your settings on the double-click test pad by selecting and de-selecting it, using double-clicks.

NOTE The newly defined double-click speed does not take effect in windows that are currently open. To implement the new speed, you must restart the Desktop.

Mouse speed

You can change the speed at which you can move the mouse pointer across the screen.

The "Acceleration" slider bar sets the speed to which the pointer accelerates after being moved a short distance. The faster the setting, the faster the pointer can move. A fast setting lets you make short, precise movements but still move quickly when you want to move across the screen.

The "Distance moved before mouse accelerates" slider bar specifies the distance the pointer moves before it accelerates. The longer the setting, the longer it takes the pointer to accelerate.

To exit and save your changes, select **OK**.

To restore your mouse settings to their state before you started modifying them, select **Cancel**.

See also:

- Chapter 10, "Configuring mouse behavior" (page 195)

Configuring the keyboard

To start the **Keyboard** control, double-click on the **Keyboard** icon in the **Preferences Editor**, located in the *Controls* window.

To adjust key click volume, move the slider bar. The number above the slider bar changes to reflect the new volume, which can range from 0 (no sound) to 100 (the loudest). Key click volume can only be controlled on X terminals; this feature is not supported by the SCO X server.

To enable or disable keyboard auto repeat, click on the **Auto Repeat** button. When auto repeat is on, holding down a key causes the corresponding character to repeat on the screen until you release that key. When auto repeat is off, the character appears on the screen only once, regardless of how long the key is held down.

NOTE Changes remain in effect only for the duration of your current session.

See also:

- **xset**(X) manual page.

Changing the system bell

To start the **Bell** control, double-click on the **Bell** icon in the **Preferences Editor**, located in the *Controls* window.

To increase or decrease the bell volume, pitch, or duration, drag the respective slider bar to the left or right. The system bell sounds when the mouse button is released. Bell volume can only be controlled on X terminals; this feature is not supported by the SCO X server.

To hear the current Bell settings, click on the bell icon at the top of the **Bell** window.

To return to the default settings, (volume 50, pitch 40KHz, duration .10 seconds), click on the **Default** button.

NOTE Changes remain in effect only for the duration of your current session.

To change the default slider bar ranges, see **scobell**(XC).

Controlling access to your display

Once the X server is running, the **Host** control allows you to grant or restrict access to your X display. It also displays a list of hosts and users that have access to your current session.

To start the **Host** control, double-click on the **Host** icon in the **Preferences Editor,** located in the *Controls* window.

To grant access to your X server:

- click on **Allow all connections.** All remote users and hosts can display images on your screen.

  or

- click on the "Add host" field. Enter the name of any user or host that you want to be able to access your X server. To grant access to more than one user or host, press ⟨Enter⟩ after each entry you make. When you finish, click on **Apply.**

If a given host does not exist, or if you entered the name incorrectly, **Host** removes the host entry from the Access list and displays:

```
unable to add host hostname
```

To remove host access, select the host in the Access list, and click on **Remove.** To select more than one host, drag the cursor over the desired hosts and then click on **Remove.**

When you are finished, click on **Apply.** To exit, click on **OK.**

To restore your previous settings, click on **Cancel.**

See also:

- Chapter 4, "Running remote programs" (page 65) for more information on configuring access to your display and on running remote clients

Changing desktop, directory, dialog box, and icon behavior

You can customize the behavior of:

- desktops, including the main Desktop, the Treeview desktop, and desktop windows
- directory windows
- dialog boxes
- icons

To modify attributes for any of these parts of the Desktop:

1. Double-click on the **Preferences Editor** icon in the *Controls* window.
2. Double-click on the desired icon in the **Preferences Editor** window and a Preferences dialog box is displayed.
3. Make your desired choices in the list boxes and/or text entry boxes:
   - For list boxes, click the down arrow to display the list of alternatives and select the item you want
   - For text entry boxes, type in the value you want
4. Click on **OK** to implement your changes.

See also:

- "Main Desktop behavior options" (page 38)
- "Desktop window behavior options" (page 38)
- "Treeview desktop behavior options" (page 39)
- "Directory window behavior options" (page 39)
- "Dialog box behavior options" (page 40)
- "Icon behavior options" (page 40)
- "Configuring tools" (page 41)
- "Configuring devices" (page 41)

Main Desktop behavior options

To change aspects of the main Desktop's behavior, double-click on the **Main Desktop Behavior** icon in the **Preferences Editor.**

You can configure the following:

Desktop as root window	sets whether or not the main Desktop occupies the Root window. If set to "True", the Desktop fills the entire screen.
Default desktop display mode	specifies whether icons are displayed as images or names
Desktop opening display method	if you change the way icons are displayed on the main Desktop from the **View** menu, specifies whether future desktops display icons the same way ("Always inherit") or display according to the value set in the "Default desktop display mode" field ("Never inherit")
Desktop traversal mode	determines if opening a new desktop reuses the current window, or displays a new window

See also:

- "Desktop window behavior options" (this page)

Desktop window behavior options

To change the behavior of desktop windows, double-click on the **Desktop Behavior** icon in the **Preferences Editor.**

You can configure the following:

Save on close	sets whether or not you are prompted to save changes made to the desktop window before you close it
Default desktop display mode	specifies whether icons are displayed as images or names

Desktop opening display method	if you change the way icons are displayed in a desktop window from the **View** menu, specifies whether future desktop windows display icons the same way ("Always inherit") or display according to the value set in the "Default desktop display mode" field ("Never inherit")
Desktop traversal mode	determines if opening a new desktop reuses the current window, or displays a new window

See also:

- "Main Desktop behavior options" (page 38)
- "Treeview desktop behavior options" (this page)

Treeview desktop behavior options

To change the behavior of the Treeview desktop, double-click on the **Treeview Behavior** icon in the **Preferences Editor**.

You can configure the following:

Initial display depth	specifies the levels of subdirectories that are displayed
Expand rate	when selecting **Grow Branch** from the Treeview desktop **File** menu, determines the number of levels added to the view

Directory window behavior options

To change the behavior of directory windows, double-click on the **Directory Behavior** icon in the **Preferences Editor**.

You can configure the following:

Directory opening display method	if you change the way icons are displayed in a directory window from the **View** menu, specifies whether future directories display icons the same way ("Always inherit") or display according to the value set in the "Default directory display mode" field ("Never inherit")

Default sort order	specifies the criteria used to sort and display files
Default directory details	defines the information displayed when you select **Details** from the directory **View** menu
Default directory display mode	specifies whether icons are displayed as images or names
Directory traversal mode	determines if opening a new directory reuses the current window, or displays a new window

Dialog box behavior options

To change the behavior of dialog boxes, double-click on the **Dialogs** icon in the **Preferences Editor.**

You can configure the following:

Dialog positioning	specifies whether or not dialog boxes are always centered on your screen when displayed
Report Copy/Move progress	specifies whether or not an **fyi** dialog box is displayed when you perform a move or copy by dragging and dropping icons

You can also define the **Font** used by dialog boxes. For more information, see "Changing Desktop fonts" (page 30).

Icon behavior options

To change the behavior of icons, double-click on **Icon Configuration** in the **Preferences Editor.**

You can configure the following:

Picture areas sensitive	specifies whether you can activate an icon by double-clicking on the entire object, or just on the picture

Drag mode	determines if the entire image moves when you drag an icon, or if a symbolic cursor represents the icon during a drag operation
Title background mode	specifies if icon labels are transparent or opaque
Image size	specifies how you want icons displayed. You can select from several combinations of large or small icons and large or small cursors.
Click on title behavior	allows you to alter the behavior of a single mouse click on an icon label. "Alternate click" refers to the second mouse button.

You can also define the **Icon title fontset** used by icon labels. For more information, see "Changing Desktop fonts" (page 30).

Configuring tools

You can configure many of the applications that are started by various Desktop icons, including the editor, mail reader, and calendar.

To view the tools that can be configured, or to make configuration changes, double-click on the **Tools** icon in the **Preferences Editor**.

Configuring devices

You can assign different physical devices and format commands to the 3.5" and 5.25" device icons in the *Devices* window.

To configure your device icons, double-click on the **Device Configuration** icon in the **Preferences Editor**.

Chapter 3

Customizing startup of the Graphical Environment

This chapter discusses how to customize the startup characteristics of the SCO OpenServer Graphical Environment. Specifically, this chapter covers how to:

- use the display manager (**scologin**) (page 44)
- use the **startx** script (page 46)
- use the session manager (**scosession**) (page 48)
- use environment variables (page 51)
- customize **scologin** to manage multiple servers (page 53)
- use the Graphical Environment on X terminals (page 60)

Starting a Graphical Environment session

By default, the Graphical Environment runs the **scologin** display manager on the second of your console multiscreens (*/dev/tty02*). This display manager starts the X server and keeps it running on your system, even when a user is not engaged in a Graphical Environment session.

However, you can choose to turn the **scologin** client off and start the X server manually, or you can run an additional server session on another multiscreen manually. To run the X server manually, run the **startx** script.

Regardless of the method you use to actually run the X server, a default Graphical Environment session is controlled by the session management client, **scosession**. **scosession** defines the clients that are run when you start the server and controls their appearance and behavior.

See also:

- "Running scologin" (this page)
- "Running the startx script" (page 46)
- "Using the session manager" (page 48).

The above sections assume that you are using clients in their default configuration.

Running scologin

The **scologin** display manager provides graphical login windows to local and remote X servers, as well as services that are similar to those provided by **login** or **getty**. In particular, **scologin**:

- keeps the X server running
- prompts for user login and password
- authenticates users
- requests new passwords when appropriate
- establishes secure Graphical Environment sessions

NOTE See "Customizing scologin" (page 53) for information on modifying **scologin** to manage multiple displays, including X terminals.

The **scologin** client is started as a daemon from the *P86scologin* script in */etc/rc2.d*. By default, **scologin** controls the display on the second multiscreen, */dev/tty02*.

The **scologin** window appears on the screens of all active X servers for which **scologin** is configured to manage. The **scologin** window contains two fields into which you enter your login name and password. The box also contains three buttons: **Login**, **Restart**, and **Help**. To start your session, enter your login and password, then press ⟨Enter⟩ or click on **Login**. To restart the X server and redisplay the **scologin** window, click on **Restart**.

If the login is successful, the following environment variables are set: **$DISPLAY**, **$HOME**, and **$PATH**. If you run the Desktop client, the **$LANG** environment variable is also set. These variables are discussed in "Using environment variables" (page 51).

Once a user is successfully authenticated, several scripts are executed. These scripts are located in */usr/lib/X11/scologin* and are listed in Table 3-1, "scologin session scripts".

Table 3-1 scologin session scripts

Configuration file	Description
Xstartup	a startup script that defines actions **scologin** takes before beginning the user's session
Xsession, Xsession-csh, Xsession-ksh, Xsession-sh	defines the nature of the user's X server session by running the **/usr/bin/startx** script, which starts **scosession**
Xreset	defines the actions that **scologin** takes when the user ends a session

See also:

- "Configuring scologin's startup behavior" (this page)
- "Defining X server sessions" (this page)
- "Logging out of scologin" (page 46)

Configuring scologin's startup behavior

After **scologin** authenticates a user, it executes the startup script, */usr/lib/X11/scologin/Xstartup*.

NOTE This script is run as *root* and as such, should be written with security issues in mind.

This script does not execute any commands by default — it is empty except for a few comment statements. You can place shell commands in the file to perform custom startup tasks, such as mounting users' home directories from file servers, displaying the message of the day, setting custom shell environment variables, and so forth.

Once this script has been executed, **scologin** begins the user's session.

Defining X server sessions

After executing the startup script, **scologin** searches for a script that defines the X server session. First, it looks for a file called *.xsession* in the user's home directory.

If no user-specific file is found, **scologin** looks for */usr/lib/X11/scologin/Xsession-SHELL*, where ***SHELL*** is the user's current shell. For example, a session that is running **csh** would use the *Xsession-csh* file.

The *Xsession* files are started as login shells, which set any environment variables that are specified in the user's *.profile* or *.login* file. Then the *Xsession* files run the **startx -t** script. Basically, **scologin** passes the responsibility for session management to the **startx** script, which then passes control to the **scosession** client. For more information on these next stages of session startup, see "Running the startx script" (this page).

Logging out of scologin

When you end your Graphical Environment session and log out of the system, **scologin** runs a "reset" script, called */usr/lib/X11/scologin/Xreset*. This script executes as *root* and removes the session manager property from the Root window.

You can also use this script to undo the effects of commands that were executed in the *Xstartup* script. For example, the *Xreset* script could unmount directories from a file server that were mounted when the session was started.

When a Graphical Environment session is terminated, **scologin** resets the X server and redisplays the **scologin** window.

Running the startx script

If you want to start a Graphical Environment session from the command line, you must run the **startx** script:

startx &

If you started a session by logging in through the **scologin** window, **scologin**'s *Xsession-SHELL* file also runs the **startx** script, with the **-t** option. See "Defining X server sessions" (page 45) for more information on the *Xsession-SHELL* file.

If the **startx** script is run without any options, it:

- modifies the **$PATH** environment variable to include */usr/bin/X11*, if necessary
- checks to see if the **$DISPLAY** environment variable is set or not. If not, it sets the variable to:

  hostname:display_number

  where ***hostname*** is the name of the current host and ***:display_number*** is the next available display. If no other servers are running, the ***:display_number*** is set to zero.

- runs **xinit**, which starts the X server
- reads the **$HOME**/*.startxrc* file, if it exists, and executes any clients specified in the file. If a *.startxrc* file is not located in the user's home directory, the */usr/lib/X11/sys.startxrc* file is read.

If the **startx** script is executed with the **-t** option, as it is from the **scologin** *Xsession-SHELL* file, the script does all of the tasks above, including modifying the **$PATH** environment variable. However, the **-t** option does *not* set the **$DISPLAY** environment variable or run **xinit** to start the X server. In the case of the **scologin** display manager, it is unnecessary to start the server because it is already running. The **-t** option is also useful if you want to run a Graphical Environment session on an X terminal, which uses its own internal server. For more information on using the Graphical Environment with X terminals, see "Using X terminals" (page 60).

NOTE If you run the **startx** script with the **-t** option, you must set the **$DISPLAY** environment variable before you run **startx**. Otherwise, you see the error message:

```
DISPLAY environment variable not set
```

For information on the **$DISPLAY** environment variable, see "Using environment variables" (page 51).

The */usr/lib/X11/sys.startxrc* file specifies the clients and commands that are run by default in X server sessions for all users on the system. Because the default configuration uses the session manager to control Graphical Environment sessions, **scosession** is the only client that is run by the *sys.startxrc* file. This file contains the following line:

```
exec scosession 2> /dev/null
```

If you want your system to use **scosession** to manage Graphical Environment sessions, you should not modify this file.

The **startx** script also looks for a local *.startxrc* file, located in a user's home directory. If a user wants to use the session manager, there is no need to put a *.startxrc* file in **$HOME**. The *sys.startxrc* file is used to run **scosession**.

If, however, a user does not want to run **scosession**, a *.startxrc* file is needed in **$HOME** to start the desired clients, particularly the window manager. The *.startxrc* file is not placed in a user's home directory by default. To create this file, copy */usr/lib/X11/sys.startxrc* to *.startxrc* in your home directory.

NOTE You are strongly urged to use **scosession** to control the clients you want to run automatically in a Graphical Environment session, instead of adding clients to either **$HOME**/*.startxrc* or */usr/lib/X11/sys.startxrc*. If you do not use the session manager, you may accidentally overlook starting an important element of the Graphical Environment, resulting in the loss of some functionality.

See also:

- **startx**(X) manual page
- "Using the session manager" (this page)

Using the session manager

The session manager client, **scosession**, is responsible for the startup and shutdown of your X server session. Regardless of whether you start your X server through **scologin** or by running **startx** on the command line, the **scosession** client is invoked by the */usr/lib/X11/sys.startxrc* file by default.

scosession uses several files to determine its behavior. These files are located in */usr/lib/X11/sco/ScoSession*, and are listed in Table 3-2, "scosession configuration files".

Table 3-2 scosession configuration files

Configuration file	Description
startup	defines **scosession**'s tasks when a Graphical Environment session is started
static	defines the clients that are run for the default session
shutdown	defines **scosession**'s tasks when a Graphical Environment session is ended
xrdbcomp	compares the system resources loaded by **xrdb** with any resources added to the resource database for the current session and saves the settings so they can be used in future sessions. For information on **xrdb**, see Chapter 5, "Understanding resources" (page 79).

scosession also stores information on individual users' sessions in files in the **$HOME/***.odtpref* directory. These files are listed in Table 3-3, "User scosession files".

Table 3-3 User scosession files

Configuration file	Description
$HOME/.odtpref/ScoSession	this directory contains files related to the management of a user's session
$HOME/.odtpref/ScoSession/dynamic	contains the clients that are saved from a previous session. These clients are started if the user resumes the previous session.
$HOME/.odtpref/ScoSession/static	contains the clients that constitute a user's default session. This file only exists if the user selected to save a session configuration from the **Session** control. (page 26) Otherwise, the default session is derived from the *static* file, located in */usr/lib/X11/sco/ScoSession*.
$HOME/.odtpref/ScoSession/xrdb.save	contains the resource settings from the resource database that existed at the end of a user's session. Resources are stored in this file by the **xrdbcomp** utility. These resources are loaded into the resource database the next time the session is resumed.

The **$HOME/***.odtpref* directory may contain other directories and files, depending on the clients you use and configure.

See also:

- "Starting scosession" (page 50)
- "Stopping scosession" (page 50)
- "Using scosession options" (page 51)
- **scosession**(XC) manual page
- **xrdb**(XC) manual page

Starting scosession

When **scosession** is started, the */usr/lib/X11/sco/ScoSession/startup* script is read. This file sets up your Graphical Environment session. In particular, it:

- loads resources located in files in */usr/lib/X11/sco/startup* into the resource database, using the **xrdb** command. The script also loads resources stored in the file **$HOME**/*.odtpref/ScoSession/xrdb.save*. (For information on resources and the resource database, see Chapter 5, "Understanding resources" (page 79).) These resources reside in the server and determine the basic appearance and behavior of many of the clients you run.
- restores any state information from your previous session, including mouse acceleration, threshold, mouse double-click interval, and left- or right-handed button mapping preferences. This information is determined by files located in **$HOME**/*.odtpref*.
- reads the file **$HOME**/*.odtpref/ScoSession/dynamic* if you resume a previous session or **$HOME**/*.odtpref/ScoSession/static* if you select the default session, and starts all of the specified clients. If neither of these files are located, **scosession** runs the clients indicated in */usr/lib/X11/sco/ScoSession/static*.

  These files indicate not only the clients to run, but any special command line options used to start the applications, their geometry (size and location on the screen), the host machine from which the client can be accessed, and whether or not the client should be run in an iconified or normal state.
- starts the window manager client that is specified by the **ScoSession*windowManager** resource. By default, the SCO Panner window manager, an enhanced version of the OSF/Motif window manager, is run. See Chapter 5, "Understanding resources" (page 79), for more information on resource specifications.

Stopping scosession

When you end your Graphical Environment session and either stop the X server or log out of **scologin**, **scosession** runs the */usr/lib/X11/sco/ScoSession/shutdown* file, which in turn calls */usr/lib/X11/sco/ScoSession/xrdbcomp*. These perform the following:

- Remove the resource database from the RESOURCE_MANAGER property of the Root window. Any resources that you merged into the resource database during the session are stored in the file *xrdb.save*, located in **$HOME**/*.odtpref/ScoSession*. These resources are also loaded into the resource database the next time you run a Graphical Environment session. (For more information on resources and the resource database, see Chapter 5, "Understanding resources" (page 79).)

- Note the state of clients left running when you ended your session and save this information in **$HOME**/*.odtpref/ScoSession/dynamic*. These clients are run in the same state for your next session, if you choose to resume the previous session.
- Save all state information in the appropriate files in **$HOME**/*.odtpref*.
- Shut down all running clients, including the window manager, in a controlled manner.

Using scosession options

You can use the following options with **scosession**:

-stop shuts down the clients comprising the session and saves the state of the session. If you run **scosession -stop** from a **scoterm** window, you are logged out.

-configure configures how **scosession** starts and stops your session. This option brings up a dialog box that allows you to specify if you want subsequent sessions to start in the same state you left your previous session, or if you want to start in the default state. This dialog box also allows users to save the current session's state as a customized default state and to choose the option of an interactive logout prompt.

-help provides a list of the available **scosession** options

See also:

- **scosession**(XC) manual page

Using environment variables

When you start a Graphical Environment session, the **$DISPLAY**, **$HOME**, and **$PATH**. environment variables are set. When you run the Desktop client, the **$LANG** environment variable is also set.

The **$PATH** and **$HOME** environment variables are actually set when you first log in, whether through a multiscreen running **getty** or through **scologin**. However, the X server modifies the **$PATH** variable to include the */usr/bin/X11* directory.

The **$DISPLAY** and **$LANG** environment variables are described below:

- The **$DISPLAY** environment variable is used to tell a client to which server it should send its output.

  The X display consists of one or more screens, a keyboard, and a mouse. A system may have several displays, and each display may, in turn, have more than one screen. Each display has exactly one server process controlling all its input and output. Therefore, the terms "display" and "server" are used synonymously.

  When a client is run, it must open a connection to a display. You must be able to tell the client the name of the display that you want it to use for output. You can also indicate a specific screen for the display. Because the display can be anywhere on the network, you have to provide the network name of the system to which the display is connected to fully identify the display.

  Use the following format when setting the **$DISPLAY** variable:

  [***hostname***]:***display_number***[.***screen_number***]

  where:

  hostname specifies the name of the machine to which the display is connected, and must be either a machine name or the machine's network address. If the ***hostname*** is not specified, the client assumes it should communicate with the display on the same machine.

  :display_number specifies the number of the display, or X server, that you want the client to use. Each display on a system is assigned a ***:display_number***. If the display is managed by **scologin**, the ***:display_number*** is specified explicitly in the */usr/lib/X11/scologin/Xservers* file. If the X server is started by **startx**, the server is assigned the first available ***:display_number***, starting with ":0".

  Usually, if only one X server is running, its ***:display_number*** is ":0". If more than one server is running on your system, you must determine which display number corresponds to the X server you want to specify.

  screen_number specifies the screen on which the server is running.

  The default display name is stored in the **$DISPLAY** environment variable when the X server is started by either **scologin** or the **startx** script. However, if you want a client to use a different display, you must reset the **$DISPLAY** variable so it specifies the other server.

For example, to run your clients on a remote server on a machine named *scooter*, you would enter:

DISPLAY=scooter:0.0; export DISPLAY *(for sh, ksh)*

or

setenv DISPLAY scooter:0.0 *(for csh)*

NOTE Most clients understand the **-display** command line option. This option temporarily overrides the contents of the **$DISPLAY** variable. For more information on using this command line option, see Chapter 5, "Understanding resources" (page 79).

- The **$LANG** environment variable specifies the language that is used on your system. By default, the **$LANG** variable is set to "english_us.ascii".

Customizing scologin

The default configuration of **scologin** runs the X server, and the **scologin** client, on the second multiscreen (*/dev/tty02*) of the console. You can change this configuration so that **scologin** does not run at all, or you can specify that **scologin** manage multiple displays, on your system or on remote systems, including X terminals.

There are several files that are used to configure **scologin**'s behavior. These files are all located in */usr/lib/X11/scologin* and are listed in Table 3-4, "scologin configuration files".

Table 3-4 scologin configuration files

Configuration file	Description
Xconfig	a special configuration file that specifies resources that determine the scripts used by **scologin**. The resources in this file configure the following files.
Xerrors	**scologin** error messages that would otherwise go to standard error (`stderr`) are directed to this file
Xhelp	contains the help text that you see if you click on the **Help** button on the **scologin** window
Xresources	contains resources that configure **scologin**'s appearance. These resources are loaded into the resource database by **xrdb**.
Xservers	contains entries for all of the non-XDMCP X servers that **scologin** is to manage

See also:

- "Using the scologin administration script" (this page)
- "Configuring scologin on multiple displays" (this page)
- "Using X terminals" (page 60)

Using the scologin administration script

The Graphical Environment provides a script, */etc/scologin*, that allows system administrators to control the **scologin** process. The script must be run as *root*.

There are six options that you can use with this script:

start starts the **scologin** process, which in turn reads the files *Xconfig*, *Xservers*, and *Xresources*, all located in */usr/lib/X11/scologin*

stop stops the **scologin** process. Run **scologin stop** to halt all current sessions managed by **scologin**. For example, use the **stop** option if you want to reclaim **scologin**-managed *ttys* and restore **getty** processes.

> **NOTE** This option shuts down *all* **scologin** processes on your system, which results in the closure of any sessions running at the time you run the script. You should notify users before you run this script.

query shows the current state of the **scologin** process

disable stops the current **scologin** process and prevents **scologin** from starting when the system re-boots; re-enables **getty** processes on **scologin**-managed *ttys*

enable ensures that **scologin** starts when the system re-boots and starts the **scologin** process if it is not already running

init if **scologin** is enabled, disables **getty** processes on screens that are configured for **scologin**. **scologin init** should only be run by **init** at boot time.

Configuring scologin on multiple displays

The **scologin** display manager can do more than run the simple session that its default configuration provides. In fact, **scologin** can control multiple servers, both on the local machine and on remote machines, or X terminals.

There are two ways to specify the X servers that you want managed by **scologin**:

- If the server supports the X Consortium standard X Display Manager Control Protocol, also known as XDMCP, you can usually specify the name or network address of a remote machine running **scologin** at the server.

  XDMCP is a dynamic mechanism whereby connections are made when requested by a display, such as a workstation or an X terminal, that can communicate through the protocol. The SCO X server (**Xsco**) supports XDMCP.

- If you want to configure **scologin** to run on a set of console ttys (for example, on *tty01* through *tty12*), or if you want **scologin** to manage an X server that does not support XDMCP, you can add an entry for each of the displays in the */usr/lib/X11/scologin/Xservers* file. Each line in this file specifies a display that should constantly be managed by **scologin**.

See also:

- "About XDMCP X server options" (this page) for a list of the X server options for using XDMCP
- "Running scologin with XDMCP" (page 56) for information on running **scologin** on remote systems using XDMCP
- "Running scologin with the Xservers file" (page 57) for information on manually configuring **scologin** sessions
- "Using X terminals" (page 60) for information on managing an X terminal's display with **scologin**

About XDMCP X server options

Any X server that supports the XDMCP protocol can request a **scologin** session. To do this, the server must be started with the appropriate options to request the session.

The SCO X server (**Xsco**) uses the following options to determine how it uses XDMCP:

-broadcast enables XDMCP and broadcasts BroadcastQuery packets to the network. The first responding display manager is chosen for the session.

-class *display_class* sets the value of the additional XDMCP display qualifier, which is used in resource lookup for display-specific options. By default, the value is "MIT-Unspecified".

-cookie ***xdm-auth-bits***	sets the value of a private key shared between the X server and the manager, which is used when testing XDM-AUTHENTICATION-1
-displayID ***display-id***	allows the display manager to identify each display so that it can locate the shared key
-indirect ***host_name***	enables XDMCP and sends IndirectQuery packets to the specified host
-once	exits the X server after the first session is over. Normally, the X server keeps starting sessions, one after the other.
-port ***port_num***	specifies an alternate port number for XDMCP packets. It must be specified before any **-query**, **-broadcast** or **-indirect** options.
-query ***host-name***	enables XDMCP and sends Query packets to the specified host

See also:

- **Xsco**(X) manual page for a complete list of X server options

Running scologin with XDMCP

To configure the SCO X server to request a **scologin** session using the XDMCP protocol, do one of the following. You must be logged onto the system as *root*.

- Specify the desired **Xsco** options from the command line — see "About XDMCP X server options" (page 55) from the command line. For example:

  /usr/bin/X11/Xsco -broadcast -once

  This broadcasts to all machines on the network for a **scologin** session. The session is provided by the first machine on the network to answer.

  You can also request a session from a specific machine with this command:

  /usr/bin/X11/Xsco -query *hostname* -once

  This requests a session from the host ***hostname***.

- Alternately, you can create a shell script that runs the **Xsco** server, as in one of the examples above.

Running scologin with the Xservers file

You can use the */usr/lib/X11/scologin/Xservers* file to configure **scologin** management of displays on your local system or on X servers that do not support XDMCP. You can also use this approach if you do not want to reconfigure the SCO X server, as described in "Running scologin with XDMCP" (page 56).

To configure **scologin** to manage multiple displays using the *Xservers* file, use the following procedure. You must be logged onto the system as *root*. For information on each of the steps in this list, see the sections immediately following the procedure.

1. Use the **scologin** administration script to stop **scologin**, if it is currently running on your system.

   /etc/scologin stop

2. On the host machine where you want to run **scologin**, add the servers you want to manage to the */usr/lib/X11/scologin/Xservers* file. Use the following format when making entries in this file:

   display_name **[*display_class*]** ***display_type*** **[*startup_command*]**

   When you are finished, save and exit the file.

3. To manage a remote display, you must provide access to the server. On the system where the display is to be managed, edit the */etc/Xn.hosts* file, where ***n*** represents the display number you want to use on the remote machine, and add the name of the machine on which **scologin** will be running.

4. When managing a remote display, you must start the X server on that display before **scologin** can gain control. On the actual screen you want managed by **scologin**, run the X server:

   /usr/bin/X11/X *:display_number*

   On a local system, this step is unnecessary because **scologin** automatically starts the X server.

5. Returning to the **scologin** host machine, use the **scologin** administration script to restart **scologin**, so it reads its configuration files, including *Xservers*:

   /etc/scologin start

The **scologin** display manager should now be running on all of the displays you configured.

Step 1: Stopping existing scologin processes

Before you set up **scologin** to manage multiple displays, you must first stop any **scologin** processes that are currently running. You can do this using the **scologin** administration script:

/etc/scologin stop

NOTE This script shuts down *all* **scologin** processes on your system, which results in the closure of any Graphical Environment sessions running at the time you run the script. You should notify users before you run this script.

Step 2: Editing the Xservers file

For every X server you want **scologin** to manage, you must add an entry to the */usr/lib/X11/scologin/Xservers* file. This file should include entries for additional displays on the local machine and entries for displays on remote machines.

Entries in the *Xservers* file use the following format:

display_name [***display_class***] ***display_type*** [***startup_command***]

The various segments of the format are described below:

display_name name of either a local X server or a remote X display using the following syntax:

[***hostname***]:***display_number***[.***screen_number***]

hostname specifies the name of the machine to which the display is connected. If you omit ***hostname***, the display on the current machine is assumed. ***:display_number*** specifies the number of the display you want to use. ***screen_number*** specifies the number of the screen on the display that you want to use.

display_class defines a display class with which ***display_name*** is associated. Although ***display_class*** is optional, it is useful if you have a large collection of similar displays and want to set **scologin** configuration resources for groups of them. To include several X displays in the same class, use the same ***display_class*** in each *Xservers* entry.

display_type indicates either a local or remote X server:

- if ***display_type*** is "local," **scologin** manages a local display on which an X server should be run
- if ***display_type*** is "foreign," **scologin** manages a remote display *on which the X server is already running*

startup_command applies only to local displays, and by default is **/usr/bin/X11/X**. Use ***startup_command*** to specify command line options to the X server, such as the local *tty* you want **scologin** to manage.

For example, to manage a local display on */dev/tty03* that is not yet running and a display on another SCO system named *scooter*, include the following lines in the */usr/lib/X11/scologin/Xservers* file:

```
1  :0 local /usr/bin/X11/X :0 -crt /dev/tty03
2  scooter:1 foreign
```

In this example, `:0` on line 1 and `scooter:1` on line 2 are the ***display_name***. Also, `local` on line 1 and `foreign` on line 2 are ***display_type***. `/usr/bin/X11/X :0 -crt /dev/tty03` is the ***startup_command***. The **-crt** option associates the X server with a particular console multiscreen, in this case */dev/tty03*.

The `scooter:1 foreign` entry indicates that you want **scologin** to manage the second X server running on the remote machine, *scooter*.

Step 3: Enabling access to the remote display

If you only want to manage the local display, you can skip this step.

If you want **scologin** to manage a remote display that is running the X server, you must enable the server to provide access to your host machine. On the system where the display is to be managed, edit the */etc/Xn.hosts* file, where ***n*** represents the display number you want to use. Add the name of the host machine where **scologin** will be running.

For example, to gain access to the *scooter:1* display, include the name of the **scologin** host machine in the */etc/X1.hosts* file on *scooter*.

Step 4: Running the X server on the remote display

If you only want to manage the local display, you can skip this step.

To enable **scologin** to manage a remote display, you must first start the X server on the desired screen of the display where you want the **scologin** window to appear:

/usr/bin/X11/X *:display_number*

For example, for **scologin** to manage a second X server on the fourth multiscreen on the machine *scooter*, you would log in on *scooter*'s */dev/tty04* and run:

/usr/bin/X11/X :1

Step 5: Starting scologin

Now you are ready to start **scologin** on all of the displays you configured in the *Xservers* file. On the host system, use the **scologin** script to start the client:

/etc/scologin start

This process reads the **scologin** configuration files, including */usr/lib/X11/scologin/Xservers*. A **scologin** process is started for all of the displays specified in the *Xservers* file.

Using X terminals

You can use X terminals to run Graphical Environment sessions. In fact, you can configure your X terminal so the **scologin** display manager automatically manages the X terminal's display. When you log in through the **scologin** window, you start a Graphical Environment session, running on the host machine and displaying on the X terminal.

Many X terminals can use the X Display Manager Control Protocol (XDMCP) to facilitate the connection to remote hosts through **scologin**. From a user's standpoint, the main advantage of XDMCP is that it allows you to turn an X terminal off and instantly re-establish the connection to the **scologin** host machine when you turn the X terminal back on. When you turn on an X terminal, **scologin** automatically displays a login window. The exchange of information between the X terminal and the remote host is invisible to the user. In fact, XDMCP and **scologin** are intended to make X terminals as easy to use as traditional character terminals. With XDMCP, an X terminal basically requests a connection to a remote host, is recognized by the host, and is sent a login prompt by **scologin**.

If you are using X terminals at your site, the way you set up **scologin** depends on whether or not the terminals can communicate through XDMCP. If a terminal cannot communicate using XDMCP, you must include an entry for it in the */usr/lib/X11/scologin/Xservers* file and the terminal must be left powered on at all times to maintain the connection to the host machine.

If an X terminal can communicate through the protocol, the machine that will host the **scologin** process requires no configuration. However, the X terminal must be configured to communicate with the host through the X terminal's setup procedures, which vary from one model to another. Some X terminals let you specify the address of a host machine from which you want to run the display manager. Some X terminals can broadcast a request for a host over the network and then display a list of all available hosts from which the user can choose. Other X terminals can broadcast a request and merely accept the first available host.

See also:

- "Managing an X terminal display with scologin" (this page)
- "Running a session on an X terminal without scologin" (page 63)

Managing an X terminal display with scologin

The following procedures explain how to run the **scologin** display manager so it manages the server on an X terminal.

Once configured, you can log in directly to the machine running **scologin** using the **scologin** window. The **scosession** manager is started, the Desktop appears, and you can begin your session. If an *.Xdefaults-**hostname*** file exists in your home directory on the host machine, clients you run use the resources defined in this file.

These procedures assume that the X terminal has already been correctly connected to the network and that the X terminal's name and IP address are known to the machine that will host the **scologin** process. For information on how to do this, refer to "Configuring TCP/IP" in the *Networking Guide* and to the documentation supplied with your X terminal.

See also:

- "X terminals that do not support XDMCP" (this page)
- "X terminals that support XDMCP" (page 62)

X terminals that do not support XDMCP

If the X terminal **does not** support XDMCP, use this procedure to set up a **scologin** session:

1. Log into the host machine as *root*. If **scologin** is currently running on the host machine, use the **scologin** administration script to stop it:

   /etc/scologin stop

2. On the host machine, edit the */usr/lib/X11/scologin/Xservers* file so that it contains an entry for the X terminal.

   For example, to configure a terminal named *vortex* so it runs **scologin** from a host machine named *scooter*, add the following line to the *Xservers* file on *scooter*:

   vortex:0 foreign

3. Start **scologin** on the host machine by running **/etc/scologin start**.
4. On the X terminal, restart the X server. When the server is running again, the **scologin** window appears on the X terminal screen.

See also:

- "Running scologin with the Xservers file" (page 57)
- "Using the scologin administration script" (page 54)

X terminals that support XDMCP

If the X terminal **does support** XDMCP, use the following procedure. Note that you do not need to configure the *Xservers* file in this situation.

1. Configure the X terminal to use XDMCP. While there are three ways you can do this, the most common method is mentioned first:
   - Set the display manager access parameter to "Direct" and specify the IP address of the machine on which **scologin** will be running. This method transmits a Query packet directly to the specified host machine.
   - Set the terminal's display manager access parameter to "Broadcast." This method broadcasts a query packet, to which one or more hosts may respond. Depending on how your X terminal functions, the display can request management from the host that responds first or it can provide a list of available hosts and allow you to pick one.
   - Set the display manager access parameter to "Indirect." This method transmits a query packet to an intermediate host, which relays it to another host.
2. Verify that **scologin** is running on the host machine with the **/etc/scologin query** command. Enter **/etc/scologin start** to start the display manager if it is not already running.
3. Restart the server session on the X terminal. The **scologin** window displays on the X terminal's screen.

See also:

- The documentation supplied with your X terminal for more information on setting display manager access

Running a session on an X terminal without scologin

You can run a Graphical Environment session on your X terminal without going through the process of configuring **scologin** to manage your X terminal's server. To do so:

1. Start a telnet session on the X terminal. Connect to the host machine on which you want to run your Graphical Environment session.
2. Once you are logged into the host machine, set the **$DISPLAY** environment variable to the X terminal's display. For example, if your X terminal is named *vortex*, you would specify:

   DISPLAY=vortex:0; export DISPLAY

3. Run the **startx** script, using the **-t** option, to start the session:

   startx -t &

4. The **scosession** client is started, the Desktop appears, and you can begin your session.

If an *.Xdefaults-**hostname*** file exists in your home directory on the host machine, clients you run automatically use the resources defined in this file.

Chapter 4

Running remote programs

Networking plays a large part in the SCO OpenServer Graphical Environment; it allows you to mount remote filesystems, and it lets you execute X clients on remote machines while interacting with them from your display. The ability to operate programs over a network does, however, introduce the issue of controlling which clients can access a display. For reasons of security, you may not want every user on every system in your network to have permission to use your X server.

There are three tasks you must perform before you can use remote clients on your display:

- gain access to the remote client (this page)
- set up access permissions to your display (page 66)
- run the client (page 73)

This chapter describes how to do each of these tasks.

Gaining access to the remote client

Before you can run clients on other machines, you must also be able to access the remote machines. If the client is NFS-mounted on your system, then you can run it locally, and no display access permission is required. If, on the other hand, you want the client to run on the remote machine, you need an account on the remote machine to access the remote client by any of the following methods:

- run the client remotely using the **rcmd**(TC) command
- log in to your account on the remote machine with **telnet**(TC) or **rlogin**(TC), then run the client

If you want to run the client via **rcmd**, you must have *user equivalence* on the remote machine or the **rcmd** command returns a "Permission denied" error message.

To establish user equivalence on the remote host:

1. Log in to the remote host machine.
2. Create a file named *.rhosts* in your **$HOME** directory, or if one already exists, open it for editing and add the following line:

   localhost login

   localhost is the name of the machine on which you are running the X server. ***login*** is your account name on ***localhost***.

   When you have finished, save and exit the file.
3. Make sure that the user ID (UID) of ***login*** is the same on both machines. Check the file */etc/passwd* on both machines and make sure the UID fields match. You can also use the **id**(C) command to verify the UIDs on each machine. For information about the structure of the */etc/passwd* file, see the **passwd**(F) manual page.
4. Log out from the remote host.

You can now use **rcmd**(TC) to run clients on the remote host. You can also log in to the remote host without being prompted for your password.

For more details on the above network commands, see *Networking Guide*.

Setting up access permissions to your display

Before you can display a remote client on your X server, you must allow the client to access your display. There are two ways you can control access to your display:

- tell the X server to allow the remote machine to access your display (page 67)
- supply an authorization code to specific accounts on the remote machine (page 68) so only those users can access your display

NOTE For information on controlling access to X terminals, refer to your X terminal documentation.

Each method has advantages and disadvantages. Granting access to a remote host machine is easy to accomplish, but it provides no control over which accounts on the remote host are able to access your display. In systems where security is of greater concern, it is recommended that you use the authorization code method. Be aware that the security of a system that uses the authorization method is only as secure as the user's account; if anyone else can read

the authorization file that contains the authorization code, they can also access your display. Also, the authorization code method only works with X servers that are provided with the Graphical Environment (**Xsco**) or with X terminals that support the XDMCP protocol.

Granting access to specific hosts

The X server maintains a list of machines that have access to the display. You must establish this list even if you plan to only grant access to users with an authorization code.

To grant access to your display for specific host machines, perform the following steps. For more information on each of the steps in this procedure, see the sections immediately following this list.

1. To specify system-wide access permissions, edit the */etc/X**n**.hosts* file, where ***n*** represents the display number that you want the host machine to be able to access. Add the names of the host machines to which you want to grant access. This step is optional. (Do not add hosts to this file if you only want to grant access to your display for one session.) You must be *root* to edit these files.

   The access permissions take effect when the X server is restarted.

2. To specify display access permissions for a single X session, run the server and add or remove host machines in the access permission list with **xhost**(X). Any user can perform this step.

The desired host machines now have permission to access your display.

Step 1: Establishing system-wide host access

The X server maintains a list of machines that have access to your display. This list is contained in the */etc/X**n**.hosts* files on the local machine, where ***n*** represents the display number for which you want to assign access. The machines listed in these files are granted access to displays 0 through 7 at the time the servers are started. For example, to specify the host machines that are allowed to access local display :0, add the names of these machines to */etc/X0.hosts*.

Each line in the *X**n**.hosts* files consists of just the name of the host machine that has access to the server. For example, to allow any user on *boston* to access the *tusconey:0* display, add the following line to */etc/X0.hosts* on *tusconey*:

```
boston
```

You must be logged in as *root* to edit these files. The changes you make take effect when the server is restarted.

Note that any user on a host machine specified in these files has access to your display whenever the X server is running. If you only want to grant access to a host for a single X session, do not modify these files. Instead, see Step 2 (this page).

In addition to adding hosts to the *Xn.hosts* files, be sure to remove hosts that you do not want to have access to your display.

Step 2: Setting temporary display access

Once the X server is running, you can examine the current host permissions with the following command, from a local **scoterm** window:

xhost

Use this command on the local machine where the server is running. **xhost** with no command line options displays a list of machines that currently have access to your display. Use this list to determine if the system-wide configuration provides access to the appropriate host machine.

To add a host to the X server's host access list, execute the following command:

xhost ***+hostname***

If you omit ***hostname***, the X server removes all access restrictions, allowing any client on any machine on the network to access your display.

NOTE Use **xhost** + with caution; it allows any user on any machine on the network to access your display.

If there are hosts configured that you do not want accessing your X server, remove them from the server's host access list by running the following command:

xhost ***-hostname***

This command removes ***hostname*** from the host access list. If you omit ***hostname***, **xhost** specifies that no remote host can access your display for the duration of the current session. This option is very useful to reverse the effect of running **xhost** +.

Granting access to specific accounts

If you log in through **scologin**, you can control display access by using an authorization protocol called MIT-MAGIC-COOKIE. If **scologin**'s **authorize** configuration resource is set, upon logging in, both the X server and the user receive an authorization code called a "magic cookie". If a user attempts to run a client on an X server but does not have the required authorization record, the server denies the client access. For details on configuring **scologin**, see "Customizing scologin" (page 53) and the **scologin**(XC) manual page.

The user receives the magic cookie through an authorization file in the **$HOME** directory, named *.Xauthority*. The authorization file may contain authorization codes for multiple X servers, allowing the user to run clients on these servers. For security, only the user has read or write permissions on authorization files. The user that logged in through **scologin** can share authorization records with other users, however.

NOTE Host access permissions override user authorization restrictions. If the host machine has obtained access permission through one of the */etc/Xn.hosts* files or the **xhost** utility, any user on the host can access the display without having the X server's authorization code. See "Granting access to specific hosts" (page 67) for more information on */etc/Xn.hosts* files and the **xhost** utility.

To grant access permission to a specific user, perform the following steps. For more information on each of these steps, see the sections immediately following this procedure. You must be logged in as *root* to perform this task.

1. Edit the */etc/Xn.hosts* files and remove any host names listed.
2. Configure **scologin** so it runs the X server with the magic cookie authorization protocol. The */usr/lib/X11/scologin/Xconfig* file must contain the following resource specification:

   ```
   DisplayManager*authorize: true
   ```

   When you have finished, restart **scologin**.
3. Log in through the **scologin** window.
4. Run **xhost** to make sure no other host access permissions exist for your session. (For example, host permissions may be specified in **$HOME**/*.startxrc*.) If any host does have access permission, remove the permission with the following command:

   xhost -

5. To make sure your authorization file includes an authorization record for the X server, run the following command:

   xauth list

   Extract the X server's authorization code by running:

   xauth extract ***tempfilename display***

   In this command, ***tempfilename*** is a temporary file in which the authorization code is stored before it is merged. The ***displayname*** is the name of the display as shown by the previous **xauth list** command.

   Finally, merge it with the other user's authorization file by running:

   xauth merge ***tempfilename***

   In this command, ***tempfilename*** is the same temporary file created by the **xauth extract** command.

Step 1: Disabling system-wide display access

The X server only employs its authorization protocol if all host access is disabled. Because the X server obtains a list of authorized hosts each time it starts, make sure the server provides no initial access permissions.

To disable initial host access, edit the */etc/Xn.hosts* file that corresponds to your display. The */etc/Xn.hosts* files determine which host machines have access to the X server, regardless of who starts the server. For example, to remove initial access permissions for local display :0, remove all host names from */etc/X0.hosts*.

Make sure **xhost** is not executed automatically from your **$HOME/***.startxrc* file or from a file in your **$HOME/***.odtpref* directory.

Step 2: Configuring scologin

The X server authorization protocol is only used if you log in through the display manager, **scologin**. **scologin** must be configured to run your X server with the authorization protocol. The authorization protocol is not used if you start the X server by running **startx**(X), **xinit**(X), or **Xsco**(X).

To configure **scologin** for authorization, you must edit the */usr/lib/X11/scologin/Xconfig* file. However, if **scologin** is already running on your system, you must stop it before you edit the *Xconfig* file. As *root*, use the **scologin** administration script to do this:

/etc/scologin stop

NOTE This script shuts down all **scologin** processes on your system, which results in the closure of any Graphical Environment sessions running at that time. You should notify users before you actually stop **scologin**.

Now you can edit the */usr/lib/X11/scologin/Xconfig* file and verify that the **scologin authorize** resource is set to true. If this resource specification is not yet defined, add the following line to the *Xconfig* file:

DisplayManager*authorize: true

or

DisplayManager*_displayname_*authorize: true

In almost all circumstances, the first resource specification is suitable. Only use the latter syntax if you want specific X servers to use the magic cookie protocol. For details on setting **scologin** configuration resources, see the **scologin**(XC) manual page.

If you want to authorize access for a display other than the one **scologin** manages by default (*/dev/tty02*), modify the */usr/lib/X11/scologin/Xservers* file to configure the desired X server and tty on which you want the server to run.

See "Running scologin with the Xservers file" (page 57) for information on how to do this.

Now you can start **scologin**, using the **scologin** administration script. As *root*, run the following command:

/etc/scologin start

If you want to store your authorization records in a file other than **$HOME**/*.Xauthority*, add a line to your *.login* file that sets the **$XAUTHORITY** environment variable to the desired filename.

Step 3: Logging in through scologin

Log in through the **scologin** window. **scologin** generates a random authorization code and passes it to the server. Your user account also receives the authorization record in the authorization file in your **$HOME** directory. By default this file is named *.Xauthority*.

Step 4: Disabling user-defined display access

If you personally configured access to your display for remote machines using the **xhost** utility, you should disable this access.

You can examine the current host permissions, to see if any access is still permitted with the following command:

xhost

If this command returns a list of machines that still have permission to use your display, run the following command to deactivate the permissions:

xhost -

This command eliminates access to your display for any machines you may have configured earlier.

Step 5: Sharing authorization records with other users

To allow other users to access your display, transfer the authorization record that **scologin** generates for your X server from your **$HOME**/*.Xauthority* file into the **$HOME**/*.Xauthority* file of the other users.

Although it is easy to give other users copies of your **$XAUTHORITY** file, this practice is not recommended, especially if the file needs to be transferred over a network. Preferred practice is to run **xauth** to extract the authorization record for a specific display and merge it into another user's authorization file.

First, list the servers for which you have authorization by running the following command:

xauth list

Note that each line starts with a display name in the following format:

hostname:display_number

To extract the authorization record, run the following command:

xauth extract ***tempfile displayname***

Be sure ***displayname*** matches the string displayed by the **xauth list** command. ***tempfilename*** is a file that you and other users have agreed to use.

If the other users are going to merge the authorization record into their authorization files themselves, be sure to set ***tempfilename***'s permissions so that the other users can access it.

The other users can now merge the authorization record in ***tempfilename*** into their own authorization files, or you can do it for them as *root*, with the following command:

xauth merge ***tempfilename***

When the other users have merged the authorization record into their authorization files, delete ***tempfilename***.

If you do not want to create the temporary file, ***tempfilename***, you can use a pipe to redirect the authorization record from the **xauth extract** command to the **xauth merge** command as follows:

xauth extract - *displayname* | xauth -f *authfile* merge -

The dashes in each **xauth** command cause output to be directed to standard output instead of to a file, and for input to come from standard input instead of from a file. In this case, ***authfile*** is the pathname of the other user's authorization file. You must have read and write permission to ***authfile*** for this command to work.

You can use a similar command line to share authorization records across the network. For example, if you log in on *boston* and want to give your server's authorization record to your account on *tusconey*, execute the following command on *boston*:

xauth extract - boston:0 | rcmd tusconey /usr/bin/X11/xauth merge -

See also:

- **xauth**(X) manual page

Running the remote client

Once you have set access permission for the client and gained access to the remote machine, you can execute clients on the remote machine. There are two ways to tell the remote client which X server to interact with over the network:

- before running the client, set the **$DISPLAY** environment variable (this page) on the remote machine to point to your X server
- specify your X server in the client command line with the **-display** option (page 74)

Running clients with the DISPLAY environment variable

If you log in to the remote host to run clients, you can set the **$DISPLAY** environment variable to point to your X server so that you do not have to specify your server on the command line every time you run a client.

If you are using **csh**, set the **$DISPLAY** environment variable with the following command:

setenv DISPLAY ***displayname***

If you are using **sh** or **ksh**, set the **$DISPLAY** environment variable with the following command:

DISPLAY=*displayname*; export DISPLAY

The ***displayname*** specification uses the following form:

[***hostname***]:***display_number***[.***screen_number***]

hostname specifies the machine on which the display is running and must be either a machine name or the machine's network address, as listed in */etc/hosts*. If you omit ***hostname***, the display is presumed to be running locally.

:display_number specifies one of the displays on ***hostname***. Each display on a system is assigned a ***:display_number***, beginning with 0. ***screen_number*** specifies the screen on which the display is running.

See also:

- "Running clients with the -display option" (page 74)
- "Using environment variables" (page 51) for more information on the **$DISPLAY** environment variable

Running clients with the -display option

The X clients supplied with the system accept the standard "Xt" command line options, including the **-display** option, which allows you to direct the output of the client to a specific X server when you start the client. Use the following command line syntax:

client* -display *displayname

Display names are specified in the following form:

[*hostname*]:*display_number*[.*screen_number*]

See "Running clients with the DISPLAY environment variable" (page 73) for more information on how to set ***displayname***.

You can use the above syntax regardless of whether you are executing the client while logged in to the host through **rlogin** or **telnet**, or through **rcmd**. For example, if you are running the :0 server on *boston* and want to run the **xclock** client on *tusconey*, run the following command from *boston*:

rcmd tusconey "/usr/bin/X11/xclock -display boston:0" &

Similarly, if you log in to *tusconey* with **rlogin** or **telnet**, execute the following command:

/usr/bin/X11/xclock -display boston:0 &

Note that in the above cases, the ampersand (&) runs the client in the background so that you do not have to close the client to get your command line back. You can close the client with the **Window** menu or by typing ⟨Alt⟩⟨F4⟩.

Example of running a remote client on your display

This section provides a comprehensive example that ties together many of the concepts and procedures discussed in this chapter.

For the purposes of this example, let's assume you have accounts on two SCO OpenServer machines: one that sits on your desk, named *boston*, and another, named *tusconey*, that resides in another room. The two machines are part of the same network, and their names and IP addresses are listed in each other's */etc/hosts* files. You have root privileges on *boston*, but you do not have *root* privileges on *tusconey*. Your account name is the same on both machines. You are accustomed to using the clients installed on *boston* from the default server running on tty02. You just received mail that on the machine *tusconey* there is a nicely-configured version of the desktop client, **xdt3**, and you want to try it out.

This example explains how to:

- establish user equivalence on *tusconey*
- configure the X server on *boston* so that you are the only user on *tusconey* that can access *boston*'s display
- log in through **scologin** using "failsafe" so the Desktop client does not start
- run **xdt3** on *tusconey* and display the client on *boston*.

The following steps result in *tusconey*'s Desktop client displaying on your X server on *boston*:

1. On *boston*, switch to *tty01* by pressing ⟨Ctrl⟩⟨Alt⟩⟨F1⟩.
2. Log into *boston* as *root*.
3. Log into *tusconey* remotely, using the **rlogin** command. When prompted, enter your login name and password.
4. On *tusconey*, edit the *.rhosts* file in your home directory, or if that file does not exist, create it. Add the following line:

   boston *username*

   In this command, ***username*** is your login name. When you have finished, save and exit the file. You have established user equivalence on *tusconey*. You can now log off *tusconey*.
5. Back on *boston*, change to the */etc* directory and open the *X0.hosts* file for editing. If *X0.hosts* contains any host machine names, remove them or comment them out with "#" characters. When you have finished, save and exit the file.
6. Change to the */usr/lib/X11/scologin* directory and edit the *Xconfig* file. If the **DisplayManager*authorize** resource is not defined as "true," or if it is not listed at all, add the following line:

   DisplayManager*authorize: true

   When you have finished, save and exit the file.
7. If you modified the contents of */usr/lib/X11/scologin/Xconfig*, run **scologin stop**, then run **scologin start**.

   NOTE This script shuts down all **scologin** processes on your system, which results in the closure of any Graphical Environment sessions running at that time. You should notify users before you actually stop **scologin**.
8. Switch to *tty02* by pressing ⟨Ctrl⟩⟨Alt⟩⟨F2⟩.
9. When the **scologin** window appears, start a "failsafe" session. Log in by typing your user login and password in the appropriate fields, then press ⟨F1⟩ instead of a carriage return. This starts a failsafe session instead of

running a default session managed by **scosession**. The failsafe session consists of only a **scoterm** window. The window manager and Desktop are not started. For more details on failsafe sessions, see the **scologin**(XC) manual page.

10. Type the following command in the **scoterm** window:

    xauth list

    The result is a list of servers for which you have authorization records. The line beginning with "boston:0" indicates that you have authorization to your local X server.

11. To authorize your account on *tusconey* to access your display, run the following command:

    xauth extract - boston:0 | rcmd tusconey /usr/bin/X11/xauth merge -

    If the *.Xauthority* file does not already exist, the following message appears:

    ```
    /usr/bin/X11/xauthority: creating new authority file username/.Xauthority
    ```

    In this message, ***username*** is your login name.

12. To make sure the authorization code was successfully transferred to your account on *tusconey*, run the following command:

    rcmd tusconey /usr/bin/X11/xauth list

    The line beginning with "boston:0" indicates that your account on *tusconey* has authorization to access your X server.

13. Log in to *tusconey* with the following command:

    rlogin tusconey

    Note that you are not prompted for a password because you have user equivalence on *tusconey*. When you are logged in, run the following commands:

    ```
    PATH=:/usr/bin/X11
    DISPLAY=boston:0
    export DISPLAY PATH
    scosession 2>/dev/null &
    ```

    When the Desktop appears, double-click on the **UNIX** icon to open a **scoterm** window. In the window, run **hostname** to verify that you are using *tusconey*'s filesystem.

    When you are done using the Desktop, exit through the **File** menu.

The preceding steps only apply to your current X session. If you want to make these changes take effect every time you start a session, you must reauthorize your account as shown in Step 11 above. You can do this automatically by placing the command given in Step 11 in the file **$HOME**/*.startxrc*.

NOTE In the steps above, you logged in using the failsafe login. When you exit the original "failsafe" window, this will not only close that window but also end your current X session.

Chapter 5
Understanding resources

The SCO OpenServer Graphical Environment provides a mechanism that allows you to specify characteristics that take effect every time you run a client. Almost every feature of a client program can be controlled by changing the value associated with a variable called a resource. You can specify how a client looks on the screen: its size and placement, its border and background color or pattern, whether the window has a scroll bar, and so on. Some applications even allow you to redefine the keystrokes or pointer actions used to control the application.

You can also use command line options to customize your clients. However, you are limited in the number of features you can change with command line options.

Specifically, this chapter describes:

- what resources are (page 80)
- the syntax for setting resources (page 81)
- the different places you can specify resources (page 87)
- setting resources in the X server with **xrdb** (page 90)
- using command line options to set resources (page 93)
- guidelines for managing resources (page 98)

This chapter is provided as a reference for understanding what resources are and how they are used. Following chapters discuss the specifics of changing resources for colors, fonts, window geometry and so forth.

About resources

A *resource* is any parameter that affects a client's behavior or appearance, such as foreground colors, background colors, fonts, window size, and window placement. A resource is set through a *resource specification*, which contains the *resource variable* and a *resource value*. For example:

client*resource_name: resource_value

When you execute a client, it locates any resource specifications that affect it and then uses those attributes to define its appearance and behavior.

A resource is typically named for the aspect of appearance or behavior that it controls. For example, the **fontList** resource controls the font that is used to display text in the Graphical Environment.

In applications written with the X Toolkit (or an Xt-based toolkit such as the OSF/Motif toolkit), resources may be associated with separate *widgets* within an application. This allows you to control an entire class of a widget in the client, as well as control specific instances of a class. For example, you can set all of a client's buttons to display in blue, except for the **Help** button, which displays in red.

Thus, with resources, you can modify behavior and appearance on a general level, or a very specific level. You can specify a resource that controls only one feature of a single application or specify a resource that controls one feature of multiple objects within multiple applications.

There are several ways you can specify resources: through the resource database (using a program called **xrdb**), with system resource files, with resource files in your home directory, and from the command line.

- Generally, resources are specified in files. Depending on the file you use, however, the precedence of your resource specifications vary.
- The system uses certain files to place some primary resource values directly in the server, through the resource database. This approach makes the values available to all clients, regardless of the host where the client is executed. The resource database is managed by **xrdb**, the X resource database manager.
- The system also contains system-wide resource files that set defaults for clients run by all users on the system. These resources are loaded when a particular client is invoked. You can also create individual resource files that contain resource settings for a variety of clients, for an individual user. These values apply to the clients that are run on the host machine only.
- You can also set resources on the command line, using command line options.

A set of routines called the "resource manager" determines things like the order in which the various resource files are read, the syntax for resource specifications, and the rules of precedence by which conflicting or competing resource specifications are resolved.

The concepts and terminology associated with resources may seem complex at first. However, once you read this chapter and spend some time experimenting by creating your own resource specifications, you will find the task of setting resources straightforward.

See also:

- "Syntax for resource specifications" (this page)
- "Setting resources in the X server" (page 90)
- "Using command line options to configure clients" (page 93)

Syntax for resource specifications

A resource specification consists of the following components,

[client][restrictions*restrictions...]*resource_name: resource_value*

where:

- ***client*** is the client or application to which you want this specification to apply. You can supply either the client's binary or class name. This component is optional; if you omit this part of the specification, the resource definition applies to all clients that support this resource.
- ***restrictions,*** usually in the form of widget names or classes, define the extent to which you want the resource definition to effect a client's appearance or behavior. You can only supply ***restrictions*** that are used and understood by the client(s) you are customizing, otherwise the resource specification is ignored. You can specify any number of restriction components, leaving the resource specification very general, or narrowing its focus to a discrete part of a client's functionality. This component is optional; if you omit this part of the specification, the resource definition applies to all relevant widgets of the specified client.
- ***resource_name*** is the actual resource variable that you want to define. Each client and each widget has a set of resources that it recognizes. (Refer to the client's manual page for a list of the resources that can be used to customize the application's behavior and appearance.) This component must be specified.
- ***resource_value*** is the value that you want to assign to the resource variable. For example, if you want to define a font resource, you would supply a font name as the value. Different resources require different types of input; an

overview of the various resource values you can expect to use is provided in "Specifying values in resource specifications" (page 85). Refer to the manual pages of the clients you want to configure for the values that are expected by the clients' supported resources.

- The various components of a resource specification are separated by a delimiter, in this case the asterisk "*" character. If you are unsure about what type of delimiter to use in a resource specification, you can always safely use the asterisk. The delimiters, also known as "bindings", are described later in "Using delimiters in resource specifications" (page 84).
- A colon and whitespace separate the ***client***, ***restrictions***, and ***resource_name*** components from the ***resource_value***.

> **NOTE** Be careful that you do not omit the colon at the end of a resource specification. This is an easy mistake to make and the resource manager does not provide any error messages. If there is an error in a resource specification, including a syntax error such as the omission of the colon, or a misspelling, the specification is ignored and the value you set does not take effect.

For information on choosing the actual names for the ***client***, ***restrictions***, and ***resource_name*** components, see "Using classes and instances in resource specifications" (page 83).

The following examples show various combinations of the above components and describe the effects the resource specifications have on the system:

`*foreground: yellow`	specifies that the foreground color is yellow. Because no client is specified, the color applies to all clients. Because no restrictions are indicated, the color applies everywhere within all clients.
`XClock*foreground: pink`	specifies that the foreground color for the **xclock** client is pink. The specification applies to all aspects of the **xclock** client that use the **foreground** resource.
`Xman*topBox*foreground: blue`	specifies that the foreground color is blue for the **topBox** widget, which is **xman**'s main options menu. The **topBox** component of the resource specification is a restriction, limiting the use of the **foreground** resource to a small portion of the **xman** client.

Using classes and instances in resource specifications

When specifying the ***client***, ***restrictions***, and ***resource_name*** components of a resource specification, you need to decide if you want to use the component's class name or instance name. The "class" is the general category to which the component belongs, whereas the "instance" is the actual client, widget, or resource variable.

A class can consist of several different versions of an application, or several different widgets or resources. For example, in the case of the **scoterm** client, the background color resource (**background**), the active background color resource (**activeBackground**), and the top shadow color resource (**topShadowColor**) are all instances of the same class, **Background**.

You might choose to specify a class name instead of an instance name, if you wanted to set values for several resources that fell within the same class. This way, you could create the following resource specification:

```
ScoTerm*Background:     blue
```

This specification is much quicker to define and achieves the same result as:

```
ScoTerm*background:         blue
ScoTerm*activeBackground:   blue
ScoTerm*topShadowColor:     blue
```

In the case of clients, you might want to specify an application's class name instead of the binary name if, for example, there are different versions of the same program (perhaps to accommodate different machine architecture) on your network. By using the class name to set a resource, you could ensure that the resource specification would be used by all versions of the same client.

You can combine the use of class names and instance names in a resource specification so that you define a default for a wide range of cases and then define a particular case that varies from the default without overriding it. For example, you could specify that the buttons in all of the dialog boxes for a particular client be displayed in blue, with the exception of a particular button, which you want displayed in red. You might create resource specifications that read:

```
xclient*buttonbox*XmPushButton*foreground:  blue
xclient*buttonbox*delete*foreground:        red
```

The **XmPushButton** class configures all buttons in the dialog boxes, but the **delete** instance overrides the default for the **Delete** button only. This type of specification works because an instance name always overrides the corresponding class name.

By convention, class names begin with an uppercase letter, while instance names begin with a lowercase letter. However, if an instance name is a compound word, such as **activeBackground**, the second word is usually capitalized.

To identify the class and instance names for the resource variables that you can set for a client, refer to the client's manual pages.

Using delimiters in resource specifications

Components of a resource specification are separated with delimiters, which are also known as "bindings". The following delimiters can be used:

- an asterisk " * ", to specify a "loose" binding
- a dot " . ", to specify a "tight" binding

The asterisk is a wildcard character and signifies that there can be any number of levels in the widget hierarchy between the two surrounding components. On the other hand, the dot indicates that the components must be next to each other in the hierarchy.

If you want to specify tight bindings, you must be very familiar with the widget hierarchy; it is easy to use these bindings incorrectly.

For example, the following resource specification, indicating that **xterm** windows should be created with a scroll bar, does **not** work:

```
XTerm.scrollBar: true
```

This specification ignores the widget hierarchy of **xterm**, in which the VT100 window is considered one widget, the Tektronix window another, and the menus a third. To configure **xterm** to use a scroll bar using tight bindings, you would need to specify:

```
XTerm.VT100.scrollBar: true
```

Rather than decipher the widget hierarchy of a client and risk making a mistake, it is far simpler to use the asterisk delimiter in your resource specifications:

```
XTerm*scrollBar:  true
```

The asterisk, acting as a wildcard, tells the resource manager to locate any widgets in **xterm**'s widget hierarchy that support a scroll bar and assign a scroll bar to them.

NOTE In an application that supports multiple levels of widgets, you can mix loose and tight bindings. However, it is generally recommended that you use the asterisk delimiter, even in resource specifications for "simple" clients. This is because clients can change from release to release, incorporating new widgets in the hierarchy. Using loose bindings ensures that your resource specifications will work on future versions of a client.

Specifying values in resource specifications

The type of text string that you supply for a ***resource_value*** depends on the nature of the resource variable. Resources tend to fall into eight categories, each requiring a different type of value.

These basic categories are described below:

Colors: Color resources require a color value, as specified in the color database, */usr/lib/X11/rgb.txt*. For example:

```
XClock*foreground:  magenta
XCalc*background:   aquamarine
```

Because the Graphical Environment provides **scocolor**, a color editor that allows you to create palettes of colors, it is recommended that you refrain from using actual color values with resources and instead assign palette resource variables. These variables are then assigned actual color values through **scocolor**. The following example shows how several color resources have been assigned appropriate palette resource variables instead of actual color values:

```
ScoEdit*background:  scoBackground
ScoMail*foreground:  scoForeground
```

For information on the color database, the **scocolor** client and its interaction with palette resource variables, and for details on how to specify color resources, see Chapter 6, "Changing colors" (page 99).

Fonts: Font resources require the name of an available font. You can use either a full name, a wildcarded specification, or a font alias. For example:

```
ScoTerm*Font: -adobe-courier-bold-r-normal--14-140-75-75-m-90-iso8859-1
ScoTerm*Font: *courier-bold-r*140*
ScoTerm*Font: courierB14
```

For information on fonts and specifying font resources, see Chapter 7, "Changing fonts" (page 125).

Geometry: Geometry resources require information to determine the size of the client's window and/or *x* and *y* grid coordinates to determine the location of the window. For example:

```
XCalc*geometry:   180x240-0-0
XClock*geometry:  -0+5
```

For information on specifying geometry resources, see Chapter 8, "Configuring window size and location" (page 159).

Cursor names: Cursor resources require the name of the file in */usr/include/X11/bitmaps* that contains the cursor you want to use. For example:

```
ScoTerm*pointerShape: gumby
```

For information on specifying a different cursor pixmap for the **scoterm** client, see Chapter 9, "Changing cursor appearance" (page 173).

Pixmaps: Pixmaps are patterns, like bitmaps, that are used to texture or color an area on your display. Pixmap resources are specified like cursors or bitmaps. For example:

```
Pmwm*scocolor*backgroundPixmap:  scales
*bottomShadowPixmap:             mensetmanus
```

Numeric quantities:
Some resources require a numeric value. For example:

```
XClock*update:      30
XLoad*update:       60
XLogo*borderWidth:  10
```

Boolean values: Some resources require a boolean value, such as "true" or "false", "yes" or "no", or "on" or "off". For example:

```
ScoTerm*scrollBar:  false
Pmwm*focusAutoRaise:  true
```

Translations: Some resources assign particular strings to keys, or assign actions to keys or mouse buttons. These resources are fairly complex and the meaning behind the values may be hard to understand from a simple example. See Chapter 11, "Configuring the keyboard for the server" (page 209) and Chapter 26, "Mapping mouse triggers for the Desktop" (page 371) for more information on specifying these types of resources.

Precedence rules for resource specifications

Because of the great flexibility you have in defining resource specifications, it is not uncommon to create specifications that conflict with each other. For example, take the following resource specifications for a hypothetical client, **xclient**:

```
xclient*Buttons*background:  blue
xclient*help*background:     red
```

The first resource specification sets the background color of all buttons to blue. The second resource specification overrides the value specified in the first specification, but only for the instance of widgets with the name "help." In the event of conflicting specifications such as this, there are a number of rules that the resource manager follows in deciding which resource specification should have priority. In the case of a conflict, the most specific resource definition is used.

The precedence rules are as follows:

- The dot delimiter takes precedence over the asterisk delimiter. For example, **XTerm.VT100.geometry** is more specific than **XTerm*geometry**.
- Instance names take precedence over class names. For example, ***foreground** is more specific than ***Foreground**.
- An instance or class name that is stated explicitly takes precedence over one that is omitted. Specifying a component is more specific than omitting it. For example, the following definition:

  ScoTerm*XmMenuShell*XmRowColumn*XmLabel*fontList

  is more specific than:

  ScoTerm*fontList.

Methods for specifying resources

Overall, there are eight different ways to specify resources. These different approaches can be grouped in the following categories:

Application-specific resources resource specifications, stored in files that are named for the applications they configure. These files are only read by the particular client, when it is first started.

Server-specific resources resource specifications that apply to all clients, regardless of the host on which an application is running

Host-specific settings	resource specifications that relate only to the host on which an application is running, regardless of where the client is displayed
Command line options	resource specifications for the current invocation of a client

When a client is executed, the eight sources of resource definition are consulted in the following order. Resources that are defined later in this list have precedence over resources defined earlier.

1. **System-wide application-specific resource files:**
   These files contain site-wide defaults for classes of applications. The files are named after the applications' class names and they are stored in the */usr/lib/X11/app-defaults* directory. For example, the resource file for the **scoterm** client is called */usr/lib/X11/app-defaults/ScoTerm*. When a client is executed, the resources in the client's resource file are loaded into the resource manager.

2. **User application-specific resource files:**
   These files have the same name as those above, however they are stored in a different location, specified by the **$XAPPLRESDIR** environment variable. If this variable is not set, the files are located in a user's home directory. Use these files to override system-wide resource specifications for individual users.

3. **The X server's resource database:**
   A number of resources are automatically loaded into the resource database and stored in the X server when the **scosession** session manager is first started. These resources are defined in files, named for an application's class, in the */usr/lib/X11/sco/startup* directory. The resources defined in these files are placed in the resource database by the **xrdb** client. These resources are available to all applications, regardless of the host on which they execute.

   If **scosession** is not running, you can load resources into the resource database by running **xrdb** manually. These resources have the same level of priority as resources loaded when **scosession** is started.

4. **The $HOME/.Xdefaults file:**
   This file is **only** used if resources are not loaded into the X server by **xrdb**. If you are running **scosession**, which runs **xrdb**, this file is never used. Resources specified in this file are only available on the local machine; you need to create one of these files for each machine on which you run clients.

5. **The file defined by $XENVIRONMENT:**
   The contents of any file specified by the **$XENVIRONMENT** variable are loaded into the resource manager.

6. **The $HOME/.Xdefaults-*hostname* file:**
   If the **$XENVIRONMENT** variable is not defined, the resource manager looks for a file named *.Xdefaults-**hostname*** in a user's home directory, where ***hostname*** is the name of the machine on which a client is running. The resources defined in this file effect any client that is run on the local machine, even if its output is sent to a remote display. You need to create one of these files for each machine on which you run clients.
7. **Application-specific options:**
   Some applications have specific command-line options that only work for that client. For example, **xclock**'s **-chime** option or **xpr**'s **-scale** option.
8. **X Toolkit-standard options:**
   Generally, all applications that use the X Toolkit, including the clients supplied with the system, accept a standard list of command-line options. In particular, the **-xrm** option can used to load any resource specification for the current Graphical Environment session from the command line.

The resource manager searches these locations for valid resource specifications, in the indicated order. All of the resource specifications are loaded and sorted. If a conflict in resource specifications is located, the conflict is resolved according to the precedence rules described earlier in this chapter. If the same resource is assigned conflicting values, the last definition encountered is used. For example, if the resource manager encounters the following resource specification early on:

```
ScoTerm*scrollBar: false
```

and then locates the exact same specification, with a different value later, in a host-specific file:

```
ScoTerm*scrollBar: true
```

the "true" value is used and **scoterm** is assigned a scroll bar.

After the resource manager determines which of the resource specifications should be implemented for a particular client, the client then merges these values with its own internal defaults, if any.

See also:

- "Setting resources in the X server" (page 90) for information on storing resources directly in the X server with **xrdb**
- "Using command line options to configure clients" (page 93) for information on using command line options to set resources

Setting resources in the X server

Resources are loaded into the X server by the X resource database client, **xrdb**. The **xrdb** client is run automatically by **scosession** when you log in through **scologin** or when you run the **/usr/bin/startx** script from the command line.

The */usr/lib/X11/sco/startup* directory contains several display-specific resource files, named for the clients they represent. For example, the *ScoHelp* file contains resources for **scohelp** and the *ScoMail* file contains resources for **scomail**. This directory also contains two other files, *Colors* and *Fonts*. These files contain global color and font resources that are used by clients unless they specifically define their own color and font values.

The **xrdb** client reads the values in these files when **scosession** is started and loads them into the resource database, storing them directly in the X server. (Technically speaking, the resource values are stored in a data structure referred to as the RESOURCE_MANAGER property of the Root window for that server. This property is simply referred to as the resource database.)

Resources that are stored in the X server are available to all clients, regardless of the machine on which they are run.

While the **xrdb** client is run by **scosession**, it can also be invoked interactively, using the following syntax:

xrdb [***options***] [***filename***]

When using **xrdb**, note the following:

- The **xrdb** client takes several options, the most important of which are described in the following sections. You can also refer to the **xrdb**(XC) manual page for more information on these options.
- The optional ***filename*** argument specifies the name of a file from which you want **xrdb** to read resource values. The ***filename*** you specify can be any resource file, including the *.Xdefaults-**hostname*** file, if you made changes to it during a Graphical Environment session. If no filename is specified, **xrdb** reads its data from standard input.

> **NOTE** If you run **xrdb** during a Graphical Environment session without using the **-merge** option, (page 91) you override all of the resources in the resource database that were loaded when you first started the X server.

See also:

- "Examining the contents of the resource database" (this page)
- "Loading new values into the resource database" (this page)
- "Saving new specifications in a resource file" (page 92)
- "Removing resource definitions from the resource database" (page 93)
- "Using the session manager" (page 48) for information on how **scosession** uses **xrdb**

Examining the contents of the resource database

You can determine what resources are currently loaded in the resource database using the **-query** option. For example:

xrdb -query

The **-query** option produces a list of all of the currently recognized resource specifications, such as:

```
*Background:  scoBackground
*Font:  -*-helvetica-medium-r-*--10-*-p-*
*FontList:  -*-helvetica-medium-r-*--10-*-p-*
*Foreground:  scoForeground
*XmLabel*FontList:  -*-helvetica-medium-r-*--10-*-p-*
*XmLabelGadget*FontList:  -*-helvetica-medium-r-*--10-*-p-*
```

Note that if your system was modified so that **scosession** (which runs **xrdb**) is not started, or if you removed all definitions from the resource database, this command produces no output.

Loading new values into the resource database

If you want to add new resource values to the resource database without overriding the current values, you must use the **-merge** option with the **xrdb** client.

For example, to add new resources that are stored in a file called *myresources*, you would enter:

xrdb -merge myresources

As another example, if you wanted to configure subsequent versions of **scoterm** to display scroll bars, you could use standard input and enter:

xrdb -merge
ScoTerm*scrollBar: True

When you are finished, press ⟨Ctrl⟩**d** to end the standard input.

Any new resource specifications that you add during a Graphical Environment session will not affect any clients that are currently running. If you want your current clients to reflect your new resource settings, you need to restart them.

NOTE If you add a new resource specification but do not see the behavior you expect, it may be that a more specific resource specification has already been loaded into the resource database, and not that you used the **-merge** option incorrectly.

If your specifications do not seem to work, use the **-query** option (page 91) to list the current values in the resource database. Check this list for any conflicting resource specifications.

Saving new specifications in a resource file

If you loaded new resource specifications into the resource database using standard input or the **-xrm** command line option (page 97) and now you would like to save the settings permanently in a resource file, you do not need to edit the file manually (although you certainly could.) The **-edit** option allows you to write the current contents of the resource database to a file. For example:

xrdb -edit ~/.Xdefaults-*hostname*

This example saves the current contents of the database in the file *.Xdefaults-**hostname***, in your home directory.

NOTE If the file you specify already exists, its contents are overwritten with the new values. However, **xrdb** is smart enough to preserve any comments and preprocessor declarations in the file being overwritten, replacing only the old resource specifications.

If you want to save a backup copy of an existing resource file, use the **-backup** option, in addition to the **-edit** option:

xrdb -edit ~/.Xdefaults-*hostname* -backup old

The string following the **-backup** option is used as an extension that is appended to the old filename. In this example, the old copy of the *.Xdefaults-**hostname*** file is saved as *.Xdefaults-**hostname**old.*

Removing resource definitions from the resource database

You can remove the entire current contents of the resource database from the X server, using the **-remove** option.

There is no way to delete a single resource definition other than to read the current **xrdb** values into a file, remove the unwanted resource specifications from the file, and then load the file back into the resource database.

For example:

xrdb -query > *filename*

Use an editor to edit the file, deleting the resource definitions you no longer want and save the file. Then read the edited values back into the resource database:

xrdb -load *filename*

If you are running **scosession**, these changes to the resource database are stored and used again in future sessions. If you are not using **scosession**, however, these modifications are only implemented for the current Graphical Environment session. If you log out and log back in, the deleted resource definitions are once again loaded into the resource database. To remove a resource specification permanently without running **scosession**, you must edit the resource file that contains the definition.

Using command line options to configure clients

Most clients have a set of command line options that control some of their appearance and behavior characteristics. These options control X resource specifications such as colors, fonts, window geometry, and so forth. (You can use resource specifications to configure client appearance and behavior on a much wider scale. However, resources are more complicated to set than the command line options.)

You can use these command line options when launching any client from a **scoterm** window. Ordinarily, the effects of a command line option are temporary, only existing for the current session. However, if you end your Graphical Environment session while the client you configured with command line options is still running, the client reappears in its same configuration in your next session.

For example, if you start the **xclock** client from a **scoterm** window, using the **-geometry** command line option to place the window at the upper-left corner of the screen, and then end your Graphical Environment session, the next time you resume the session, **xclock** will still be located in the upper-left corner. Note, however, that if you move the window to a new location, quit the

session and then resume it at a later time, the **xclock** window is located in the upper-left corner according to your command line argument, and not in the location where you moved the window before quitting the Graphical Environment.

Command line options also take precedence over all other resource specifications, unless the command line setting is more general than a specified resource definition.

A client's manual page lists all of the command line options that are valid to use with that client. However, most clients support the options listed in Table 5-1, "Standard client command line options".

Table 5-1 Standard client command line options

Command line option	Alternate option	Description
-bg	-background	sets the background color of a client's window
-bd	-border	sets the border color of a client's window frame, if you are not running **pmwm** or **mwm**
-bw		sets the border width in pixels, if you are not running **pmwm** or **mwm**
-display	-d	sets the name of the display you want the client to use
-fg	-foreground	sets the foreground color of a client's window (usually the text)
-fn	-font	specifies a font name
-geometry		sets the size and location of a client window on the display
-iconic		starts the application in iconified form
-name		specifies a name for the application being run
-rv	-reverse	reverses foreground and background colors
+rv		does not reverse foreground and background colors
-title		sets the string that is used for the title in the window frame
-xrm		passes a resource specification to the resource manager

See also:

- "Window appearance options" (this page)
- "Display specification option" (this page)
- "Font specification option" (page 96)
- "Window size and location option" (page 96)
- "Client name option" (page 97)
- "Window title option" (page 97)
- "Resource specifications on the command line" (page 97)

Window appearance options

Most of the standard command line options determine the appearance of a client's window. They specify the background color (**-bg**), the foreground color (**-fg**), the border color (**-bd**) and the border width (**-bw**) in pixels. The colors you assign must be available from the color database.

For example, to start a **scoterm** client so it uses yellow characters on a navy background, you would enter the following command:

```
scoterm -bg navy -fg yellow &
```

If a color name includes embedded space, you must include the color name in quotes. For example, to specify light blue as the background for **scoterm**, you would enter:

```
scoterm -bg "light blue" &
```

The **-rv** option specifies that the client should reverse the colors that were defined for the background and foreground resources.

NOTE Because the window manager places its frames over the window borders provided by the X server, the **-bd** and **-bw** options have no effect unless a window manager is **not** running.

See also:

- Chapter 6, "Changing colors" (page 99) for more information on colors, including the color database

Display specification option

An important command line option is **-display**, which you can use to specify the display on which you want the client's output to appear. You can use this option as an alternative to setting the **$DISPLAY** environment variable before starting the application.

For example, if you want to run the **xbiff** client, and display its output on a remote server on a machine named *scooter*, you would enter:

xbiff -display scooter:0.0 &

where "scooter" is the name of the remote host machine.

See also:

- Chapter 4, "Running remote programs" (page 65) for more information on specifying remote displays

Font specification option

A client's font resource controls the font that is used to produce text output. Like colors, fonts are specified by names; the font name you use must be included in the font database.

For example, to configure the **scoterm** client to use a 12-point courier font, you would enter:

scomail -fn -adobe-courier-medium-r-normal--12-120-75-75-m-60-iso8859-1 &

Instead of entering an entire font name, you can also use wildcard characters, or a defined font alias.

See also:

- Chapter 7, "Changing fonts" (page 125) for more information on specifying fonts

Window size and location option

Another common option is **-geometry**, which is used to specify the size and location of the client's main window. The geometry is specified in the following format:

[***width*** x ***height***][±***xoff***±***yoff***]

The ***width*** and ***height*** values specify the size of the window in pixels. (For **scoterm** or **xterm**, you must specify the ***width*** and ***height*** in terms of columns and rows of text.) These values are optional.

The ±***xoff*** and ±***yoff*** values indicate the position on the grid, in pixels, where the window should be located. These values are also optional.

A positive ***xoff*** value indicates the number of pixels that the left side of the window is offset from the left side of the screen. A negative ***xoff*** value, on the other hand, specifies the number of pixels that the right edge of the window is offset from the right edge of the screen. Similarly, positive and negative ***yoff***

values indicate offsets from the screen edges of the top and bottom edges of the window, respectively.

For example, the following command places a 120-pixel by 120-pixel **xclock** window in the upper right corner of your screen with a 16-pixel gap between the clock's frame and the screen's top and right edges:

xclock -geometry 120x120-16+16 &

See also:

- Chapter 8, "Configuring window size and location" (page 159) for more information on specifying window geometry

Client name option

The **-name** option changes the name by which the X server identifies a client. Changing the name of the application itself affects the way the application interprets resource files.

Window title option

The **-title** option allows you to specify a text string as the title of a client's window. If your client has a title bar or if the window manager puts title bars on windows, this string appears in the title bar.

Window titles can be useful in distinguishing multiple instances of the same application. For example, if you use two **scoterm** windows, one of which is run on a remote host machine, you can set the title of the **scoterm** window so it indicates the name of the host machine on which it is running. You would enter the following:

scoterm -display *hostname:display_number* -title "'uname -n'" &

Resource specifications on the command line

The **-xrm** option allows you to define on the command line, any resource specification that you would otherwise put into a resource file. For example:

scoterm -xrm 'ScoTerm*scrollBar: true' &

Note that the resource specification must be quoted using single quotes, as in the above example.

Any resources that you specify with the **-xrm** option are implemented for the current client session only. As a result, using this approach is a good way to

temporarily change the appearance or behavior of a client without overwriting the default settings.

NOTE If you are using **scosession** and you exit the Graphical Environment with a client that was run from the command line still open on your display, the client is restored when you resume your session. Any resources specified with the **-xrm** option when the client was started are also restored. In this way, the **-xrm** option can define client behavior on a more permanent basis

The **-xrm** option is most useful for setting classes, since most clients have command line options that correspond to instance variable names. For example, the **-fg** command line option sets the **foreground** attribute of a window, but **-xrm** must be used to set **Foreground**.

A resource that is specified with the **-xrm** option does not take effect if a resource that has precedence has already been loaded into the resource database. For example, if the resource database contains the following resource specification:

```
ScoTerm*pointerShape: bogosity
```

the following command line specification of another cursor for **scoterm** will fail:

scoterm -xrm '*pointerShape: gumby' &

This failure results because the resource **ScoTerm*pointerShape** is more specific than the resource ***pointerShape**. To override the resource database so you can use the "Gumby" cursor, you need to use a resource specification that is equally or more specific than the designation in the database. For example:

scoterm -xrm 'ScoTerm*pointerShape: gumby' &

Guidelines for managing resources

*.Xdefaults-**hostname*** files are searched sequentially. Consequently, some performance gain may be realized by putting resource specifications for frequently used clients (such as **pmwm** and **xdt3**) in front of resource specifications for other clients (such as **xclock**).

If you have a resource specification that applies to a specific client and a global resource specification that applies to all clients for the same resource, place the specific definition before the global definition.

These guidelines are particularly relevant if your *.Xdefaults-**hostname*** file is large.

Chapter 6
Changing colors

The SCO OpenServer Graphical Environment provides a large color database from which you can select colors. In most cases, you should use the **scocolor** palette editor to make these color changes. However, there are times when you will need to make a specific color designation using resources.

Specifically, this chapter describes:

- background information about the color database and color resources (page 100)
- changing system-wide colors (page 109)
- changing colors for individual users (page 112)
- changing colors from the command line (page 116)
- adding custom colors to the color database (page 118)

There is also a section of examples (page 121) at the end of this chapter that helps to tie together many of the concepts and procedures discussed in this chapter.

See also:

- "Changing colors with the Color control" (page 27)
- Chapter 5, "Understanding resources" (page 79)

About colors

There are several issues to consider when changing colors on your system. Colors are available from a color database. The **scocolor** client, a convenient tool for previewing colors and changing the color appearance of the Graphical Environment, makes available the colors that are provided by this database, as well as hundreds of other colors that you can create yourself.

The color database is based on the RGB (Red/Green/Blue) color model. The **scocolor** client is based on the RGB color model, but also provides the option of using the HSV (Hue/Saturation/Value) color model as an alternative.

While the color database provides hundreds of colors, there are limits on the number of different colors that you can actually display at the same time. The severity of this limitation depends on the X server you are using and, consequently, on the size of your system colormap.

See also:

- "The color database" (this page)
- "The RGB and HSV color models" (page 101)
- "The scocolor client" (page 103)
- "Colormaps" (page 107)

The color database

The color database, based on the RGB color model, provides a predefined set of colors and shades of gray. This database specifies hundreds of different colors, including various shades for some colors. The database also provides 101 shades of gray. This large number of precisely graduated grays provides a wide variety of shading for gray scale screens.

The color database information is stored in compiled format in two files, *rgb.dir* and *rgb.pag*, located in the */usr/lib/X11* directory. In the same directory, the *rgb.txt* file lists all of the database information in ASCII format.

The *rgb.txt* file lists the name of each color defined in the database, variations on the color names (differing only in spelling, spacing, and capitalization), and the Red, Green, and Blue (RGB) values that are used to create each color. For more information on the RGB color model, see the next section.

To examine the contents of the *rgb.txt* file, open it with any text editor. The following is a partial listing of the *rgb.txt* file:

```
255 250 250     snow
248 248 255     ghost white
248 248 255     GhostWhite
245 245 245     white smoke
245 245 245     WhiteSmoke
220 220 220     gainsboro
255 250 240     floral white
255 250 240     FloralWhite
253 245 230     old lace
253 245 230     OldLace
250 240 230     linen
250 235 215     antique white
250 235 215     AntiqueWhite
255 239 213     papaya whip
255 239 213     PapayaWhip
255 235 205     blanched almond
255 235 205     BlanchedAlmond
255 228 196     bisque
```

You can also use the **showrgb** client to display the contents of the color database.

See also:

- **showrgb**(X) manual page

The RGB and HSV color models

Most color monitors on the market today are based on the RGB color model. Each pixel on the screen is made up of three phosphors, or colors: Red, Green, and Blue. When all three of these colors are illuminated, the pixel appears white. When all three are dark, the pixel appears black. Various combinations of the three phosphors result in a large number of distinct colors and shades. For example, equal portions of red and green, without any blue, result in a shade of yellow.

The intensity of each of the phosphors is controlled by a three-part digital value. As noted in the previous section, the *rgb.txt* file consists of lines such as:

```
127 255 212     aquamarine
```

On each line, the three numeric fields represent the RGB components that comprise the color. These components are decimal values and must be within the range of 0 to 255, where 255 sets the color to full intensity. In the preceding example, the aquamarine color is created from 127/255ths of maximum red, 255/255ths of maximum green, and 212/255ths of maximum blue.

In most cases, using the color names defined in the color database should be adequate for customizing colors for the Graphical Environment. However, if you want to specify a more precise color or use a color that is not defined in the database, you can use a hexadecimal RGB value instead of a name to set a resource or a command line option.

NOTE If you are unfamiliar with hexadecimal numbering, see a basic computer textbook for more information.

Hexadecimal RGB values are somewhat different from the RGB definitions in the *rgb.txt* file. When you specify a hexadecimal value, follow these rules:

- Begin the value with a pound sign "#"
- Specify the red, green, and blue values using 1 to 4 hex digits. Each color must use the same number of hex digits.
- Use a value of "ffff" to set the corresponding color to its maximum value; a value of zero ("0") indicates no color
- You must specify all three RGB values, even if the value for one of the colors is "0"

Using these rules, you can choose from one of the following hexadecimal formats:

```
#RGB
#RRGGBB
#RRRGGGBBB
#RRRRGGGGBBBB
```

When fewer than four digits are used, they represent the most significant bits of the value. For example, "#8E4" is the same as "#8000E0004000". In addition, while #RGB and #RRRGGGBBB have the same relative intensities of the three color components, the second value results in a much brighter color than the first.

You can specify hexadecimal RGB values exactly as you would a color name:

client*resource_name: #RRRGGGBBB

NOTE Because the precision of different adapters and monitors varies, the exact same color values may not produce the exact same shade of color on different displays.

The **scocolor** client (this page) is also based on the RGB color model. However, it allows you to switch to the HSV color model to provide greater flexibility in choosing colors.

Unlike the RGB color model, which is hardware-oriented, the HSV model is user-oriented, based on the more intuitive appeal of combining hue, saturation, and value elements to create a color.

However, the **scocolor** client automatically translates any colors that are selected or created using the HSV model to corresponding RGB values. The HSV option is provided only as an alternative interface for selecting colors.

The scocolor client

The Graphical Environment provides the **scocolor** client, a color palette editor. This client allows users to control the colors that are used for the basic display elements of the Graphical Environment, including window frames and backgrounds. In general, this client is the preferred tool if a user wants to change the colors used by the Graphical Environment.

NOTE The **scocolor** client requires an X server that supports at least 16 colors or grayscales. Only PseudoColor and grayscale visual X servers are supported. If you try to run **scocolor** on a monochrome system, you see an error message.

It is not the aim of this section to explain how to use the **scocolor** client. Rather, this section focuses on how the **scocolor** client interacts with client color resources to specify the color elements of the Graphical Environment display. If you are unfamiliar with how to use the **scocolor** client, refer to "Changing colors with the Color control" (page 27).

See also:

- "Color palettes" (page 104)
- "Color resources and the color palettes" (page 105)

Color palettes

A color palette is a collection of eight colors that controls the color scheme for the Graphical Environment. The eight palette colors are loaded into color buttons, or cells, where each button controls a specific aspect of the environment. The following list correlates each color button with the elements of the environment it affects:

Background controls the background color of all windows on the screen, including directory windows, **scoterm** windows, and windows in which other clients are running

Foreground controls the foreground color of all windows. The foreground usually consists of text, including lists, menus, buttons, icon labels, and so forth.

Top shadow controls the top shadow color of the window frame. This resource gives a window its three-dimensional appearance, especially if the color is a lighter shade of the background color.

Active window controls the background color in the window frame for the currently active window

Active foreground controls the foreground color in the window frame, usually text, for the currently active window

Active top shadow controls the color of the top shadow on the frame of the currently active window

Alternate background controls the color of the Desktop background, as well as the color for scrollbar and sliderbar troughs and the icon box background, if in use

Highlight controls the color of a button when it is pressed

Note that the bottom shadow of a window frame is always black.

The following diagram illustrates the effects the different color buttons have on your Graphical Environment display:

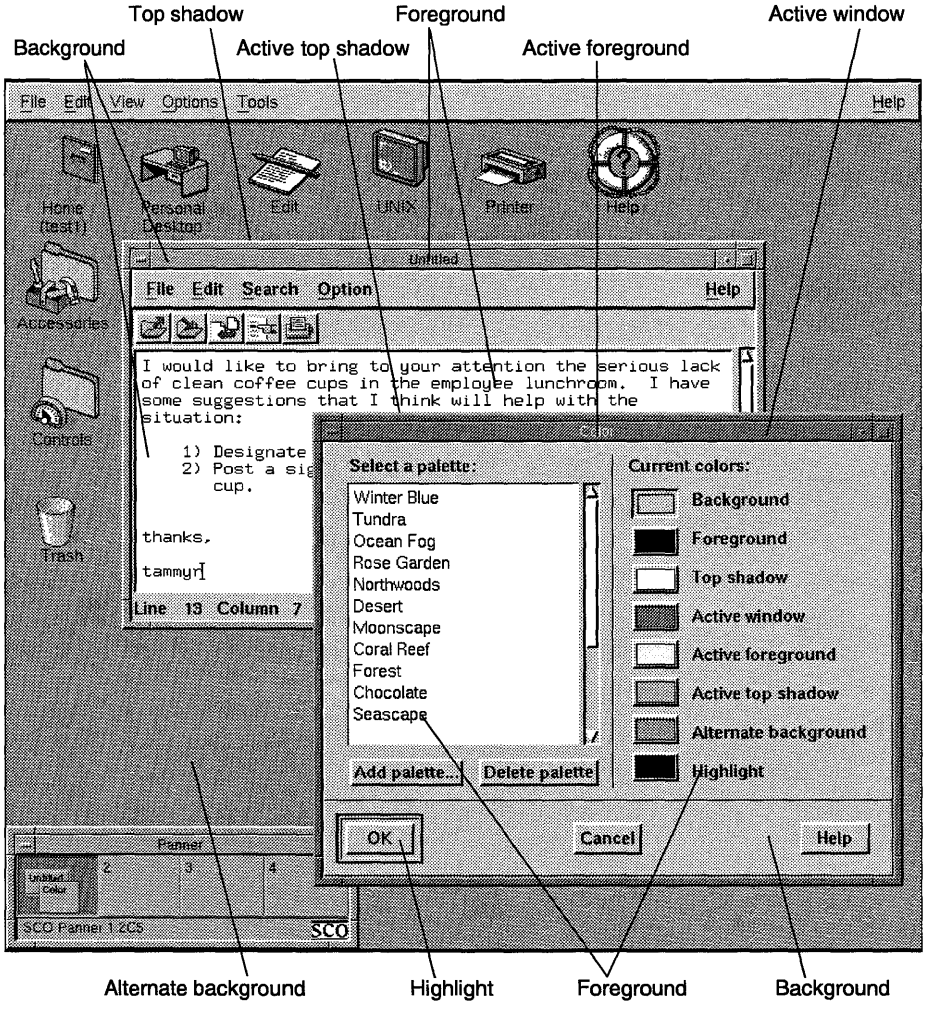

Figure 6-1 Effects of color palettes on the Graphical Environment

Color resources and the color palettes

All of the clients provided with the system, including the standard X11 clients, have been configured to use colors defined by the current palette. If you change the color in a palette, or select an entirely new palette, all of the clients that you run adhere to the same color scheme.

X clients are configured to use **scocolor**'s palettes through "palette resource variables". You assign a palette resource variable to a color resource instead of specifying an actual color value for the resource. The X server replaces the palette resource variables with the actual colors that are configured by the current palette when a client is invoked.

Table 6-1, "Relationship between palette resource variables and color buttons" lists the palette resource variables and the color buttons to which they correspond:

Table 6-1 Relationship between palette resource variables and color buttons

Palette resource variable	Corresponding color button
scoBackground	Background
scoForeground	Foreground
scoTopShadow	Top shadow
scoActiveBackground	Active window
scoActiveForeground	Active foreground
scoActiveTopShadow	Active top shadow
scoAltBackground	Alternate background
scoHighlight	Highlight

Some of the client resource files in the */usr/lib/X11/sco/startup* and */usr/lib/X11/app-defaults* directories, such as *Pmwm* and *ScoTerm*, set color resources using the resource variables described above. Other clients do not set color resources, either because there is no resource file for the client, or the file simply does not include color resources. These clients draw their color designations from the global color resource file, *Colors*, located in */usr/lib/X11/sco/startup*. This file defines all of the basic color resources a client might need, using the appropriate palette resource variables.

The following example lists the resource settings specified in the *Colors* file:

```
*Background:                scoBackground
*Foreground:                scoForeground
*topShadowColor:            scoTopShadow
*bottomShadowColor:         Black
*activeBackground:          scoActiveBackground
*activeForeground:          scoActiveForeground
*activeTopShadowColor:      scoActiveTopShadow
*activeBottomShadowColor:   Black
*troughColor:               scoBackground
*armColor:                  scoHighlight
*highlightColor:            scoForeground
*selectColor:               scoForeground
*borderColor:               Black
```

For example, if you run the **xbiff** client, which does not have a resource file, the background of the **xbiff** window is assigned the color that is linked to the

scoBackground palette resource variable. If you are using a palette that sets window backgrounds to steel blue, then the background of the **xbiff** window is steel blue. If you run the **scoedit** client, which has a resource file in */usr/lib/X11/app-defaults* that sets the ***background** resource to **scoBackground**, then the background of the **scoedit** window is also steel blue.

While the **scocolor** client is generally the preferred method for customizing colors, there may be occasions where you need to use an actual color value for a resource specification. "Changing colors for individual users" (page 112) explains how to specify color values instead of palette resource variables.

Colormaps

Although it is possible to use specific color values to establish different colors for the Graphical Environment, you should avoid specifying too many colors this way. Depending on the number of colors you can display on your monitor at the same time, there are limits to the number of resources you should set with color values.

The number of colors you can display on your screen simultaneously depends on the capabilities of your hardware, and, therefore, your X server. Some graphics adapters are capable of displaying only 16 unique colors, while others can display 256 distinct colors. These hardware dependencies determine whether your system has a 16-color or a 256-color X server.

The X server uses a colormap to keep track of its color usage. A "colormap", sometimes called a color lookup table, is simply an array of color cells. Each color cell contains the RGB values for a particular color. A 16-color server uses a colormap with 16 color cells, and a 256-color server uses a 256-cell colormap.

When the X server is first started, two colors, black and white, are automatically loaded into the first two cells of the colormap. If you are running the **scosession** session manager, a color palette daemon allocates the next eight cells and assigns the values of the current color palette to these cells.

If you are using a 16-color X server, you only have six additional color cells in the colormap that you can fill. These cells are allocated to clients on a first-come, first-serve basis. If you run an application whose color resources are defined by palette resource variables, no additional cells are filled. But if an application defines a specific color value, even if that color is part of the current color palette, one of the remaining six color cells is assigned that color. In most cases, other applications can then use this color cell as well.

The six extra color cells are only deallocated when the last application specifying one or all of the cells is terminated. Therefore, it is very easy to use up these last six cells if you specify many unique color resources. When all 16

cells of the colormap are allocated for a 16-color X server, no new color designations can be made until a cell is freed. If you make a color designation that cannot fit in the colormap, you see an error message that indicates there are no free color cells. The X server then tries to assign a color from the colormap that is closest to the color you originally wanted.

Of course, if you have a 256-color server, your colormap is considerably larger and you have more latitude for setting color resources with specific colors.

See also:

- "The SCO Merge and SCO Wabi colormaps" (this page)

The SCO Merge and SCO Wabi colormaps

While most applications are designed to share the colors in a colormap, some applications require their own colormaps. This is the case for SCO Merge and SCO Wabi.

SCO Merge requires the use of the 16 standard DOS colors. If you are using a 256-color X server, the standard colormap is probably large enough to accommodate the colors you usually use and the 16 colors for DOS. However, if you are using a 16-color X server, unavoidable color conflicts occur when running SCO Merge.

If you run a SCO Merge session so your DOS application fills the entire screen, even on a 16-color X server, you can avoid these color conflicts. The server simply loads the DOS colormap and the DOS program can access any of the colors it needs.

When you want to run a DOS application in a window, however, you must take certain steps to avoid color conflicts. One possible solution to this problem is to use **scocolor** to switch to the DOS Primary Colors palette before starting a SCO Merge session. This palette provides eight of the primary colors most often used by DOS programs. In most cases, these eight colors result in readable text for all windows. However, there may still be cases where you get unreadable windows.

SCO Wabi also requires the use of its own colormap. See Appendix B, "Color from Windows to the SCO Wabi program" in the *SCO Wabi User's Guide* for more information.

See also:

- *SCO Merge User's Guide*
- *SCO Wabi User's Guide*

Changing colors for the entire system

In general, it is not recommended that you define color resources with specific color values on a system-wide level. These specific color settings are loaded into the colormap, in addition to the two cells allocated by the X server for black and white and the eight cells allocated for the palette colors. If your system uses a 16-color X server and users are also defining unique color resources, the likelihood of filling the colormap is very high.

However, there are two color configuration tasks that are appropriate for a system administrator to do on a system-wide level: changing a color in an existing palette (this page) and adding new palettes to the system (page 111).

See also:

- "Creating a new system-wide palette" (page 111)
- "Changing colors with the Color control" (page 27)
- "The SCO Merge and SCO Wabi colormaps" (page 108)

Changing colors in an existing palette

You can change the colors in any existing palette so that the color schemes of the palettes better suit the needs of your users. You must be logged onto the system as *root* to perform this task.

To change a color in an existing palette, perform the following steps. For more information on each of these steps, see the sections immediately following this procedure.

1. Select the color(s) that you want to replace the color settings in the palette. Use one of the following methods to aid your choice:
   - Run the **scocolor** client to preview the available colors. When you decide on a color, note the Red, Green, and Blue decimal values for that color.
   - Examine the *rgb.txt* file in */usr/lib/X11* for a color you want to use. Note the Red, Green, and Blue decimal values for that color.
2. In the system *palettes* file in */usr/lib/X11/sco/ScoColor*, locate the line that corresponds to the color you want to change and replace it with the RGB values for the new color.

Step 1: Selecting a replacement color

The **scocolor** client is an excellent tool for previewing different colors and determining how they actually appear on your monitor. The Red, Green, and Blue decimal values that are displayed above the RGB slider bars correspond directly to the RGB values for the colors in the database.

When you decide on a particular color, note the RGB decimal values for the color. You need to specify these values to replace the old color.

A quicker way to determine the RGB values for a color is to examine the *rgb.txt* file in */usr/lib/X11*. This approach is most useful if you already know the color that you want to use but do not know the color's RGB values.

NOTE Choosing a color from the *rgb.txt* file based simply on the color's name may result in a disappointing appearance. Because of differences in graphics adapters and monitors, colors are not always reproduced in the way you might expect. For example, a color named Lemon Chiffon might seem more beige than yellow on some displays.

Step 2: Adding the new color to the palette

All of the default, system-wide palettes are located in the *palettes* file in */usr/lib/X11/sco/ScoColor*. Palette definitions in this file consist of nine lines of information. The following example shows the definition for the Tropics palette and a description of each line in the definition:

Tropics			Name of palette
62	243	220	Background color (RGB values)
0	52	44	Foreground color (RGB values)
188	255	247	Top shadow color (RGB values)
255	255	0	Active background color (RGB values)
0	71	67	Active foreground color (RGB values)
255	255	255	Active top shadow color (RGB values)
255	124	171	Alternate background color (RGB values)
233	160	106	Highlight color (RGB values)

Locate the section that defines the palette you want to modify. Then replace the line that contains the color you want to change with the RGB values for your new color choice. Use tabs to separate the three columns.

NOTE You cannot comment out lines within a palette definition. If more than nine lines exist for a palette, **scocolor** cannot access any of the palettes that follow the altered palette definition.

If you want to save a copy of a default palette before altering it, either make a backup copy of the *palettes* file or create a new palette, (page 111) based on the default palette, from within **scocolor**.

Creating a new system-wide palette

You can add new palettes to your system that all users can access. Unfortunately, system administrators cannot use the **scocolor** client to create system-wide palettes. The **scocolor** application always stores new palettes in *.odtpref/ScoColor*, in the user's home directory. If you run **scocolor** as *root*, palettes you create are stored in the *.odtpref* directory in /, not in the system-wide palettes file. Because of this, you need to merge palettes you create into the system-wide palettes file, after you are finished using **scocolor**.

To create a new palette that all users can select through the **scocolor** client, perform the following steps. You must be logged onto the system as *root* to perform this task. For more information on the steps in this procedure, see the sections immediately following this list.

To create a new palette that all users can access through **scocolor**:

1. Run the **scocolor** client and create the new palette.
2. Append the new palette, located in */.odtpref/ScoColor*, to the system-wide palette file, */usr/lib/X11/sco/ScoColor/palettes*, by entering the following command:

   cat /.odtpref/ScoColor >> /usr/lib/X11/sco/ScoColor/palettes

Step 1: Creating the new palette

To create a new palette, first run the **scocolor** client. From the **scocolor** window, click on the **Add palette** button. You are prompted to supply a name for the new palette.

The new palette inherits the colors of the current palette. If you want to build a new palette using an existing palette as a baseline, select that palette before clicking on **Add palette**.

Apply the desired colors to the eight palette buttons. When you finish creating the palette, click on **OK** and then exit **scocolor**.

See also:

- "Changing colors with the Color control" (page 27)

Step 2: Adding the new palette

When you create a palette as *root*, the palette definition is stored in a file called *ScoColor*, in the */.odtpref* subdirectory. To make the new palette available to all users, you need to incorporate the palette definition into the system-wide palette file, *palettes*, located in */usr/lib/X11/sco/ScoColor*.

Enter the following command to add the new palette definition to the end of the *palettes* file:

cat /.odtpref/ScoColor >> /usr/lib/X11/sco/ScoColor/palettes

After you append the new palette, it is advisable to edit the *palettes* file and verify that the format of the file is acceptable. There cannot be any blank lines between palette definitions. Also, if there are additional palette definitions in the */.odtpref/ScoColor* file that you do not want other users to access, delete the lines for those palettes from the system-wide file.

Changing colors for individual users

In general, users should use the **scocolor** client (page 27) when they want to modify the colors that are used by the Desktop and other clients. However, there may be circumstances where a user wants to change the color for a particular aspect of the Graphical Environment display without changing the colors defined in the current palette. For example, a user may run the **scoedit** client often and want the background of the **scoedit** window to be a unique color so it stands out on the display. Another user might want the background of all non-active window frames to use a different color than the background of the windows themselves. Color changes of this kind should be made using an *.Xdefaults-**hostname*** resource file in the individual user's home directory.

Before you actually make any color resource specifications, however, you should be familiar with the information in Chapter 5, "Understanding resources" (page 79).

The following procedure can be undertaken by either *root* or an individual user. To change the color settings for a particular aspect of a client, perform the following steps. For more information on each of these steps, see the sections immediately following this procedure.

1. Create a file called *.Xdefaults-**hostname*** in the user's home directory.
2. Edit the *.Xdefaults-**hostname*** file and add the new color resource specification, using the following format:

   client*resource_name: color

   Save your changes and exit the resource file when you are finished.
3. If you created the *.Xdefaults-**hostname*** file as *root*, assign the appropriate user permissions to the file:

   chown *username* .Xdefaults-*hostname*
   chgrp *groupname* .Xdefaults-*hostname*
4. Start a Graphical Environment session.

Step 1: Creating an .Xdefaults-hostname file

Individual users can assign their own color values, instead of palette resource variables, to color resources. A resource that uses a specific color value always has precedence over the same resource set with a palette resource variable.

Specific color resource settings are placed in a file called *.Xdefaults-**hostname***, where ***hostname*** is the name of the host, or machine, where the client is running. This file must be located in the user's home directory.

You can add an *.Xdefaults-**hostname*** file to a user's home directory in one of two ways:

- Create a file named *.Xdefaults-**hostname*** and then add the desired color specifications to the file. This approach is most useful if you are only making a few color designations.
- Copy one or more of the client default files from */usr/lib/X11/app-defaults* to the user's home directory, merging the files into a single file called *.Xdefaults-**hostname***. You can then use the file as a template, deleting resources you do not want to change and entering specific color values for the color resources you do want to change.

NOTE Lines in the resource files in the *app-defaults* directory do not specify the client, because each file applies to only one client. If you create an *.Xdefaults-**hostname*** file by copying the relevant lines from a file in the *app-defaults* directory, be sure to add the client's class or instance name to the beginning of each line if it is not already there. Otherwise the color resource specification will affect all clients.

When the user invokes a client, the client checks to see if an *.Xdefaults-**hostname*** file exists in **$HOME**. If such a file does exist, the resource values specified in the user resource file take precedence over any values assigned to the same resource in the resource database.

See also:

- "Methods for specifying resources" (page 87) for more information on the *.Xdefaults-**hostname*** file

Step 2: Setting the color resource

As with all resources, color resource specifications must use the correct format:

client*resource_name: color

client refers to the client you want to affect. You can supply either the client's binary or class name. ***resource_name*** is the actual resource variable you want to define. You can use either the resource's class or instance name. ***color*** is either the name of the color you are selecting or the color's hexadecimal RGB values. If you enter a color name, it must match an entry in the *rgb.txt* file. Hexadecimal RGB values, (page 101) however, do not have to exist in the color database.

There are many resources that control the colors used for the Graphical Environment. Table 6-2, "Common color resources" lists the most commonly used color resources. For a more complete list of color resources that are valid for a client, refer to the client's manual page.

Table 6-2 Common color resources

Resource name	Default value	Effect
Class: Foreground		
*foreground	scoForeground	Color of text in windows
*activeForeground	scoActiveForeground	Color of text in active window frame
*bottomShadowColor	Black	Bottom shadow color of window frame
*activeBottomShadowColor	Black	Bottom shadow color of active window frame
*normal*foreground	Black	Color of text in Desktop icon labels
Class: Background		
*background	scoBackground	Window background color
*activeBackground	scoActiveBackground	Color of background in active window frame
*topShadowColor	scoTopShadow	Top shadow color of window frame
*activeTopShadowColor	scoActiveTopShadow	Top shadow color of active window frame
*back*background	scoBackground	Directory window background color
*desktop*back*background	scoAltBackground	Desktop background color
*normal*background	White	Desktop icon label background color

Note that the resources in this table are grouped by class. If desired, you can set all of the resource instances in a class to the same color value simply by specifying the class in the *.Xdefaults-**hostname*** file. For example, to set all instances of the Background class to yellow, specify the following:

client*Background: yellow

Whether you are replacing a palette resource variable with a specific color name or specifying a new, more specific resource, the resource line should use the same format. For example, to specify that the background for a **scoedit** window should be red, add the following resource line to the *.Xdefaults-**hostname*** file:

```
ScoEdit*background:  red
```

See also:

- Chapter 5, "Understanding resources" (page 79) for more detailed information on specifying resources

Step 3: Assigning correct ownership permissions

If you generated an *.Xdefaults-**hostname*** file for a user from the *root* account, whether by creating the file or by copying the file from files in */usr/lib/X11/app-defaults*, you must assign the file the correct ownership permissions. Run the **chown** command to assign the correct owner and the **chgrp** command to assign the correct group to the *.Xdefaults-**hostname*** file.

If you created your own *.Xdefaults-**hostname*** file, you can omit this step. Your ownership permissions are already correct.

Step 4: Starting a Graphical Environment session

To see your new color settings, start a Graphical Environment session, either through **scologin** or by running **startx** from the command line.

Run the desired clients. When a client is started, it reads the **$HOME**/*.Xdefaults-**hostname*** file for your personal resource specifications. The new color settings are noted and specified colors are displayed accordingly.

If you created your own *.Xdefaults-**hostname*** file while running a Graphical Environment session, and you want the new color values applied to clients that are currently running, you need to restart the clients.

If you want the Desktop client to reflect your new color specifications, you must end your session and then start the X server again, either by logging in through a **scologin** window or by running the **startx** script from the command line.

Setting colors from the command line

You can specify colors for different elements of a client's display on the command line. If you specify a color resource from the command line, however, the color is only set for the current client session. Subsequent sessions return to the default color specification for the client.

NOTE If you are using **scosession** and you end your Graphical Environment session with a client that was run from the command line still open on the display, the client is restored when you resume your session. Any command line options that were used to define the client are also restored. In this way, command line options can define client behavior on a more permanent basis.

To set different color characteristics for a client from the command line, open a **scoterm** window and use one of the following command line options:

- the **-xrm** option (this page) sets an actual color resource

  ***client* -xrm *'client*resource_name: color'* &**

- the **-bg** and/or **-fg** options (page 118) sets specific background and foreground colors

  ***client* -bg *color* -fg *color* &**

The -xrm option

The **-xrm** option allows you to set any resource value, including color specifications, from the command line. You must enter the resource specification as well as the desired color when using this option. You can specify a color by using either its database name or its hexadecimal RGB values. (See "The RGB and HSV color models" (page 101) for information on how to set resources with hexadecimal RGB values.)

For example, to change the color of the active window frame for **scoterm**, enter the following at the command line:

scoterm -xrm 'ScoTerm*activeBackground: red' &

When using the **-xrm** option, you should follow these rules:

- You must quote a resource specification using the single quote character (').
- You can specify more than one resource value from the command line at the same time. You must enter the **-xrm** option for each specification you make:

  scoterm -xrm 'ScoTerm*activeBackground: red' -xrm 'ScoTerm*activeForeground: navy' &

A color resource specified at the command line with the **-xrm** option will not take effect if X has already recognized a color resource, either in the resource database or in an *.Xdefaults-**hostname*** file, that takes precedence. (See "Precedence rules for resource specifications" (page 87) for information on the precedence of resource specifications.) For example, you might create an *.Xdefaults-**hostname*** file with the following resource specification:

```
ScoTerm*mainMenu*background: green
```

In this case, the following command line specification would not cause the **scoterm** client to display an aquamarine-colored main menu:

```
scoterm -xrm 'ScoTerm*background: aquamarine' &
```

Because the **ScoTerm*mainMenu*background** designation is more specific, the **ScoTerm*background** entry on the command line cannot change the background color of **scoterm's** main menu.

To override the resource database and get the new menu background color, you would need to use a resource equal to or more specific than the default setting. For example, the resource **ScoTerm*mainMenu*background** would provide the desired color change.

The -bg and -fg options

Most clients also accept the **-bg** and **-fg** command line options. These options control the following elements of a client's display:

-bg sets the client window's background color

-fg sets the client window's foreground color (usually the color of the text in the window)

You can set these options using either a color's database name or its hexadecimal RGB values. (page 101)

For example, to run the **xclock** program with a plum background and navy text, enter the following at the command line:

```
xclock -bg plum -fg navy &
```

If a color name includes blank spaces, you need to include quotation marks around the name. For example, to specify light blue as the background for **xclock**, enter:

```
xclock -bg "light blue" &
```

Adding custom colors to the database

By experimenting with the RGB slider bars in the **scocolor** client or with RGB values on your own, you can create colors and shades that are not included in the color database. If you create custom colors that you think might be useful, you should add them to the color database.

Once incorporated in the database, the new colors become available to all users on your system and can be selected through the **scocolor** client in exactly the same way you would select any of the other default colors in the database. You must be logged into the system as *root* to perform this task.

To add a custom color to the color database, perform the following steps. For more information on each of these steps, see the sections immediately following this procedure.

1. Edit the *rgb.txt* file in */usr/lib/X11* and add your new color, using the following format:

   Red_value Green_value Blue_value colorname

   Save and exit the file when you are finished.

2. From the */usr/lib/X11* directory, run the **rgb** command to recompile the color database:

   rgb < rgb.txt

3. Run the **showrgb** command to check that the new color is now listed in the color database:

   /usr/bin/X11/showrgb | more

Step 1: Adding the new colors to rgb.txt

To add a new color to the color database, first include the color name and the color's RGB decimal values to the *rgb.txt* file, located in */usr/lib/X11*. Use the following format when adding the color to the file:

Red_value Green_value Blue_value colorname

Red_value, ***Green_value***, and ***Blue_value*** represent the percentages of red, green, and blue, respectively, that are used to create the color. These three entries must be specified with a decimal value between 0 and 255, where 255 indicates a color is at full intensity. ***colorname*** indicates the name of the color that is associated with the related RGB decimal values. You can use blank spaces or tabs to separate the four fields in an RGB designation.

NOTE Before you modify the *rgb.txt* file, it is recommended that you make a backup copy of the file. This way, you can return to the original color database, if desired.

If the name you choose for your new color has more than one word or more than one spelling (for example, gray and grey), you are advised to create several different entries in the *rgb.txt* file to account for these differences. This gives users a range of flexibility when entering color names in resource files. Use the same RGB values for the alternate entries.

For example, you might create a custom color that you want to name "light yellow silk." You might make the following entries in *rgb.txt*:

243 255 162 light yellow silk
243 255 162 LightYellowSilk

In the same way, if you create a color that you want to name "blue gray," you should make the following entries:

94 139 157 blue gray
94 139 157 BlueGray
94 139 157 blue grey
94 139 157 BlueGrey

You can place your custom color definitions anywhere you want in the *rgb.txt* file. However, the default colors in this file are arranged in groups of similar shades. For example, all shades of blue are grouped together; all shades of pink are grouped together, and so forth. Therefore, it may be advisable to add your new colors within the shade groupings that best match the colors.

When you finish adding your color definitions to the *rgb.txt* file, save your changes and exit the file.

Step 2: Running rgb

After you add your new color definitions to the *rgb.txt* file, you need to run the **rgb** program to recompile the color database.

The **rgb** program uses the information in *rgb.txt*, including your new color definitions, as input and then creates new versions of the *rgb.dir* and *rgb.pag* files. You should change to the */usr/lib/X11* directory to run this command.

Enter the following command to recompile the color database:

```
rgb < rgb.txt
```

NOTE The **rgb** program does not make backups of the *rgb.dir* and *rgb.pag* files before it overwrites them with the new versions. If you want to preserve your old color database files, make sure you create backup versions of *rgb.dir* and *rgb.pag* before you run **rgb**.

Step 3: Running showrgb

After you recompile the color database, you should check to make sure the new colors are actually recognized in the database.

The **showrgb** command examines the *rgb.dir* and *rgb.pag* files and constructs a list of all the colors it finds in the database. The command then displays this list on your screen. The list includes the names assigned to the colors and the RGB values used to generate the colors. Note that all the color names that are displayed by this command are in lowercase letters only, even if the database recognizes uppercase versions of a color.

Because the colors that are available in the database number in the hundreds, you should use a paging command, such as **more** or **pg**, when you run **showrgb**:

/usr/bin/X11/showrgb | more

Your custom colors are now available to all users on the system. All users can select any one of the new colors from the list of available colors displayed by **scocolor**.

Examples of changing colors

This section provides two examples that tie together many of the concepts and procedures discussed in this chapter.

- The first example (this page) describes how to add a custom color to the color database and then change a default palette so it uses the new color.
- The second example (page 123) explains how a user might customize the colors that are used by the Graphical Environment.

Example 1: Using custom colors in default palettes

Let's assume you are an administrator for a system whose X server and clients are accessed by several users. Your users have asked you to replace the window background color in the default, system-wide palette called Northwoods with a more muted shade of blue, because the current color, as it displays on your monitors, is too bright.

After some experimentation, you decide that none of the shades of blue available in the database will satisfy all of your users, so you create a custom color to resolve the problem. This example covers all aspects of incorporating this color into the Northwoods palette, including how to:

- add the custom color to the color database, and
- add the color to the system-wide palette file.

The following steps result in a modified default palette:

1. Log into the system as *root*. If you did not log into the *root* account through a **scologin** window, start a Graphical Environment session by entering:

   startx &

2. From a **scoterm** window, change to the */usr/lib/X11* directory and open the *rgb.txt* file for editing.
3. Because your new color is a shade of blue, locate the section of the *rgb.txt* file that defines blue colors. This section starts with the following entry:

   ```
   25 25 112      midnight blue
   ```

4. Within the blue section, open a line and enter the RGB values and the name for your custom color. Color names are separated from the RGB values by a tab.

   For this example, enter:

   184 216 255 alpine blue

   Because the color name in this example consists of two words, you should also enter the following:

   184 216 255 Alpine Blue

5. Save and exit the *rgb.txt* file.
6. Recompile the color database with the **rgb** command so the X server can recognize the new color:

   rgb < rgb.txt

7. When your prompt returns, run the **showrgb** command to check that the new color is now listed in the database:

   showrgb | grep alpine

   This command produces the definition line for the Alpine Blue color. Remember to use lowercase letters when **grep**ing for a color name with the **showrgb** client.

8. You are now ready to modify the Northwoods palette and add your new color. Change directories to */usr/lib/X11/sco/ScoColor* and look for the file named *palettes*. This file contains all of the system-wide palette definitions.
9. Make a backup copy of the *palettes* file, in case you should want to return to the original Northwoods palette some day:

   cp palettes palettes.old

10. Open the *palettes* file for editing and search for the Northwoods palette definition. You should see:

```
Northwoods
172     199     224
52      0       0
255     241     241
158     228     151
16      86      124
255     238     255
255     220     180
255     148     48
```

11. The window background color is defined by the **Background** color button. The color for this button is specified in the second line of the Northwoods palette definition. Replace the second line, which reads `172  199  224`, with the RGB values for the new color, using tabs to separate the three columns:

    184 216 255

12. Save and exit the *palettes* file.
13. To verify that you correctly modified the Northwoods palette, run the **scocolor** client and select the Northwoods palette. You should see a much more pleasing shade of blue for the backgrounds of your windows.

Example 2: Customizing colors with resources

For this example, let's assume that you use the **scomail** program frequently and want to quickly distinguish its window from other windows on your screen. A good way to do this is to change the colors that are used by the **scomail** window without changing the colors that all other windows use.

This example covers how an individual user can make this kind of color change without changing colors in the current palette. It is not recommended that a system administrator change colors on a system-wide level in this fashion, because possible colormap conflicts may arise for users.

The following steps result in a **scomail** client that displays colors that are unique from those used by the rest of the Graphical Environment.

1. Log into the system. If you did not log in through a **scologin** window, start a Graphical Environment session by entering:

   startx &

2. Launch a **scoterm** window by double-clicking on the **UNIX** icon.

3. In your home directory, create a file called *.Xdefaults-**hostname***, where ***hostname*** is the name of your system. This example uses *scooter* as your host name.
4. Edit your *.Xdefaults-scooter* file and add the following resource specifications:

   ScoMail*background: blue
   ScoMail*activeBackground: yellow

   These resource specifications tell the X server that the background of the **scomail** window should be blue and the background of the window frame should be yellow when the **scomail** window is active. With these color settings, your **scomail** window will always be easy to distinguish.
5. When you finish entering the color resource specifications, save and exit the *.Xdefaults-scooter* file.
6. To verify that the X server recognizes your color specifications, run the **scomail** client. You should see a window with a blue background and a yellow active border.

Chapter 7

Changing fonts

The SCO OpenServer system includes a variety of fonts for text that display when you use the Graphical Environment. Most clients allow you to specify the font that is used to display text in windows, in menus and icon labels, or in any other text field. You make these font specifications through resources.

Specifically, this chapter describes:

- background information about fonts (page 126)
- using the font server (page 130)
- configuring the font server (page 133)
- listing available fonts on your system (page 138)
- previewing a specific font (page 141)
- specifying system-wide and personal fonts (page 143)
- creating font aliases (page 151)
- adding new fonts to your system (page 152)

There is also an example (page 156) at the end of this chapter that helps tie together many of the concepts and procedures discussed in this chapter.

See also:

- "Changing Desktop fonts" (page 30) for information on changing Desktop fonts using the **Preferences Editor**
- Chapter 5, "Understanding resources" (page 79) for a detailed explanation of resources

About fonts

The supported fonts on your system form a font database. By default, all font files are located in subdirectories of the */usr/lib/X11/fonts* directory. The five standard subdirectories are:

misc	fixed-width fonts and cursor and glyph fonts
75dpi	fixed- and variable-width fonts for 75 dots per inch (dpi) displays
100dpi	fixed- and variable-width fonts for 100 dots per inch (dpi) displays
Speedo	contains outline fonts for the Bitstream(r) Speedo rasterizer. A single font face, in normal, bold, italic, and bold italic, is provided, contributed by Bitstream, Inc.
Type1	scalable, PostScript Type1 fonts

Each of these directories includes a *fonts.dir* file, which together provide a database for the X server. When the X server needs a font, it uses these files to locate it.

The *fonts.dir* files contain two columns of information: the first column lists all of the font files in the directory and the second column lists the full font names associated with the font files. The first line in *fonts.dir* lists the number of fonts available in that directory.

By default, the directories *misc*, *75dpi*, *100dpi*, *Speedo*, and *Type1*, in */usr/lib/X11/fonts* constitute the font search path. The X server uses the search path to determine where on the system it should look for font files. If needed, other directories can be added to the font search path. For information about adding new directories to the X server's font search path, see "Adding a font to your system" (page 152).

The font search path is defined by the */usr/bin/startx* script. You can see the font search path by typing the following line at the shell prompt:

xset q

Fonts can be specified with a full name, wildcards, or aliases. For example, a 14-point courier bold font could be specified in any of the following ways:

```
-adobe-courier-bold-r-normal--14-140-75-75-m-90-iso8859-1
'*courier-bold-r*140*'
courierB14
```

The first line is the full name of the font, the second line uses wildcards, and the third line uses an alias.

See also:

- "Font names" (this page)
- "Using wildcards" (page 128)
- "Font aliases" (page 129)
- "The font server" (page 130)

Font names

The full name of a font is a string such as the following:

```
-adobe-courier-bold-o-normal--10-100-75-75-m-60-iso8859-1
 1     2       3    4 5      6 7   8  9 10 11 12    13  14
```

This is the full name of the *courBO10.pcf* font in the */usr/lib/X11/fonts/75dpi* directory.

A font name is comprised of 14 fields, with a hyphen (-) separating each field. The fields have the following meanings:

1. foundry that digitized and supplied the font
2. font family, based on standard typesetting names for various fonts. The major families include Charter, Courier, Helvetica, Lucida, Lucidabright, Lucidatypewriter, New Century Schoolbook, and Times.
3. font weight, usually "bold" or "medium"
4. font slant, usually "i" for italic, "o" for oblique, or "r" for Roman.
5. font proportionate width, as set by the foundry. Most fonts use "normal"; other possible values are "condensed", "narrow", and "double width."
6. add_style_name; used to provide additional information not covered in the rest of the font name, for example "serif," or "decorated." Typically this field is not used.
7. font size in pixels. This measurement depends on the resolution of the font. For example, if the font has a resolution of 100 dots per inch (dpi), a 12-point font has a pixel size of 17. If a font has a resolution of 75 dpi, there are fewer dots per pixel, so a 12-point font has a pixel size of 12.
8. font size in tenths of a point. A point is a printer's unit that measures 1/72 of an inch.
9. horizontal resolution in dots-per-inch
10. vertical resolution in dots-per-inch. Horizontal and vertical figures are required because a screen may have different capacities for horizontal and vertical resolution.
11. font spacing. The valid values are "m" for monofont (or fixed-width) and "p" for proportional.

12. average (mean) width of all characters in the font, measured in tenths of a pixel (60)
13. font character set registry. This specification refers to the ISO 8859 character set. The ASCII character set is a part of ISO 8859.
14. character set encoding. "1" is the Latin character set.

You should also be aware of the following when dealing with font names:

- Font names are case sensitive. Whenever you enter a font name, be sure to use the correct combination of upper- and lowercase letters.
- No spaces can be used anywhere in the font name, except when they are contained within the font family specification, in which case they must be enclosed in double quote characters.
- Terminal emulator clients (**scoterm, xterm**) require fixed-width (monofont) fonts. Most other clients can use either a fixed-width or a variable-width font, depending on your preference. Fixed-width fonts resemble the fonts used on non-graphics terminals, whereas variable-width fonts resemble the fonts used in typeset text.

Using wildcards

Wildcard characters can be used to shorten the string needed to specify a font. The asterisk (*) and question mark (?) serve as wildcard characters in much the same way they do for the operating system: an asterisk represents any combination or variety of characters and a question mark represents any single character.

To prevent the shell from interpreting the wildcards, either enclose the entire font name in quotes or use a backslash before each wildcard character. For example, the two following lines are valid wildcard representations of the font discussed in the previous section. In this example, the font is being specified for the **scoterm** client using the **-fn** command line option.

```
scoterm -fn '*courier-bold-o-*-100*' &
scoterm -fn \*courier-bold-o-\*-100\* &
```

You should be aware of the following when using wildcards in font specifications:

- Font specifications using wildcards should explicitly name enough parts of the font's full name to create an unambiguous reference. Wildcarded font specifications usually specify the font family, weight, slant, and point size.
- If the wildcarded font specification is ambiguous (in other words, more than one font matches the specification), the X server chooses which font to use. If fonts from more than one font directory match the wildcarded

name, the server chooses a font from the directory that occurs first in the font path. In other words, fonts defined in the *75dpi* directory are chosen rather than fonts in the *100dpi* directory.

- It is better to match the point size field (which is measured in tenths of a point) than the pixel field. This allows your wildcarded font name to work properly with monitors of different resolutions, because the wildcarded font name matches either the *75dpi* or *100dpi* font set.

Using font wildcards is a convenient method of specifying fonts. However, it can also lead to unexpected results, so it is recommended that you test your shortcut thoroughly before making it generally available. In many cases, font aliases, described in the next section, provide a safer and easier way to shorten your references to full font names.

Font aliases

Another way to abbreviate font names is to create a font alias, a shorter name that is used as an alternative to the full font name. Font aliases are specified in a file called *fonts.alias*. By default, the three directories *misc*, *75dpi*, and *100dpi* in */usr/lib/X11/fonts* each contain a *fonts.alias* file. If you add new font directories to your system, you can create *fonts.alias* files for those directories too.

The default *fonts.alias* files already contain several aliases. You should study these files as you will probably find the aliases are more convenient to use then specifying full font names, or even wildcarded names. Of course, you are free to change the existing aliases, add new aliases, or even replace the entire file, although this should be done with caution. Generally, it is a good idea to simply add new aliases to the existing files.

NOTE The first two entries in the *fonts.alias* file in */usr/lib/X11/misc* should not be changed or removed. These entries are shown below:

```
fixed    -misc-fixed-medium-r-semicondensed--13-120-75-75-c-60-iso8859-1
variable -*-helvetica-bold-r-normal-*-*-120-*-*-*-*-iso8859-1
```

Many client font resource specifications use these aliases and if you remove them the clients will not be able to display a font.

The *fonts.alias* files use a two-column format, much like the *fonts.dir* file. The first column contains the aliases and the second column contains the full names of the fonts being aliased.

When creating aliases, you should be aware of the following:

- To specify an alias that contains spaces, enclose the alias in double quotation marks.
- To include double quotation marks, backslashes, or other special characters in your alias, precede each special character with a backslash.
- If you create an alias or an alias file while running an X server, the server does not automatically recognize the additions. You must reset the server to activate new aliases.

The font server

The font server, **fs**, provides a way to minimize disk use and standardize available fonts for a network of X terminals and SCO OpenServer systems. The font server allows you to install fonts on a single machine, then access those fonts from any capable X server on the network.

The font server uses the standard X fonts. The same rules about constructing and using font files, directories and resources with the standard font system apply to the font server. If you are running the font server, you can still use local fonts by adding the appropriate font directories to your font path.

By default, the X server does not use the font server. You must start it from the command line, or by editing a configuration or startup file. See "Using the font server" (this page) for more information.

Using the font server

The font server can be started in one of several ways:

- from the command line (useful for testing the font server)
- from a system startup file such as */etc/rc2.d/S91fontserv*;

There are two parts to the process of using the font server:

1. Start the font server on a machine with fonts installed. That machine will provide fonts, via the font server, to other machines on the network.
2. Configure the X servers on the network to request fonts from the font server.

Each of these steps are described in the upcoming sections on starting the font server.

There are several ways you can configure font services, including setting up multiple font server machines, setting an optimum number of servers to quickest possible access to fonts, limiting the number of fonts available, and changing the TCP ports used by the font server.

See also:

- "Configuring the font server" (page 133)
- "Running the font server from the command line" (this page)
- "Using the font server from scologin" (this page)
- "Using the font server from startx" (page 132)
- "The font server" (page 130)

Running the font server from the command line

To start the font server from the command line, at a shell prompt enter the following:

/etc/fontserv start

This starts the font server immediately, and uses the default configuration file */usr/lib/X11/fs/config*.

NOTE Starting the font server from the command line is not recommended for everyday use. If you want the font server to run consistently on the system, configure it to start when the system boots using **/etc/fontserv enable**, which automatically creates the */etc/rc2.d/S91fontserv* file.

Using the font server from scologin

If you want **scologin** to control whether the X server uses the font server:

1. Edit the file */usr/lib/X11/scologin/Xservers*.
2. Find the line that contains the X startup command. For example:

   ```
   :0 local /usr/bin/X11/X :0 -crt /dev/tty02
   ```

   Change this line so that it starts the font server:

   ```
   :0 local /usr/bin/X11/X :0 -crt /dev/tty02 -fp tcp/server:7000
   ```

   Replace ***server*** with the name of the server on which the font server will run.
3. Restart **scologin**.

Using the font server from startx

You can configure the font server to start when a user starts the X server using **startx**. Follow these steps to start the font server either from the system file *sys.startxrc* or from a user's **$HOME**/*.startxrc* file:

1. Edit the appropriate file, either */usr/lib/X11/sys.startxrc* or the *.startxrc* file in a user's home directory.
2. Place the following line after the comment lines, but before any clients are run:

   ```
   xset fp=tcp/server:port
   ```

   The ***server*** is the name of the machine on which the font server is running, and ***port*** is the TCP port on which font server is broadcasting (by default 7000). For example, this line causes the X server to request fonts from the server **boston** which is running a font server that is broadcasting on TCP port 7000:

   ```
   xset fp=tcp/boston:7000
   ```

3. If the X server is already running, shut it down, then restart it using the **startx** command.

Running the font server from system startup files

You can enable the font server to start whenever the system enters multiuser mode. This is useful if you want to run the font server on a system that will not be running the Graphical Environment. Follow these steps:

1. Log in as root.
2. Enter this command:

   fontserv enable

   The next time the system enters multiuser mode, the font server is automatically started.
3. If you want to start the font server immediately, enter this command:

   fontserv start

The **fontserv enable** creates the file */etc/rc2.d/S91fontserv*, which is a script that starts the font server when the system enters multiuser mode.

Configuring the font server

The font server can be configured using command-line options, or using a configuration file. You can change different aspects of the font server, including:

- the fonts that a server will make available to other font servers (this page)
- the default point size and resolutions the font server makes available to X servers (page 134)
- the hostnames from which the X server will receive fonts (page 134) (from font servers running on those hosts)
- the TCP port address that the font server uses (page 135)
- the number of connections to X servers that a given font server allows (page 136) and the action the font server takes when that number of connections is exceeded

In addition, you can set up your X server to use a mix of local fonts and remote fonts from a font server, (page 136) and even specify an alternate font server configuration file. (page 137)

Configuring available fonts

You can configure available fonts using a combination of:

- installing a specific set of fonts, and
- specifying which fonts the font server will provide

To specify which fonts are available from the font server, follow these steps:

1. Log in as root.
2. Edit the font server configuration file. By default this is */usr/lib/X11/fs/config*.
3. Change the fonts specified by the **catalogue** keyword. For example, to provide only **Speedo** and **Type1** fonts:

```
catalogue = /usr/lib/X11/fonts/Speedo,/usr/lib/X11/fonts/Type1
```

4. To make the changes take effect, use the **fontserv** command:

```
fontserv re-read
fontserv flush
```

With the **catalogue** keyword, you can also specify another font server. For example:

```
catalogue = /usr/lib/X11/fonts/Speedo,tcp/boston:7001
```

This sets the local font server's catalogue to include **Speedo** fonts and whatever fonts are provided by the remote host **boston**.

NOTE Exercise care when referencing other font servers. Font server connections place a heavy demand on network resources and bandwidth. Also, be careful not to let the number of references to other font servers become so large that your system font system becomes unmanageable.

Configuring default font size and resolutions

You can change the default point size of fonts provided by the font server. To do this, follow these steps:

1. Log in as root.
2. Edit the font server configuration file. By default this is */usr/lib/X11/fs/config*.
3. To change the default point size, edit the value of the keyword **default-point-size**. For example, to make the default point size 12pts, change the keyword line to this:

   ```
   default-point-size = 120
   ```

   You can only specify one default point size.
4. To change the default resolutions, change the **default-resolutions** keyword. For example, to specify resolutions of 75 dpi only, change the keyword to this:

   ```
   default-resolutions = 75,75
   ```

   To specify multiple resolutions, specify the resolution you want, each separated by a comma.
5. To make the changes take effect, run the **fontserv** command:

   fontserv re-read
   fontserv flush

Choosing a font server host

To change which font server an X server uses, change the X server startup command to specify the desired host. For example, if you are starting an X server from the *Xservers* file, and you want the server to obtain fonts from the host **boston**, follow these steps:

1. Log in as root.
2. Edit the file */usr/lib/X11/scologin/Xservers*.

3. Change the ***hostname*** variable to the desired host. For example, to connect to the host **boston** using TCP port 7002:

```
:0 local /usr/bin/X11/X :0 -crt /dev/tty02 -fp tcp/boston:7000
```

4. To make this change take effect, stop then restart the X server.

Note that in the *Xservers* file, you can set the font path to include not only a remote font server but also local system fonts. For example, in step 2, above, you could use the following font path:

```
-fp tcp/boston:7000,/usr/X11/fonts/Speedo,/usr/lib/X11/fonts/misc
```

This adds to the font path the directories *Speedo* and *misc* on the local system.

Changing font server TCP ports

To select a different TCP port for use with the font server, change the port number for both the font server and for any X servers that use the font server. For example, to change the port address for both the font server and the X server, follow these steps.

1. Log in as root.
2. Edit the file */usr/lib/X11/fs/config*.
3. On a line by itself, add the **port** keyword to specify the desired address:

```
port=7005
```

   The font server now listens on TCP port 7005 on the local host.

4. Reconfigure any X servers that use the font server. For example, if you specify the font server in the *Xservers* file:

```
:0 local /usr/bin/X11/X :0 -crt /dev/tty02 -fp tcp/boston:7005
```

   This causes the X server to connect to host **boston** on the TCP port 7005.

5. Shut down the X server, then shut down the font server (in this example, on the server **boston**).
6. Restart the font server on the machine **boston**, then restart the X server.

Configuring font server connection limits

It is possible to overload a given font server with requests for fonts. To prevent this from happening, you can change two aspects of the font server:

- a limit on the number of clients that can connect to the font server
- whether or not a font server will clone itself if the client limit is reached

For example, you can configure the font server to limit the number of clients that can connect to it at one time to 20. Then, if that limit is reached, clone another font server.

To do this, follow these steps:

1. Log in as root.
2. Edit the font server configuration file. The default is */usr/lib/X11/fs/config*.
3. Insert the keyword **client-limit**. Set it to 20 (or whatever value you want):

```
client-limit = 20
```

4. Reset the font server with the **fontserv** command:

fontserv reset

Using the font server and local fonts

You can use local fonts at the same time as fonts from a font server. You can do this in one of two ways:

- configure the X server to use a font server and local fonts, (this page) or
- use the **xset** to add the local fonts (or the font server fonts) (page 137)

Specifying multiple font sources with the X server

To configure the X server to use multiple font sources, including local and font server fonts, follow these steps:

1. Log in as root.
2. Edit the file */usr/lib/X11/scologin/Xservers*.
3. Find the line that starts the X server. For example:

```
:0 local /usr/bin/X11/X :0 -crt /dev/tty02
```

4. To the end of this line, add the font path option, **-fp** and the desired fonts (including the font server host and TCP port):

   ```
   -fp tcp/boston:7000,/usr/X11/fonts/Speedo,/usr/lib/X11/fonts/misc
   ```

   This sets the font path for the X server to include a font server running on the server **boston** and the local font directories *Speedo* and *misc*.

5. To make this change take effect, stop then restart the X server.

Specifying multiple font sources with xset

Follow these steps to specify multiple font sources (local fonts and remote font server fonts):

1. Log in as root.
2. Specify the font server in the X server startup file, for example */usr/lib/X11/scologin/Xservers*:

   ```
   :0 local /usr/bin/X11/X :0 -crt /dev/tty02 -fp tcp/boston:7000
   ```

   This causes the X server to request fonts from the font server running on the host **boston**.

3. In one of the **startx** configuration files, for example */usr/lib/X11/sys.startxrc*, place the following line:

   xset +fp ***font_location***

   Replace ***font_location*** with a list of directories that contain the fonts you want to use.

4. Stop the X server, then restart it.

Using alternate font server configuration files

You can use font server configuration files other than the default. This is useful for starting a custom font server.

The following steps show how to start a font server that reads a custom configuration file:

1. Create the custom configuration file. For example, the file might be called *myfont.config* and might reside in user **tammyr**'s home directory */u/tammyr*.
2. Edit the file from which the font server will start, for example *.startxrc* in user **tammyr**'s home directory. Add the command to start the font server with the custom configuration file:

   ```
   /usr/bin/X11/fs -cf /u/tammyr/myfont.config &
   ```

3. Stop the X server, then restart it.

Listing available fonts on your system

You can use either the **xlsfonts** or **fslsfonts** clients to display a list of all of the fonts that are currently available to your X server. The command you use depends upon whether or not you are using the font server, **fs**.

To display X server fonts, use **xlsfonts**. (this page) To display font server fonts, use **fslsfonts**. (page 139)

Listing X server fonts with xlsfonts

To display fonts that are available from the X server, enter the following command at the prompt in a **scoterm** window:

xlsfonts | more

This command is described in greater detail in the following section.

Running xlsfonts

When you run the **xlsfonts** client, you get a listing of the fonts that are available from the X server. Note that the **xlsfonts** client can only list fonts that are located in directories specified in the font search path. If you have font directories on your system that are not included in the font search path, the fonts located in those directories are not displayed.

Because there are so many available fonts, even if you only have the default fonts on your system, you should use a paging command such as **more** or **pg** when you run **xlsfonts**.

The **xlsfonts** client lists each font by its complete font name. For example, you might see output such as:

```
-adobe-courier-bold-o-normal--10-100-75-75-m-60-iso8859-1
-adobe-courier-bold-o-normal--11-80-100-100-m-60-iso8859-1
-adobe-courier-bold-o-normal--12-120-75-75-m-70-iso8859-1
-adobe-courier-bold-o-normal--14-100-100-100-m-90-iso8859-1
-adobe-courier-bold-o-normal--14-140-75-75-m-90-iso8859-1
-adobe-courier-bold-o-normal--17-120-100-100-m-100-iso8859-1
-adobe-courier-bold-o-normal--18-180-75-75-m-110-iso8859-1
-adobe-courier-bold-o-normal--20-140-100-100-m-110-iso8859-1
-adobe-courier-bold-o-normal--24-240-75-75-m-150-iso8859-1
-adobe-courier-bold-o-normal--25-180-100-100-m-150-iso8859-1
-adobe-courier-bold-o-normal--34-240-100-100-m-200-iso8859-1
-adobe-courier-bold-o-normal--8-80-75-75-m-50-iso8859-1
-adobe-courier-bold-r-normal--10-100-75-75-m-60-iso8859-1
-adobe-courier-bold-r-normal--11-80-100-100-m-60-iso8859-1
--More--
```

See also:

- **xlsfonts**(XC) manual page for more information on the **xlsfonts** client

  xfontsel(XC) manual page for information on how to use the **xfontsel** client to display the fonts that the X server recognizes

Listing font server fonts with fslsfonts

If you are using the font server (**fs**), use the following command to fonts available from the font server:

fslsfonts -server *hostname:port* | more

Replace ***hostname*** with the name of the host that is providing font service. Use **localhost** if you want to query the font server on the local machine. For the port address ***port***, use the default of **7000**, unless that default is changed.

This command is described in greater detail in the next section.

Running fslsfonts

When you run the **fslsfonts** client, you get a listing of the fonts that are available from the X server. The **fslsfonts** client can only list fonts that are located in directories specified in the font server configuration file */usr/lib/X11/fs/config*. If you have font directories on your system that the font server is not configured to use the fonts located in those directories are not displayed.

Because there are so many available fonts, even if the font server is configured for the default fonts on the system, you should use a paging command when you run **fslsfonts**.

The **fslsfonts** client lists each font by its complete font name. For example, you might see output such as:

```
-adobe-courier-bold-o-normal--10-100-75-75-m-60-iso8859-1
-adobe-courier-bold-o-normal--11-80-100-100-m-60-iso8859-1
-adobe-courier-bold-o-normal--12-120-75-75-m-70-iso8859-1
-adobe-courier-bold-o-normal--14-100-100-100-m-90-iso8859-1
-adobe-courier-bold-o-normal--14-140-75-75-m-90-iso8859-1
-adobe-courier-bold-o-normal--17-120-100-100-m-100-iso8859-1
-adobe-courier-bold-o-normal--18-180-75-75-m-110-iso8859-1
-adobe-courier-bold-o-normal--20-140-100-100-m-110-iso8859-1
-adobe-courier-bold-o-normal--24-240-75-75-m-150-iso8859-1
-adobe-courier-bold-o-normal--25-180-100-100-m-150-iso8859-1
-adobe-courier-bold-o-normal--34-240-100-100-m-200-iso8859-1
-adobe-courier-bold-o-normal--8-80-75-75-m-50-iso8859-1
-adobe-courier-bold-r-normal--10-100-75-75-m-60-iso8859-1
-adobe-courier-bold-r-normal--11-80-100-100-m-60-iso8859-1
--More--
```

If you want to view fonts on a specific server, use the **-server** option:

fslsfonts -server *hostname:port* | more

The ***hostname*** is the name of the host machine running the font server. The ***port*** option specifies the TCP port to which the font server connects. By default this is port 7000.

See also:

- **fslsfonts**(X) manual page

Previewing a specific font

You can display the complete character set of any valid font, using the **xfd** (X font displayer) client. This tool provides an excellent means of determining whether or not you might want to use a particular font for a client, an icon label, and so forth.

Once you have located a font that seems interesting, you can preview its characteristics using the following steps. For more information on each of these steps, see the sections immediately following this procedure.

1. Enter the following command at the prompt in a **scoterm** window:

   xfd -fn *fontname*

2. Click on the **Next Page** and **Prev Page** buttons to scroll through multiple screens, if necessary.
3. Click on a character's grid for information about that character.
4. Click on the **Quit** button when you have finished previewing the font.

Step 1: Running xfd

When running the **xfd** client, you must specify the font you want to preview with the **-fn** option, followed by the name of the desired font. You can enter the complete font name or use font name wildcards or a font alias.

NOTE If the font name includes spaces, the name must be surrounded with asterisk characters (*) and enclosed in quotation marks. For example:

```
xfd -fn "*new century schoolbook*"
```

The **xfd** client opens a window and displays the entire character set of the named font in a grid of boxes, each character in its own box. The characters are shown in increasing order from left to right, top to bottom. The full name of the font is displayed across the top of the window, even if you specified a font alias.

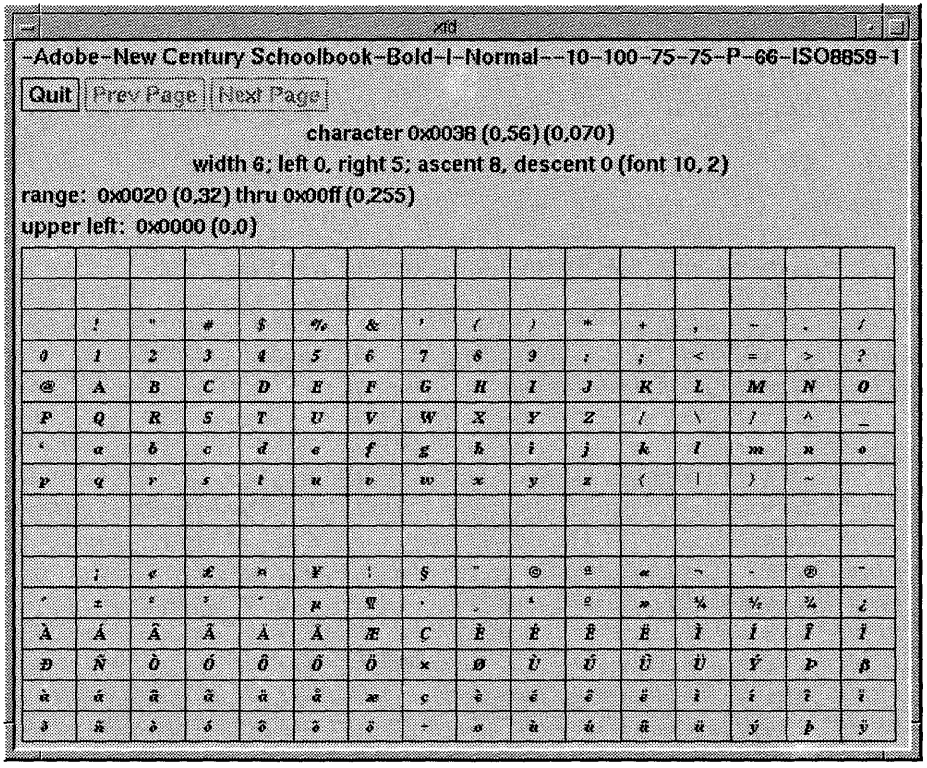

Figure 7-1 Example of xfd client

See also:

- **xfd**(XC) manual page

Step 2: Scrolling through multiple screens

Depending on the font you are previewing, it is possible that all of the font's characters may not fit in the **xfd** window at the same time. If the characters being displayed do not fit within the window, you can:

- Scroll forward through multiple screens by clicking a mouse button on the **Next Page** button. When you reach the last available screen, **Next Page** becomes inactive.
- Scroll back to the beginning screen by clicking a mouse button on the **Prev Page** button. When you return to the first screen, **Prev Page** becomes inactive.

Step 3: Displaying information on a character

You can display details about an individual character by clicking a mouse button on the grid containing the character. You are shown statistics about the character's width, left bearing, right bearing, ascent, and descent.

For more information on these statistics, see the **xfd**(XC) manual page.

Step 4: Quitting xfd

The **xfd** window remains on your screen until you click a mouse button on the **Quit** button, or type **q** or **Q**.

Specifying fonts

Fonts are X resources and, as such, can be changed like any other resource. You have a great deal of control over the fonts that are used on your system. For example, you can change the size of the icon label font, change the font for one client but not for others, and so forth. Before you actually make any font resource changes, however, you should be familiar with the information in Chapter 5, "Understanding resources" (page 79).

As with other resources, you can make font resource changes on a system-wide level or on a user-level. You can also specify font changes at the command line, affecting a client on a per-session basis.

All of these methods for specifying fonts are covered in the following sections.

Specifying fonts for the entire system

System-wide font resources are specified on a client basis, in the system-wide resource files. If you want to make font resource changes that affect an application every time it is run on your system, you must make your font changes in these files.

You must be logged into the system as *root* to perform this task.

To make a font resource change or addition in the resource database, perform the following steps. For more information on each of these steps, see the sections immediately following this procedure.

1. Open the desired client resource file for editing:
   - */usr/lib/X11/sco/startup/**client***
   - */usr/lib/X11/app-defaults/**client***
2. Make the desired font resource specification(s):
   - For an existing font resource specification, replace the old font designation with the desired font.
   - For a new font resource specification, add the new resource setting, using the following format:

     client*resource_name: fontname

   When you are finished, save your changes and exit the resource file.
3. If users are running clients that are affected by the new configuration at the time you make these changes, they must restart the clients to see the new fonts. In some cases, users may need to restart the X server itself to see the new fonts.

Step 1: Editing the client resource files

Default resources for clients are stored in files in two locations on the system: */usr/lib/X11/sco/startup* and */usr/lib/X11/app-defaults*. These directories contain several files, each named for the specific client they represent. The resource specifications defined in these files control the appearance and behavior of their specific client.

The files in */usr/lib/X11/sco/startup* contain server-specific resources. The values of these resources are loaded into the resource database and stored in the X server by the **xrdb** client when a Graphical Environment session is first started. These resource specifications are available for all clients that you run, regardless of the actual host that is running the applications.

The files in */usr/lib/X11/app-defaults* contain the majority of resource specifications for the clients on your system. The resources in these files are host-specific and only affect clients that are run on your machine. These resource files are read by the resource manager when the corresponding client is run.

If you want to configure a client to use a particular font, regardless of the machine on which the client is run, you should edit the client's resource file in */usr/lib/X11/sco/startup*. This approach has advantages and disadvantages: a client generally executes more quickly if its resources are already recognized by the X server, however, there is no guarantee that fonts that exist on your system are available to remote clients on their host machine. If you run remote clients, it is recommended that you set font resources in host-specific resource files.

If you want to configure a client to use a particular font only when it is run on the local system, edit the appropriate client file in */usr/lib/X11/app-defaults*.

If you intend to modify any of the files discussed here, it is recommended that you either make a backup copy of the file before you enter your resource changes, or comment out old resource values, using the "!" comment character, before entering new ones. This way you are assured of regaining the default values, if needed.

See also:

- "Methods for specifying resources" (page 87) for a more detailed explanation of resource files

Step 2: Setting the font resource

As with all resources, font resource specifications must use the correct format:

*client*resource_name: fontname*

client refers to the client you want to affect. You can supply either the client's binary or class name. ***resource_name*** is the actual resource variable you want to define. You can use either the resource's class or instance name. ***fontname*** is the actual name of the font you are selecting. You can use the full font name, font name wildcards, or a font alias when setting this value.

There are two resources from which you can choose when specifying fonts:

fontList (Class: **FontList**) — This resource specifies the font that is used to display text. Most clients accept this resource. This resource can also be used to specify lists of fonts, to accommodate the possibility that some systems may contain a set of fonts, while other systems contain a different set. If you list multiple fonts, they must be separated by white space.

font (Class: **Font**) — Generally, this resource specifies the font that is used for all Desktop icon labels. Other clients may also use this resource, however **fontList** is used more frequently.

Whether you are replacing the font of a currently defined resource or you are adding a new, perhaps more specific, resource, the resource line should use the same format. For example, if you want to change the default font for the **scocolor** client so it displays larger text, the resource line would read:

```
ScoColor*fontList: -adobe-helvetica-bold-r-normal--20-140-100-100-p-105-iso8859-1
```

See also:

- Chapter 5, "Understanding resources" (page 79) for more detailed information on specifying resources, including the use of class and instance names

Step 3: Activating the new fonts

Once you have made the desired font change to the client resource file or files, the new specifications are immediately available to all users. However, if users were running the affected clients while you set the new font values, they need to restart the client to see the new fonts.

If you made font specifications in any of the server-specific resource files, users will have to restart the X server before they see the new fonts.

Specifying fonts for individual users

Individual users can use their own unique set of fonts for the Desktop and other clients. These font settings do not change the default resources that are available to other users on the system.

To change fonts for an individual user, perform the following steps. For more information on each of these steps, see the sections immediately following this procedure.

1. Create a file called *.Xdefaults-**hostname*** in the user's home directory.
2. Edit the *.Xdefaults-**hostname*** file and add the new font resource specification, using the following format:

   client*resource_name**: ***fontname

   When you are finished, save your changes and exit the resource file.
3. If you created the *.Xdefaults-**hostname*** file as *root*, assign the appropriate user permissions to the file:

   chown ***username*** **.Xdefaults-*hostname***
   chgrp ***groupname*** **.Xdefaults-*hostname***
4. Start a Graphical Environment session.

Step 1: Creating an .Xdefaults-hostname file

Individual users can assign their own values to font resource specifications. You can either change the value of a font resource already set in the resource database, or you can set an entirely new font resource, perhaps a resource that changes only a particular aspect of a client's font usage. User-specified resources always override system defaults, allowing different users running the same clients to specify personal font preferences.

Individual resource settings are placed in a file called *.Xdefaults-**hostname***, where ***hostname*** is the name of the host, or machine, where the client is running.

You can add an *.Xdefaults-**hostname*** file to a user's home directory in one of two ways:

- Create a file named *.Xdefaults-**hostname***, then add the desired font specifications to the file. This approach is most useful if you are only making a few changes.
- Copy one or more of the client default files from */usr/lib/X11/app-defaults* to the user's home directory, merging the files into a single file called *.Xdefaults-**hostname***. You can then use the file as a template, deleting resources you do not want to change and specifying new font values for the font resources you do want to change.

NOTE Lines in the resource files in the *app-defaults* directory do not always specify the client, because each file applies to only one client. If you create an *.Xdefaults-**hostname*** file by copying the relevant lines from a file in the *app-defaults* directory, be sure to add the client's name to the beginning of each line if it is not already there. Otherwise, the font resource specification will affect all clients.

When the user invokes a client, the X server checks to see if an *.Xdefaults-**hostname*** file exists in **$HOME**. If such a file does exist, the resource values specified in the user resource file take precedence over any values assigned to the same resource in the resource database.

See also:

- "Methods for specifying resources" (page 87) for more information on the *.Xdefaults-**hostname*** file

Step 2: Setting the font resource

Font resource specifications must use the correct format:

client*resource_name: fontname

client refers to the client you want to affect. You can supply either the client's binary or class name. ***resource_name*** is the actual resource variable you want to define. You can use either the resource's class or instance name. ***fontname*** is the actual name of the font you are selecting. You can use the full font name, font name wildcards, or a font alias when setting this value.

There are two resources from which you can choose when specifying fonts:

fontList (Class: **FontList**) This resource specifies the font that is used to display text. Most clients accept this resource. This resource can also be used to specify lists of fonts, to accommodate the possibility that some systems may contain a set of fonts, while other systems contain a different set. If you list multiple fonts, they must be separated by white space.

font (Class: **Font**) Generally, this resource specifies the font that is used for all Desktop icon labels. Other clients may also use this resource, however **fontList** is used more frequently.

Whether you are replacing the font of a currently defined resource or you are adding a new, perhaps more specific, resource, the resource line should use the same format. For example, if you want to change the default font for the **scocolor** client so it displays larger text, the resource line would read:

```
ScoColor*fontList: -adobe-helvetica-bold-r-normal--20-140-100-100-p-105-iso8859-1
```

See also:

- Chapter 5, "Understanding resources" (page 79) for more detailed information on specifying resources, including the use of class and instance names

Step 3: Assigning correct ownership permissions

If you generated an *.Xdefaults-**hostname*** file for a user from the *root* account, whether by creating the file or by copying the file from files in */usr/lib/X11/app-defaults*, you must assign the file the correct ownership permissions. Run the **chown** command to assign the correct owner and the **chgrp** command to assign the correct group to the *.Xdefaults-**hostname*** file.

If you created your own *.Xdefaults-**hostname*** file, you can ignore this step. Your ownership permissions are already correct.

Step 4: Starting a Graphical Environment session

To see your new font settings, start a Graphical Environment session, either through **scologin** or by running **startx** from the command line.

Run the desired clients. When a client is started, it reads the **$HOME/***.Xdefaults-**hostname*** file for your personal resource specifications. The new font settings are noted and the client's text is displayed accordingly.

If you created your own *.Xdefaults-**hostname*** file during a session, and you want the new font values applied to clients that are currently running, you need to restart the clients.

If you want the Desktop client to reflect your new font specifications, you must end your current session and then start the X server again, either by logging in through a **scologin** window or by running the **startx** script from the command line.

Setting fonts from the command line

You can specify any resource setting on the command line that you would otherwise put into a resource file, including font resources.

To change a font resource from the command line, use the **-xrm** option when launching a client from a **scoterm** window:

client **-xrm** *'client*resource_name: fontname'* **&**

If you specify a font resource from the command line, the font is only set for the current client session. Subsequent sessions return to the default font specification for the client.

NOTE If you are using **scosession** and you end your current session with a client that was run from the command line still open on the display, the client is restored when you resume your session. Any command line options that were used to define the client are also restored. In this way, command line options can define client behavior on a more permanent basis.

See also:

- "Using the -xrm option" (this page)
- "Other command line font options" (page 150)

Using the -xrm option

The **-xrm** option allows you to set any resource value, including font specifications, from the command line. You must enter the resource specification as well as the desired font name when using this option. For example, to change the text font for **scoterm** to a smaller fixed-width font, you would enter the following at the command line:

scoterm -xrm 'ScoTerm*font: 5x8' &

When using the **-xrm** option, you should be aware of the following:

- A resource specification must be quoted using the single quotation mark (').
- You can specify more than one resource value from the command line at the same time. You must enter the **-xrm** option for each specification you make:

  scoterm -xrm 'ScoTerm*font: 5x8' -xrm 'ScoTerm*fontList: 5x8' &

Resource values specified at the command line only exist for the client session you are invoking. The resource values do not become part of the resource database and the values are not recreated the next time you run the client unless you use the **-xrm** option again.

Note that a font resource specified at the command line does not take effect if a font resource that takes precedence has already been loaded with **xrdb**. (See "Precedence rules for resource specifications" (page 87) for more information.) For example, say you loaded a resource file that includes the specification:

```
ScoTerm*font: 9x15
```

In this case, the following command line specification would not result in the **scoterm** client using a different font:

scoterm -xrm '*font: 5x8' &

Because the **ScoTerm*font** designation is more specific, the ***font** entry on the command line cannot override the font setting in the resource database.

To override the resource database, and get the 5x8 font, you would need to use a resource equally or more specific than the default setting. For example, the resource **ScoTerm*font** would provide the desired font change.

Other command line font options

You can also specify different fonts at the command line for some clients with an option other than **-xrm**. Most clients also accept the **-fn** option:

client* -fn *fontname

The **-fn** option can be useful in that you only need to know the name or alias of the font you want to specify; you do not need to be concerned with specifying font resources. However, if the resource database or your personal resource file sets this font value with a more specific resource, you do not see any changes when you use the **-fn** option.

Creating a font alias

You can assign aliases to fonts so you can specify them without having to type their complete font names. This is particularly useful if you tend to set a font often.

If you are creating aliases for fonts that exist within the */usr/lib/X11/fonts* subdirectories, you must be logged into the system as *root*. You should also ensure that */usr/bin/X11* has been added to *root*'s **$PATH** environment variable.

To create a font alias, perform the following steps. For more information on each of these steps, see the sections immediately following this procedure.

1. In the directory that contains the font that you want to alias, edit the file *fonts.alias* so that it contains at least one line with the following format:

   alias fontname

2. Reset the X server's font database with the following command:

   xset fp rehash

Once you have aliased a font and reset the X server, you can use the alias rather than the full font name whenever you refer to that font.

Step 1: Editing the fonts.alias file

In most cases, fonts are located in */usr/lib/X11/fonts* in one of the following subdirectories: *75dpi, 100dpi, misc, Speedo,* or *Type1*. The first three of these directories contain a *fonts.alias* file by default. You must be logged in as *root* to edit any of these files.

When you add an alias to a *fonts.alias* file, you must enter the ***alias***, the name by which you want to refer to the font, in the first column, and the ***fontname***, the font's full name, in the second column.

An easy way to include a font's full name in the *fonts.alias* file is to open two **scoterm** windows, one in which you are editing the *fonts.alias* file and one in which you open the *fonts.dir* file that contains the full name of the desired font. Use **scoterm**'s cut and paste functionality to copy the full font name from the *fonts.dir* file to the *fonts.alias* file. This approach reduces the possibility of making a typographical error in typing the font's full name.

If you make an error entering a font's full name in the *fonts.alias* file, the alias will not work when specified.

A typical *fonts.alias* file in */usr/lib/X11/fonts/75dpi* might contain these two aliases:

```
courier10  -adobe-courier-bold-o-normal--10-100-75-75-m-60-iso8859-1
courier12  -adobe-courier-bold-o-normal--12-120-75-75-m-60-iso8859-1
```

NOTE The directory that contains the font you want to alias must be included in the font search path. By default, the font search path includes the *misc*, *75dpi*, and *100dpi* directories in */usr/lib/X11/fonts*.

You can specify any number of aliases in *fonts.alias* files.

Step 2: Resetting the font database

If you created a font alias during a Graphical Environment session, you must tell the X server to reread the font database so it recognizes your new alias. To reset the font database, enter the following command from a **scoterm** window:

xset fp rehash

You must reset the font database whenever you edit a *fonts.alias* file while running a Graphical Environment session.

If you created a font alias from the operating system command line (and your SCO system was not running), there is no need to update the X server. The changes that you made to *fonts.alias* are incorporated automatically the next time you start your system.

Adding a font to your system

It is possible to add additional fonts to your system. If you want to add fonts to any of the default subdirectories in */usr/lib/X11/fonts*, you must be logged in as *root*. You should also ensure that */usr/bin/X11* has been added to *root*'s **$PATH** environment variable.

To add one or more fonts to your X server, perform the following steps. You must be logged onto the system as *root*. For more information on each of these steps, see the sections immediately following this procedure.

1. Place the new font file or files in the directory in which you want to store them.
2. Run the **bdftopcf** utility, if desired, to convert BDF font files to PCF files:

   bdftopcf *font*.bdf > *font*.pcf

3. Run the **mkfontdir** command and indicate the font's directory location, if necessary:

   mkfontdir *font_location*

4. If you added fonts to a directory not in the current font path, add the new directory to the font search path:

   xset fp+ *font_location*

5. Reset the server's font database with the following command:

   xset fp rehash

Step 1: Placing the font files on your system

When you add new fonts to your system, you need to give some thought to where you want to put the actual font files. Generally, fonts are located in subdirectories of */usr/lib/X11/fonts*. You can put your new fonts in one of these subdirectories, perhaps in */usr/lib/X11/fonts/misc*, or anywhere else on the system, including subdirectories within individual users' accounts.

New font files can be in Bitmap Display Format (BDF), Server Natural Format (SNF), or Portable Compiled Format (PCF). See Step 2 (this page) for more information on these formats.

Step 2: Converting from BDF to PCF format

The standard font format is Bitmap Display Format (BDF). BDF font files are portable and represent the characters of a font in ASCII. These files generally have a *.bdf* extension. Fonts can also be compiled in the Portable Compiled Format (PCF). PCF font files generally have a *.pcf* extension.

The X server can use font files that are in BDF, PCF, or SNF format. However, there are several good reasons to convert any BDF files on your system into PCF files:

- BDF files are much larger in size than the compiled PCF files. By converting BDF files to the Portable Compiled Format and then removing the BDF versions of the fonts, you can save a great deal of disk space.
- On an appropriate architecture machine, PCF font files are loaded more quickly than their larger BDF counterparts.
- In most cases, to use your fonts on an X terminal, the fonts must be compiled as either PCF (or SNF) files.

To convert BDF font files to PCF files, run the following command:

bdftopcf *font*.bdf > *font*.pcf

font.bdf refers to the actual name of the BDF font file that you want to convert. The file ***font.pcf*** contains the converted font.

If you run the **bdftopcf** utility from a directory other than the one containing the BDF file you want converted, remember to specify the complete pathname of the file.

NOTE If you are converting BDF fonts for use with an X terminal, be sure to read the manual that is supplied with your terminal for special instructions on running the **bdftopcf** utility. Some X terminals require that you specify various flags for the **bdftopcf** command; other X terminals only support some of **bdftopcf**'s options.

For more information on the flags that you can use with the **bdftopcf** command, see the **bdftopcf**(X) manual page.

Step 3: Running mkfontdir

If you add new fonts to an existing fonts directory or create an entirely new fonts directory, you must run the **mkfontdir** command. This command checks for the existence of a *fonts.dir* file in the directory that contains the new fonts. If such a file exists, **mkfontdir** adds an entry for each new font to the file. If the font directory is new and does not already have a *fonts.dir* file, **mkfontdir** creates the file for you.

If you run **mkfontdir** from the directory containing the new fonts, you do not need to specify the directory name. By default, the **mkfontdir** command operates on the current working directory. If, however, you run **mkfontdir** from a different location on your system, you must specify the complete path of the directory containing the new files. For example, if you added new fonts to the *misc* directory, you should enter:

mkfontdir /usr/lib/X11/fonts/misc

NOTE To run **mkfontdir** on one of the default font directories (*/usr/lib/X11/fonts/misc, /usr/lib/X11/fonts/75dpi, /usr/lib/X11/fonts/100dpi*) */usr/lib/X11/fonts/Speedo*, or */usr/lib/X11/fonts/Type1*), you must be logged in as *root*.

If you added new fonts to a subdirectory named *myfonts* within your home directory, you should enter:

mkfontdir $HOME/myfonts

See also:

- **mkfontdir**(X) manual page

Step 4: Updating the font search path

If you placed your new font files in a directory that is not part of the font search path, you need to add the font directory to the search path so the X server can locate the new files. You can tell the server of the new directory using **xset** and the **fp+** option:

xset fp+ ***font_location***

For example, if you added fonts to a subdirectory called *myfonts* in your home directory, the following command appends the directory **$HOME**/*myfonts* to the X server's font search path:

xset fp+ $HOME/myfonts

You can add multiple directories to the font path at the same time. After the **fp** option, each font directory should be separated by commas, as below:

xset fp+ $HOME/myfonts,/usr/lib/X11/fonts/newfonts

Step 5: Resetting the font database

To complete the font installation, you must make the X server aware of the additions to the font database. The font files must be located in a directory included in the font search path. To reset the font database, enter the following:

xset fp rehash

This command tells the X server to reread all of the *fonts.dir* and *fonts.alias* files in the current font path. You should now be able to use your new fonts in a resource file or on the command line.

Example of setting fonts

This section provides a comprehensive example that ties together many of the concepts and procedures discussed in this chapter.

Let's assume you are an administrator for a system whose X server and X clients are accessed by several users. Many of these users have requested a bigger default text font for the **scomail** client. This example explains the following:

- choosing a particular font
- creating an alias for the font so you do not have to type the full font name repeatedly
- incorporating the new font designation into the font database so the new font is available to all users on your system

The following steps result in a bigger text font for the **scoedit** client.

1. Log into the system as *root*. If logging into the *root* account does not automatically start the X server, do so now by entering the following command at the prompt:

   startx &

2. Before you can decide on a font to use, you need to know what kinds of fonts exist on your system. The **xlsfonts** client displays a list of all available fonts. At the prompt in a **scoterm** window, enter:

   xlsfonts | more

   You see an extensive list of fonts. For this example, let's assume the following font looks like it might be a possible choice:

   ```
   -adobe-courier-medium-r-normal--18-180-75-75-m-110-iso8859-1
   ```

3. Run the **xfd** command to preview the characteristics of your font:

   xfd -fn -adobe-courier-medium-r-normal--18-180-75-75-m-110-iso8859-1

   You see a window that displays all of the characters of the font you specified. After some examination, you decide that you want to use this 18-point Courier font.

4. Create an alias for your chosen font so that you do not have to type the full font name when making your font specifications. Because the font is a 75dpi font, change to the */usr/lib/X11/fonts/75dpi* directory and edit *fonts.alias*, adding the following line to the end of the file:

   ```
   courier18    -*-courier-medium-r-*--18-*-*-*-m-*-*-*
   ```

Notice that instead of specifying the full font name in the aliases file, you can shorten the amount of typing you need to do by using wildcard characters to replace several parts of the font name.

5. Save and exit the *fonts.alias* file.
6. Reset the X server's font database so your new alias is active:

   xset fp rehash

   You can now use the courier18 alias to specify your font change.
7. You are ready to make the new font specification for the **scomail** client. Change directories to */usr/lib/X11/sco/startup* and look for the file named *ScoMail*. This is the default resource file that you need to edit to make your font change.
8. Open the *ScoMail* file for editing and search for the following line:

   ```
   ScoMail*XmText*fontList:
   ```

   This is the resource that controls the font used to display text in the **scomail** window.
9. Comment out the line containing the font value for this resource so that it looks like this:

   ```
   !ScoMail*XmText*fontList: -*-courier-medium-r-*--10-*-m-*-iso8859-1
   ```

   By commenting out the default resource setting, instead of simply deleting the line, you leave yourself a safeguard. You can always return to this default if you make a mistake when setting a new font value.
10. Now open a line immediately below the resource you just commented out and enter your new resource designation, using the courier18 alias:

    ```
    ScoMail*XmText*fontList:  courier18
    ```

11. Save and exit the *ScoMail* file.
12. Now you should verify that the new font specification works. First, you need to load the new font value into the resource database, to override the original value for this font resource. Run the following command from a **scoterm** window:

    xrdb -merge ScoMail

    Because you are currently located in the */usr/lib/X11/sco/startup* directory, you only need to specify the name of the resource file you want to load into the resource database. If you run the **xrdb** command from a directory other than the one containing the modified resource file, you would need to specify the complete pathname of the file.

    When your **scoterm** prompt returns, run the **scomail** client. The font that displays in the editing window should now be large enough to satisfy your users.

Chapter 8

Configuring window size and location

The size and location of a window is called its "geometry". Geometry specifies four components of a window: height, width, horizontal location, and vertical location.

This chapter describes:

- background on window geometry (this page)
- configuring the initial size and location of windows (page 160)
- resizing and repositioning the main Desktop (page 169) if you do not want it to occupy the entire Root window

There is also an example (page 170) at the end of this chapter that helps tie together many of the concepts and procedures discussed in this chapter.

About window geometry

The window manager controls the layout of windows on your screen. It allows windows to overlap, and, when clients are first started, it positions client windows in what it thinks are the best possible locations.

The session manager, **scosession**, stores the layout of the windows on your screen in **$HOME**/*.odtpref/ScoSession/dynamic*. Each time you start the Graphical Environment, you can specify that **scosession** either resumes with the window specifications saved from your last session (the configurations saved in the **$HOME**/*.odtpref/ScoSession/dynamic* file), or you can specify that **scosession** start the Graphical Environment with the default configurations.

The default Graphical Environment configurations are those defined either by you or by the system. You can change geometry configurations with the mouse, or you can specify geometry configurations with the **geometry**

resource variable or with the **-geometry** command line option. Both the **geometry** resource variable and the **-geometry** command line option take the same arguments: the window's ***width***, ***height***, ***xoff*** (x coordinate offset), and ***yoff*** (y coordinate offset), where all four arguments are numbers specifying size or location.

See also:

- "Desktop geometry" (this page)
- Chapter 3, "Customizing startup of the Graphical Environment" (page 43)
- **scosession**(XC) manual page

Desktop geometry

By default, the Desktop client occupies the entire Root window. In this mode, the Desktop cannot be resized or repositioned with the mouse.

If you want to change the Desktop so that it does not occupy the entire screen, you should specify this change using the Desktop **Preferences Editor**. The Desktop is placed in a window, and it is then possible to further manipulate its size and location with the mouse.

See also:

- "Resizing the Desktop" (page 169)

Configuring window geometry

geometry is an X resource variable and, as such, can be changed like any other resource variable. Although the window manager manages client window geometry, you have a great deal of control over a client's initial window geometry. For example, you can change the location of only one client window, change the location of several client windows, change the size of the entire Desktop, and so forth.

See also:

- "Specifying geometry for the entire system" (page 161)
- "Specifying geometry for individual users" (page 164)
- "Specifying geometry from the command line" (page 168)
- Chapter 5, "Understanding resources" (page 79)
- Appendix A, "OSF/Motif window manager resources" (page 377)

Specifying geometry for the entire system

To change the geometry of a client window for all users, perform the following steps. You must be logged in as *root* to do this task. For more information on each of the steps in this task, see the sections immediately following this list.

1. Open the desired client resource file for editing:
   - */usr/lib/X11/sco/startup/**client***
   - */usr/lib/X11/app-defaults/**client***
2. Make the desired geometry resource specification(s):
   - For an existing geometry resource specification, replace the old designation with the desired geometry.
   - For a new geometry resource specification, add the new resource setting, using the following syntax:

     client**geometry**: [*widthxheight*][±*xoff*±*yoff*]

   When you are finished, save your changes and exit the resource file.
3. If users are running clients that are affected by the new configuration at the time you make these changes, they must restart the clients to see the new geometry designations.

Step 1: Editing the client resource file

Default resources for clients are stored in files in two locations on the system: */usr/lib/X11/sco/startup* and */usr/lib/X11/app-defaults*. These directories contain several files, each named for the specific client they represent. The resource specifications defined in these files control the appearance and behavior of their specific client.

The files in */usr/lib/X11/sco/startup* contain server-specific resources. The values of these resources are loaded into the resource database and stored in the X server by the **xrdb** client when a Graphical Environment session is first started. These resource specifications are available for all clients that you run, regardless of the actual host that is running the applications.

The files in */usr/lib/X11/app-defaults* contain the majority of resource specifications for the clients on your system. The resources in these files are host-specific and only affect clients that are run on your machine. These resource files are read by the resource manager when the corresponding client is run.

If you want to configure a client to use a particular geometry, regardless of the machine on which the client is run, you should edit the client's resource file in */usr/lib/X11/sco/startup*.

If you want to configure a client to use a particular geometry only when it is run on the local system, edit the appropriate client file in */usr/lib/X11/app-defaults*. If you intend to modify any of the files discussed here, it is recommended that you either make a backup copy of the file before you enter your resource changes, or comment out old resource values, using the "!" comment character, before entering new ones. This way you are assured of regaining the default values, if needed.

See also:

- "Methods for specifying resources" (page 87) for a more detailed explanation of resource files

Step 2: Setting the geometry resource

As with all resources, geometry resource specifications must use the correct format. The syntax of the **geometry** resource variable is:

*client****geometry:** [*width*x*height*][±*xoff*±*yoff*]

client refers to the client you want to affect. You can supply either the client's binary or class name. **geometry** is the resource variable you are setting. The ***width*** and ***height*** arguments represent the size of the client window; most client windows are measured in pixels; however, terminal emulators such as **scoterm** and **xterm** are measured in characters. The ***xoff*** and ***yoff*** arguments represent the x and y coordinates of the client window, respectively.

Both ***xoff*** and ***yoff*** are measured in pixels from the edge of the screen. They must always be specified in pairs, and ***xoff*** must always be specified first. Both specifications can be either negative or positive numbers.

The ***xoff*** argument represents the distance of a window from the left or right edge of the screen. A window with a +0 (plus zero) x-offset is located on the left side of your screen. A window with a -0 (negative zero) x-offset is located on the right side of your screen.

The ***yoff*** argument represents the distance of a window from the top or bottom edge of the screen. A window with a +0 (plus zero) y-offset is located at the top of your screen. A window with a -0 (negative zero) y-offset is located at the bottom of your screen.

For example, to position a window in one of the four corners of the screen, use one of the following ***xoff*** and ***yoff*** combinations:

- `+0+0` specifies the top, left edge of the screen
- `+0-0` specifies the bottom, left edge of the screen
- `-0+0` specifies the top, right edge of the screen
- `-0-0` specifies the bottom, right edge of the screen

Figure 8-1, " Display coordinates" shows the ***x*** and ***y*** corner coordinates of any given display.

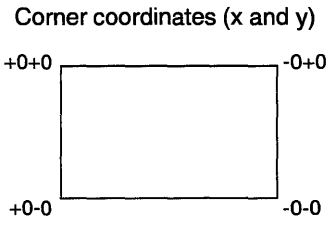

Figure 8-1 Display coordinates

Because ***xoff*** and ***yoff*** are measured in pixels, which are small units, it is difficult to specify a location and know exactly where that location is on your screen. Because corner locations are always +0 or -0, it's much easier to specify an exact corner location than it is to specify a location somewhere in the middle of your screen.

The following is an example **geometry** specification:

```
ScoTerm*geometry: 80x25+0+0
```

The size of this **scoterm** window is defined to be 80 characters wide by 25 characters high. It is located at the top, left edge of the screen.

You can specify both the size and location of a client window, or you can specify one or the other. For example, to change the size of a **scoterm** window only, specify:

ScoTerm*geometry: 70x20

The **scoterm** window size is defined to be 70 characters wide by 20 characters high. Either the **scoterm** client provides a default ***x*** and ***y*** coordinate position, or if the **scoterm** client does not have a default position defined, the window manager automatically positions the **scoterm** window.

Likewise, you can specify only the location of a client and rely on the client or the window manager to provide default size specifications. For example, to define the **scoterm** window so it appears in the bottom, right edge of your screen, specify:

ScoTerm*geometry: -0-0

See also:

- "Example of specifying window geometry" (page 170)
- Chapter 5, "Understanding resources" (page 79)

Step 3: Activating the new geometry settings

Once you have made the desired geometry changes to the client resources files, the new specifications are immediately available to all users. However, if users were running the affected clients while you set the new geometry values, they need to restart the clients before the windows reflect the new geometry settings.

Specifying geometry for individual users

Individual users can specify unique geometry settings for their client windows. These settings do not change the default geometry resources for other users on the system.

To change the geometry for an individual user, perform the following steps. For more information on each of these steps, see the sections immediately following this procedure.

1. Create a file called *.Xdefaults-**hostname*** in the user's home directory.
2. Edit the *.Xdefaults-**hostname*** file and add the new geometry resource specifications, using the following format:

   client*geometry:** [**widthxheight**][±**xoff**±**yoff**]*

   When you are finished, save your changes and exit the resource file.
3. If you created the *.Xdefaults-**hostname*** file as *root*, assign the appropriate user permissions to the file:

   chown ***username*** **.Xdefaults-*hostname***
   chgrp ***username*** **.Xdefaults-*hostname***
4. Start a Graphical Environment session.

Step 1: Creating an .Xdefaults-hostname file

Individual users can assign their own values to geometry resource specifications. You can either change the value of a geometry resource already set in the resource file, or you can set an entirely new geometry resource. These user defaults always override system defaults, allowing different users running the same clients to specify personal geometry preferences.

Individual resource settings are placed in a file called *.Xdefaults-**hostname***, where ***host*** is the name of the host, or machine, where the client is running.

You can add an *.Xdefaults-**hostname*** file to a user's home directory in one of two ways:

- Create a file named *.Xdefaults-**hostname***, then add the desired geometry specifications to the file. This approach is most useful if you are only making a few changes.
- Copy one or more of the client default files from */usr/lib/X11/app-defaults* to the user's home directory, merging the files into a single file called *.Xdefaults-**hostname***. You can then use the file as a template, deleting resources you do not want to change and specifying new geometry values for the geometry resources you do want to change.

NOTE Lines in the resource files in the *app-defaults* directory do not always specify the client, because each file applies to only one client. If you create an *.Xdefaults-**hostname*** file by copying the relevant lines from a file in the *app-defaults* directory, be sure to add the client's class name or binary name to the beginning of each line if it is not already there. Refer to the client's man page to determine its class name.

When the user invokes a client, the client checks to see if an *.Xdefaults-**hostname*** file exists in **$HOME**. If such a file does exist, the resource values specified in the user resource file take precedence over any values assigned to the same resource in the resource database.

See also:

- "Methods for specifying resources" (page 87) for more information on the *.Xdefaults-**hostname*** file

Step 2: Setting the geometry resource

As with all resources, geometry resource specifications must use the correct format. The syntax of the **geometry** resource variable is:

***client**geometry:** [*widthxheight*][±*xoff*±*yoff*]

client refers to the client you want to affect. You can supply either the client's binary or class name. **geometry** is the resource variable you are setting. The ***width*** and ***height*** arguments represent the size of the client window (most client windows are measured in pixels; however, terminal emulators such as **scoterm** and **xterm** are measured in characters). ***xoff*** and ***yoff*** refer to the ***x*** and ***y*** coordinates of the client window.

Both ***xoff*** and ***yoff*** are measured in pixels from the edge of the screen. They must always be specified in pairs, and ***xoff*** must always be specified first. Both specifications can be either negative or positive numbers.

The ***xoff*** argument represents the distance of a window from the left or right edge of the screen. A window with a +0 (plus zero) x-offset is located on the left side of your screen. A window with a -0 (negative zero) x-offset is located on the right side of your screen.

The ***yoff*** argument represents the distance of a window from the top or bottom edge of the screen. A window with a +0 (plus zero) y-offset is located at the top of your screen. A window with a -0 (negative zero) y-offset is located at the bottom of your screen.

For example, to position a window in one of the four corners of the screen, use one of the following ***xoff*** and ***yoff*** combinations:

- `+0+0` specifies the top, left edge of the screen
- `+0-0` specifies the bottom, left edge of the screen
- `-0+0` specifies the top, right edge of the screen
- `-0-0` specifies the bottom, right edge of the screen

Because ***xoff*** and ***yoff*** are measured in pixels, which are small units, it is difficult to specify a location and know exactly where that location is on your screen. Because corner locations are always +0 or -0, it is much easier to specify an exact corner location than it is to specify a location somewhere in the middle of your screen.

The following is an example **geometry** specification:

```
ScoTerm*geometry: 80x25+0+0
```

The size of this **scoterm** window is defined to be 80 characters wide by 25 characters high. It is located at the top, left corner of the screen.

You can specify both the size and location of a client window, or you can specify one or the other. For example, to change the size of a **scoterm** window only, specify:

ScoTerm*geometry: 70x20

The **scoterm** window size is defined to be 70 characters wide by 20 characters high. Either the **scoterm** client provides a default *x* and *y* coordinate position, or if the **scoterm** client does not have a default position defined, the window manager automatically positions the **scoterm** window.

Likewise, you can specify only the location of a client and rely on the client to provide default size specifications. For example, to define the **scoterm** window so it appears in the bottom, right edge of your screen, specify:

ScoTerm*geometry: -0-0

See also:

- "Example of specifying window geometry" (page 170)
- Chapter 5, "Understanding resources" (page 79)

Step 3: Assigning correct ownership permissions

If you generated an *.Xdefaults-**hostname*** file for a user from the *root* account, whether by creating the file or by copying the file from files in */usr/lib/X11/app-defaults*, you must assign the file the correct ownership permissions. Run the **chown** command to assign the correct owner and the **chgrp** command to assign the correct group to the *.Xdefaults-**hostname*** file:

chown *username* .Xdefaults-*hostname*
chgrp *username* .Xdefaults-*hostname*

If you created your own *.Xdefaults-**hostname*** file, you can ignore this step. Your ownership permissions are already correct.

Step 4: Starting a Graphical Environment session

To see your new geometry settings, start a Graphical Environment session, either through **scologin** or by running **startx** from the command line.

Run the desired clients. When a client is started, it reads the **$HOME**/*.Xdefaults-**hostname*** file for your personal resource specifications. The new geometry settings are noted and the client's windows are configured accordingly.

If you created your own *.Xdefaults-**hostname*** file during a session, and you want the new geometry values applied to clients that are currently running, you need to restart the clients.

Specifying geometry from the command line

You can specify geometry settings for client windows from the command line. If you specify a geometry specification from the command line, the setting only exists for the current invocation of the client. Subsequent sessions return to the default geometry specification for the client.

NOTE If you are using **scosession** and you end your current session with a client that was run from the command line still open on the display, the client is restored when you resume your session. Any command line options that were used to define the client are also restored. In this way, command line options can define client behavior on a more permanent basis.

To configure a different window geometry from the command line, use the following command line option when launching a client from the **scoterm** window:

client **-geometry** [*width*x*height*][±*xoff*±*yoff*]

client refers to the binary name or the class name of the client you want to affect. *width* and *height* are the arguments to **-geometry**, representing the size of the client window. The *xoff* and *yoff* arguments represent the *x* and *y* coordinates of the client window, respectively.

The **-geometry** option can be abbreviated to any substring of the word "geometry", as long as the substring does not match any of the client's command line options. For example, if the client you are resizing or repositioning does not accept other options that begin with "g" or "geo", you can specify **-geometry** with **-g** or **-geo** on the command line.

Because the **-geometry** command line option and the **geometry** resource variable take the same arguments, see Step 2 (page 166) in the previous section, "Specifying geometry for individual users," for a more detailed explanation of *width, height, xoff, yoff*.

You can also use the **-xrm** command line option to set the geometry resource variable for clients, on a per-session basis. See "Resource specifications on the command line" (page 97) for more information.

Resizing the Desktop

To change the size of the Desktop so it does not occupy the entire Root window, perform the following steps. For more information on these steps, see the sections immediately following this procedure.

1. Reconfigure the Desktop so it no longer occupies the Root window, using the Desktop **Preferences Editor**, located in the *Controls* window.
2. If desired, resize and reposition the new desktop window with the mouse.

Step 1: Using the Desktop Preferences Editor

If you want to change the Desktop so that it does not occupy the entire screen, you should use the Desktop **Preferences Editor**. When you make this change, the Desktop is placed in a window, and it is then possible to further manipulate its size and location with the mouse.

To reconfigure the Desktop's geometry with the **Preferences Editor**:

1. Double-click on the **Preferences Editor** icon in the *Controls* window.
2. Double-click on the **Main Desktop Behavior** icon.
3. Click on the arrow next to the "Desktop as root window" field and select "No" in the popup box.
4. Click on **Apply**. You are prompted to restart the Desktop. The Desktop restarts when you select **Yes**.

   When the Desktop restarts, the Desktop is placed in a window, similar in size to other client windows.

See also:

- "Using the Preferences Editor" (page 24)
- "Main Desktop behavior options" (page 38)

Step 2: Resizing the Desktop with the mouse

If you want to further change the Desktop's size and location, you can now resize and reposition it by dragging it with the mouse.

Example of specifying window geometry

This section provides a comprehensive example that ties together many of the concepts and procedures discussed in this chapter.

Let's assume you are an administrator for a system whose X server and clients are accessed by several users. Many of these users have requested that the **scoterm** client be run in a smaller window.

The following steps result in a smaller system-wide **scoterm** window.

NOTE Some full screen programs, **vi**(C) for example, automatically assume that they are running on a window that is 80 characters wide by 25 characters high. If you change the width and height dimensions on a terminal emulator, you might also need to set the **$LINES** and **$COLUMNS** environment variables to match the new terminal emulator geometry.

1. Log into the system as *root*.
2. Change directories to */usr/lib/X11/app-defaults* and locate the file named *ScoTerm*. This is the default resource file that you need to edit to make your geometry changes.
3. Open the *ScoTerm* file for editing and search for `ScoTerm*geometry`. (If this entry does not exist, enter the geometry specified in step 4.) For the purposes of this example, let's assume the line that contains this resource looks like this:

   ```
   ScoTerm*geometry: 80x25+0+0
   ```

   Comment out this line. By commenting out the default resource setting, instead of simply deleting the line, you leave yourself a safeguard. You can always return to this default if you make a mistake when setting a new geometry value.
4. Now open a line immediately below the resource you just commented out (or if a resource was not previously set, open a line anywhere in the file). After seeing that the default size of the **scoterm** window was 80 characters wide by 25 characters high, you decide that 60 by 20 would be a size that would satisfy your users. Enter your new geometry specification:

   ScoTerm*geometry: 60x20

   Notice that you do not have to specify the location of the **scoterm** window. If you want to make sure the old location remains the same, specify the old x and y coordinates on the line you add, directly after 60x20. If *ScoTerm* does not specify a default location, the window manager positions it.

5. Save and exit the *ScoTerm* file.
6. To test the new **scoterm** geometry, start a Graphical Environment session, by entering the following command at the prompt:

   startx &

7. Run the **scoterm** client. The window should now be 60 characters wide by 20 characters high.

Chapter 9

Changing cursor appearance

This chapter explains how to change the cursors in the Graphical Environment, on both a system-wide and individual basis. Specifically, you can change the appearance of:

- the Desktop cursor (page 178)
- the **scoterm** cursor (page 186)
- the Root window cursor (page 178)

There is also a section of examples (page 192) at the end of this chapter that help tie together many of the concepts and procedures discussed in this chapter.

See also:

- "About cursor appearance" (page 174)
- Chapter 5, "Understanding resources" (page 79)
- Chapter 7, "Changing fonts" (page 125)
- the documentation supplied with your favorite bitmap editor

About cursor appearance

You can configure the cursors that are used in three different parts of the Graphical Environment: the Desktop, **scoterm** windows, and the Root window. The Desktop client cursors are composed of bitmap files; **scoterm** uses fonts to control pointer cursor appearance; and the Root window cursor accepts both bitmap files and fonts.

See also:

- "Desktop cursor appearance" (this page)
- "scoterm cursor fonts" (page 176)
- "Root window cursor appearance" (page 178)

Desktop cursor appearance

The Desktop cursor is a bitmap file. Each Desktop cursor has both a data bitmap and a mask bitmap component associated with it. These two components together form a cursor shape. The data bitmap defines the image associated with the cursor. The mask bitmap acts as an outline, defining which portion of the cursor is transparent.

The Desktop has several default cursors already defined, but any of them can be redefined. Both users and system administrators can modify the default Desktop cursors. In addition, they can create new bitmaps to create brand new cursors.

Users make Desktop cursor specifications in their **$HOME**/*XDesktop3* file. System administrators make Desktop cursor specifications in the */usr/lib/X11/app-defaults/XDesktop3* file. Both a data pixmap and a mask pixmap must be specified; if only one pixmap is specified, the cursor designation is ignored.

The default Desktop cursors are located in the default picture directories, */usr/lib/X11/IXI/XDesktop/bitmaps/xdt_c_[large | small]*. You can examine the files located in these directories to see the contents of sample bitmap files and sample filenames, but do not edit the files in this directory. If you want to edit a default Desktop cursor, copy the default bitmap file, then make desired changes to the copy.

The default Desktop cursors are defined as follows:

alert	when an alert box is displayed (except within the box)
bgTrigger	when a user clicks on the background
busy	when the Desktop is processing
drag	when a single icon is being dragged
fatal	when an error box is displayed (except within the box)
iconTrigger	when a user clicks on an icon to select it
idle	when the Desktop is waiting for the user to do something
multiDrag	when several icons are being dragged
rubber	when a user is using "rubberbanding" to select one or more icons

Note that bgTrigger, iconTrigger, and rubber all use the same pixmap hand.

The following table lists the default Desktop cursors:

Table 9-1 Resources that control Desktop cursor appearance

Data pixmap	Class
alert.data	XDesktop3.Cursor.Bitmap
bgTrigger.data	XDesktop3.Cursor.Bitmap
busy.data	XDesktop3.Cursor.Bitmap
drag.data	XDesktop3.Cursor.Bitmap
fatal.data	XDesktop3.Cursor.Bitmap
iconTrigger.data	XDesktop3.Cursor.Bitmap
idle.data	XDesktop3.Cursor.Bitmap
multiDrag.data	XDesktop3.Cursor.Bitmap
rubber.data	XDesktop3.Cursor.Bitmap
Mask pixmap	**Class**
alert.mask	XDesktop3.Cursor.Bitmap
bgTrigger.mask	XDesktop3.Cursor.Bitmap
busy.mask	XDesktop3.Cursor.Bitmap
drag.mask	XDesktop3.Cursor.Bitmap
fatal.mask	XDesktop3.Cursor.Bitmap
iconTrigger.mask	XDesktop3.Cursor.Bitmap
idle.mask	XDesktop3.Cursor.Bitmap
multiDrag.mask	XDesktop3.Cursor.Bitmap
rubber.mask	XDesktop3.Cursor.Bitmap

See also:

- "Changing the Desktop cursor" (page 178)

scoterm cursor fonts

The pointer cursor in **scoterm** windows (and **xterm** windows) is a type of font. Both users and system administrators can change the pointer cursor appearance with the **pointerShape** resource.

Users specify the **pointerShape** resource in their *.Xdefaults-**hostname*** file, where ***hostname*** is the name of the host, or machine, where the client is running. System administrators specify the **pointerShape** resource in the */usr/lib/X11/app-defaults/ScoTerm* file.

The following table contains a list of the standard cursor font names:

Table 9-2 Standard cursor font names

X_cursor	dotbox	man	sizing
arrow	double_arrow	middlebutton	spider
based_arrow_down	draft_large	mouse	spraycan
based_arrow_up	draft_small	pencil	star
boat	draped_box	pirate	target
bogosity	exchange	plus	tcross
bottom_left_corner	fleur	question_arrow	top_left_arrow
bottom_right_corner	gobbler	right_ptr	top_left_corner
bottom_side	gumby	right_side	top_right_corner
bottom_tee	hand1	right_tee	top_side
box_spiral	hand2	rightbutton	top_tee
center_ptr	heart	rtl_logo	top_trek
circle	icon	sailboat	u1_angle
clock	iron_cross	sb_down_arrow	umbrella
coffee_mug	left_ptr	sb_h_double_arrow	ur_angle
cross	left_side	sb_left_arrow	ur_angle
cross_reverse	left_tee	sb_right_arrow	ur_angle
crosshair	leftbutton	sb_up_arrow	ur_angle
diamond_cross	ll_angle	sb_v_double_arrow	watch
dot	lr_angle	shuttle	xterm

Figure 9-1, "Cursor font character set" displays all of these standard cursor fonts.

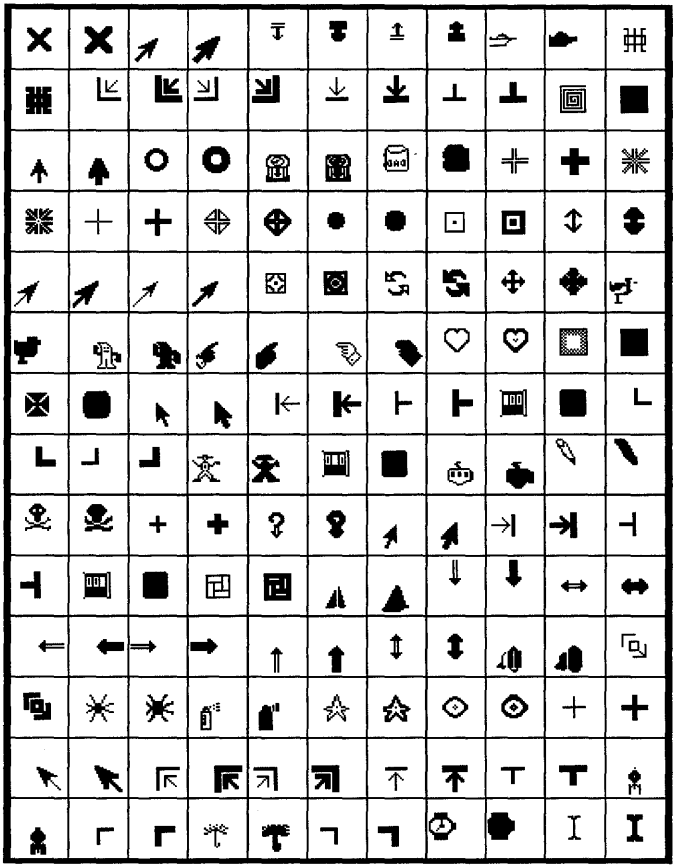

Figure 9-1 Cursor font character set

See also:

- "Changing the scoterm cursor" (page 186)

Root window cursor appearance

When the Desktop is not covering the entire screen, you see the Root window. If you want to change the pointer cursor appearance in the Root window, use the **xsetroot**(XC) command.

xsetroot takes two cursor appearance options:

-cursor ***cursorfile maskfile*** lets you change the pointer cursor (when it is outside the Desktop) to whatever you want. Cursor files and mask files are bitmaps, and can be made with a bitmap program, such as **scopaint**(XC).

-cursor_name ***cursorname*** lets you change the pointer cursor to one of the standard cursors from the cursor font

See also:

- **xsetroot**(XC) manual page for more information on changing the Root window cursor
- "Changing the Desktop cursor" (this page) and "Changing the scoterm cursor" (page 186) for more information about cursors made of bitmap files or fonts

Changing the Desktop cursor

The Desktop cursors are bitmap files. Describing how to create bitmaps is beyond the scope of this chapter. See the **scopaint**(XC) manual page for more information about bitmaps. The following procedures assume that you understand bitmap files, and that if you want to create a new Desktop cursor, you have already created the bitmap picture.

Cursor resources can be changed like any other resource, and as with other resources, you can make cursor resource changes on a system-wide level or on a user level.

See also:

- "Specifying Desktop cursors for the entire system" (page 179)
- "Specifying Desktop cursors for individual users" (page 182)
- Chapter 5, "Understanding resources" (page 79)

Specifying Desktop cursors for the entire system

To make a Desktop cursor change, perform the following steps. You must be logged into the system as *root* to perform this task. For more information on each of the steps in this task, see the sections immediately following this list.

1. Create a customized picture directory and move the new bitmap file into this directory.
2. Open the system Desktop resource file, */usr/lib/X11/app-defaults/XDesktop3*, for editing.
3. Make the desired cursor resource specification(s):
   - For an existing cursor resource specification, replace the old cursor name and/or location with the desired name and/or location.
   - For a new cursor resource specification, add the new resource setting, using the following format:

```
!Cursor
!Name: xdt3.name
!Class: XDesktop3.Cursor.Bitmap
*resource_name.data: bitmap_filename

!Mask
!Name: xdt3.name
!Class: XDesktop3.Cursor.Bitmap
*resource_name.mask: bitmap_filename
```

   When you are finished, save your changes and exit the resource file.
4. If users are running the Desktop at the time you make cursor resource changes, they must restart the Desktop to see the new cursors.

Step 1: Creating a picture directory

The default Desktop cursors are located in the default picture directories, */usr/lib/X11/IXI/XDesktop/bitmaps/xdt_c_[large | small]*. You can look in the files located in these directories to see sample bitmap files and filenames, but do not edit the files in this directory. If you want to edit a default Desktop cursor, copy the default bitmap file, then make desired changes to the copy.

In addition, instead of adding your new bitmap file(s) to one of these directories, it is recommended that you create your own picture directory and store the customized bitmaps in this new directory. The picture directory can be located anywhere, but make sure it is in a logical place so that subsequent bitmap files can be stored in this directory also. For example, it is probably logical to create a directory in a system path to store cursors that affect the Desktop system-wide. A recommended location would be in */usr/include/X11*; for example, creating a directory called */usr/include/X11/Newbitmaps* would probably be more logical than creating a directory in your **$HOME** directory path. Nevertheless, you can designate any directory to be the new bitmap directory.

Step 2: Editing the client resource file

The default resource file for the Desktop is */usr/lib/X11/app-defaults/XDesktop3*. The resources in this file are read when the Desktop is invoked. If you want to specify Desktop resources that affect all users on your system, this is the file to edit.

NOTE If you intend to modify the *XDesktop3* file, it is recommended that you either make a backup copy of the file before you enter your resource changes, or comment out old resource values before entering new ones. This way, you are assured of regaining the default values, if needed. Note also that a risk of editing a system resource file is that changes can be overwritten if your system is reinstalled.

See also:

- "Methods for specifying resources" (page 87) for a more detailed explanation of the files used for resource specifications

Step 3: Setting the cursor resources

There are two different resource specifications you might need to set: the cursor resource specification, which consists of the pair of resources ***resource_name*.data** and ***resource_name*.mask**, and the picture directory specification, which involves setting the **pictureDirectory** resource.

Cursor resource specifications must use the following format:

```
!Cursor
!Name: xdt3.name
!Class: XDesktop3.Cursor.Bitmap
*resource_name.data: bitmap_filename

!Mask
!Name: xdt3.name
!Class: XDesktop3.Cursor.Bitmap
*resource_name.mask: bitmap_filename
```

The lines preceded by an exclamation mark "!" are comments. It is important to comment your new resources so that you or other users can understand the intent of the specifications.

***resource_name*.data** represents the resource name of the data pixmap. See Table 9-1, "Resources that control Desktop cursor appearance" (page 175) for a list of valid cursor names. The data pixmap defines the image associated with the cursor, for example, a hand or an hourglass. ***resource_name*.mask** represents the resource name of the mask pixmap. The mask pixmap defines the shape, or the outline, upon which the data pixmap is drawn. The mask is needed so that the image shows up on any color of background.

bitmap_filename refers to the name and location of the new bitmap file. You can indicate where the new bitmap file is located by specifying the new bitmap location with an absolute pathname in both the **.data** and **.mask** cursor resources. For example, if your data and mask pixmap files are located in */usr/include/X11/Picture* in files *hand_d_xbm* and *hand_m_xbm*, respectively, and you want these pixmaps to represent the idle cursor on the Desktop, specify their location by adding the following to the cursor resource:

***idle.data : /usr/include/X11/Pictures/handbit_d_xbm**
***idle.mask : /usr/include/X11/Pictures/handbit_m_xbm**

If you do not indicate an absolute pathname in the **.data** and **.mask** resources, you must specify the **pictureDirectory** resource.

The **pictureDirectory** resource contains the list of directories that are searched when a picture file with a relative name is specified. This list is searched sequentially, so the most frequently accessed directories should be placed at the beginning of the list.

If you do not specify an absolute pathname for the **.data** and **.mask** cursor resources, you must add the pathname of the new picture directory (the directory where your bitmap file is located) to the **pictureDirectory** resource in the *XDesktop3* resource file. By default, the resource looks like this:

```
!Name: xdt3.pictureDirectory
!Class: XDesktop3.PictureDirectory
!Default: No default value.
*pictureDirectory: $XDTBITMAP/xdt_c_large \
$HOME/.xdt_dir/bitmaps/xdt_large \
$XDTBITMAPS/xdt_large \
/usr/include/X11/bitmaps
```

The first directory indicated is the first path searched, when relative pathnames are specified in cursor resources.

Add the absolute pathname of the directory created in Step 1 (page 179) to the beginning of this resource. For example, if you decide to store your bitmaps in a directory called */usr/include/X11/Pictures*, make sure this directory is the first directory indicated. The resource line is similar to this:

```
*pictureDirectory: /usr/include/X11/Pictures \
$XDTBITMAP/xdt_c_large \
$HOME/.xdt_dir/bitmaps/xdt_large \
$XDTBITMAPS/xdt_large \
/usr/include/X11/bitmaps
```

See also:

- Chapter 5, "Understanding resources" (page 79)

Step 4: Activating the new cursors

Once you have made the desired cursor changes to the Desktop resource file, the new specifications are immediately available to all users. However, if users were running the Desktop while you made your changes, they need to restart the Desktop before they see the new cursors.

Specifying Desktop cursors for individual users

Individual users can use their own unique set of cursors on the Desktop. These cursors do not change the default cursors that are available to other users on the system.

To change the cursor for an individual user, perform the following steps. For more information on each of these steps, see the sections immediately following this procedure.

1. Create a customized picture directory and move the new bitmap file to this directory.
2. Create a file called *XDesktop3* in the user's home directory.
3. Edit the *XDesktop3* file and add the new resource specifications, using the following format:

```
!Cursor
!Name: xdt3.name
!Class: XDesktop3.Cursor.Bitmap
XDesktop3*resource_name.data: bitmap_filename

!Mask
!Name: xdt3.name
!Class: XDesktop3.Cursor.Bitmap
XDesktop3*resource_name.mask : bitmap_filename
```

   When you are finished, save your changes and exit the resource file.
4. If you created the *XDesktop3* file as *root*, assign the appropriate user permissions to the file:

   chown ***username*** **XDesktop3**
   chgrp ***groupname*** **XDesktop3**

5. Restart the Desktop.

Step 1: Creating a picture directory

The default Desktop cursors are located in the default picture directory, */usr/lib/X11/IXI/XDesktop/bitmaps/xdt_c_[large | small]*. You can look in the files located in this directory to see the contents of sample bitmap files and sample filenames, but do not edit the files in this directory. If you want to edit a default Desktop cursor, copy the default bitmap file, then make desired changes to the copy.

In addition, instead of adding your new bitmap file(s) to one of these directories, it is recommended that you store customized bitmaps in **$HOME**/*.xdt_dir/bitmaps*.

Step 2: Creating an XDesktop3 file

Individual users can assign their own values to Desktop cursor resource specifications. You can either change the value of a resource already set, or you can set an entirely new resource. User defaults always override system defaults, allowing different users running the same clients to specify personal preferences.

Individual resource settings are placed in a file called *XDesktop3* in the user's home directory. If this file does not already exist, you can create it in one of two ways:

- Create a file named *XDesktop3*, then add the desired cursor specifications to the file. This approach is most useful if you are only making a few changes or only adding a few lines.
- Copy the *XDesktop3* default file, */usr/lib/X11/app-defaults/XDesktop3*, to a file called *XDesktop3* in the user's home directory. You can then use the file as a template, deleting resources you do not want to change and specifying new cursor values.

NOTE If you create an *XDesktop3* file by copying the relevant lines from the *XDesktop3* file in the *app-defaults* directory, be sure to add **XDesktop3** (the Desktop's class name) or **xdt3** (the Desktop's binary name) to the beginning of each line if it is not already there.

When the user invokes the Desktop, the X server checks to see if an *XDesktop3* file exists in **$HOME**. If such a file does exist, the resource values specified in the user resource file take precedence over any values assigned to the same resource in the resource database.

See also:

- "Methods for specifying resources" (page 87) for more information on the *XDesktop3* file

Step 3: Setting the cursor resources

There are two different resource specifications you may need to set: the cursor resource specification, which consists of the pair of resources ***resource_name*.data** and ***resource_name*.mask**, and the picture directory specification, which involves setting the **pictureDirectory** resource.

Cursor resource specifications must use the following format:

```
!Cursor
!Name: xdt3.name
!Class: XDesktop3.Cursor.Bitmap
XDesktop3*resource.data: bitmap_filename

!Mask
!Name: xdt3.name
!Class: XDesktop3.Cursor.Bitmap
XDesktop3*resource.mask: bitmap_filename
```

The lines preceded by an exclamation mark " ! " are comments. It is important to comment your new resources so that you or other users can understand the intent of the specifications.

***resource_name*.data** refers to the resource name of the data pixmap. See Table 9-1, "Resources that control Desktop cursor appearance" (page 175) for a list of valid cursor names. The data pixmap defines the image associated with the cursor, for example, a hand or an hourglass. ***resource_name*.mask** refers to the resource name of the mask pixmap. The mask pixmap defines the shape, or the outline, upon which the data pixmap is drawn. The mask is needed so that the image shows up on any color of background.

bitmap_filename refers to the name and location of the new bitmap file. You can indicate where the new bitmap file is located by specifying the new bitmap location with an absolute pathname in both the **.data** and **.mask** cursor resources. For example, if your data and mask pixmap files are located in **$HOME/***.xdt_dir/bitmaps/Pictures* in files *hand_d_xbm* and *hand_m_xbm*, respectively, and you want these pixmaps to represent the idle cursor on the Desktop, specify their location by adding the following to the cursor resource:

XDesktop3*idle.data: $HOME/.xdt_dir/bitmaps/Pictures/handbit_d_xbm
XDesktop3*idle.mask: $HOME/.xdt_dir/bitmaps/Pictures/handbit_m_xbm

If you do not indicate an absolute pathname in the **.data** and **.mask** resources, you must specify the **pictureDirectory** resource.

The **pictureDirectory** resource contains the list of directories that are searched when a picture file with a relative name is specified. This list is searched sequentially, so the most frequently accessed directories should be placed at the beginning of the list.

If you do not specify an absolute pathname for the **.data** and **.mask** cursor resources, you must add the pathname of the new picture directory (the directory where your bitmap file is located) to the beginning of the **pictureDirectory** resource in the **$HOME/***XDesktop3* file. The resource would now look similar to this:

```
!Cursor
!Name: xdt3.pictureDirectory
!Class: XDesktop3.PictureDirectory
XDesktop3*pictureDirectory: $XDTBITMAP/xdt_c_large \
$HOME/.xdt_dir/bitmaps/xdt_large \
$XDTBITMAPS/xdt_large \
/usr/include/X11/bitmaps
```

The first directory indicated is the first path searched, so the new *Pictures* directory is now searched first. The other directories in the resource are the directories searched sequentially by default when relative pathnames are specified.

See also:

- Chapter 5, "Understanding resources" (page 79)

Step 4: Assigning correct ownership permissions

If you generated an *XDesktop3* file for a user from the *root* account, whether by creating the file or by copying the */usr/lib/X11/app-defaults/XDesktop3* file, you must assign the file the correct ownership permissions. Run the **chown** command to assign the correct owner, and the **chgrp** command to assign the correct group to the **$HOME**/*XDesktop3* file.

If you created your own *XDesktop3* file, you can ignore this step. Your ownership permissions are already correct.

Step 5: Restarting the Desktop

If you are not running a Graphical Environment session, you must start the Desktop before your new resource values can be recognized.

If you are running a Graphical Environment session, you must restart the Desktop before your new resource values can be recognized. Use the **Restart Desktop Session** option on the main Desktop **File** menu.

Changing the scoterm cursor

The pointer cursor in the **scoterm** window is a font. Fonts are X resources and as such, can be changed like any other resource. You have a great deal of control over the fonts used for the pointer cursors in your **scoterm** window. Before you actually make any font resource changes, however, you should be familiar with the information in Chapter 5, "Understanding resources" (page 79).

As with other resources, you can:

- make font resource changes for **scoterm** pointer cursors on a system-wide level (page 187) or on a user level (page 188)
- specify font changes at the command line (page 190)

Specifying scoterm cursors for the entire system

To make a pointer cursor font change, perform the following steps. You must be logged into the system as *root* to perform this task. For more information on each of the steps in this procedure, see the sections immediately following this list.

1. Open the *ScoTerm* resource file in the */usr/lib/X11/app-defaults* directory for editing.
2. Make the desired font resource specification(s):
   - For an existing font resource specification, replace the old font designation with the desired font.
   - For a new font resource specification, add the new resource setting, using the following format:

     ScoTerm*pointerShape: ***fontname***

   When you are finished, save your changes and exit the resource file.
3. If users are running the **scoterm** client at the time you make these changes, they must restart the client to see the new cursor.

Step 1: Editing the system resource file

The default resource file for **scoterm** is named *ScoTerm* and is located in the */usr/lib/X11/app-defaults* directory. The resources in this file are read whenever the **scoterm** client is run.

NOTE If you intend to modify the *ScoTerm* file, it is recommended that you either make a backup copy of the file before you enter your resource changes, or comment out old resource values before entering new ones. This way, you are assured of regaining the default values, if needed.

See also:

- "Methods for specifying resources" (page 87) for a more detailed explanation of the files used for resource specifications

Step 2: Setting the cursor font resource

As with all resources, cursor font resource specifications must use the correct format:

ScoTerm*pointerShape: ***cursorname***

When you specify the **scoterm** client, you can enter either the binary name, **scoterm**, or the class name, **ScoTerm**. **pointerShape** is the resource variable you are setting. It is part of the resource class, **Cursor**. ***cursorname*** is the actual name of the cursor you are selecting. Use the cursor names listed in Table 9-2, "Standard cursor font names" (page 176).

See also:

- Chapter 5, "Understanding resources" (page 79)
- Chapter 7, "Changing fonts" (page 125)

Step 3: Activating the new cursor

Once you have made the desired cursor changes to the **scoterm** resource file, the new specifications are immediately available to all users. However, if users were running a **scoterm** window while you made your changes, they need to restart the client before they see the new cursors.

Specifying scoterm cursors for individual users

Individual users can use their own unique set of pointer cursors for **scoterm**. These font settings do not change the default pointer cursors that are used by other users on the system.

To change **scoterm** cursor fonts for an individual user, perform the following steps. For more information on each of these steps, see the sections immediately following this procedure.

1. Create a file called *.Xdefaults-**hostname*** in the user's home directory.
2. Edit the *.Xdefaults-**hostname*** file and add the new cursor font resource specification, using the following format:

   ScoTerm*pointerShape: ***cursorname***

   When you are finished, save your changes and exit the resource file.

3. If you created the *.Xdefaults-**hostname*** file as *root*, assign the appropriate user permissions to the file:

   chown ***username*** **.Xdefaults-*hostname***
   chgrp ***groupname*** **.Xdefaults-*hostname***

4. If necessary, restart the **scoterm** client.

Step 1: Creating an .Xdefaults-hostname file

Individual users can assign their own values to cursor font resource specifications. You can either change the value of a font resource already set in the resource database, or you can set an entirely new font resource. User defaults always override system defaults, allowing different users running the same clients to specify personal font preferences.

Individual resource settings are placed in a file called *.Xdefaults-**hostname***, where ***hostname*** is the name of the host, or machine, where the client is running.

You can add an *.Xdefaults-**hostname*** file to a user's home directory in one of two ways:

- Create a file named *.Xdefaults-**hostname***, then add the desired cursor font specifications to the file. This approach is most useful if you are only making a few changes.
- Copy the **scoterm** default file, */usr/lib/X11/app-defaults/ScoTerm*, to the user's home directory, into a file called *.Xdefaults-**hostname***. You can then use the file as a template, keeping relevant lines specifying a new font value for the **pointerShape** resource.

NOTE If you create an *.Xdefaults-**hostname*** file by copying the relevant lines from the *ScoTerm* file in the *app-defaults* directory, be sure to add **scoterm** or **ScoTerm** to the beginning of each line if it is not already there.

When the user invokes **scoterm**, the X server checks to see if an *.Xdefaults-**hostname*** file exists in **$HOME**. If such a file does exist, the resource values specified in the user resource file take precedence over any values assigned to the same resource in the resource database.

See also:

- "Methods for specifying resources" (page 87) for more information on the *.Xdefaults-**hostname*** file

Step 2: Setting the font resource

Font resource specifications must use the correct format:

ScoTerm*pointerShape: ***cursorname***

When you specify the **scoterm** client, you can enter either the class name, **ScoTerm**, or the binary name, **scoterm**. **pointerShape** is the resource variable you are setting. It is part of the resource class, **Cursor**. ***cursorname*** is the actual name of the cursor you are selecting. Use the font names listed in Table 9-2, "Standard cursor font names" (page 176).

See also:

- Chapter 5, "Understanding resources" (page 79)

Step 3: Assigning correct ownership permissions

If you generated an *.Xdefaults-**hostname*** file for a user from the *root* account, whether by creating the file or by copying the *ScoTerm* file from */usr/lib/X11/app-defaults*, you must assign the file the correct ownership permissions. Run the **chown** command to assign the correct owner, and the **chgrp** command to assign the correct group to the *.Xdefaults-**hostname*** file:

chown ***username*** **.Xdefaults-*hostname***
chgrp ***groupname*** **.Xdefaults-*hostname***

If you created your own *.Xdefaults-**hostname*** file, you can ignore this step. Your ownership permissions are already correct.

Step 4: Restarting the scoterm client

If you are running a Graphical Environment session and you used the **scoterm** window to make the desired changes to the resource file, you will not see the effects of the changes until you exit **scoterm** and invoke it again.

Setting scoterm cursors from the command line

You can specify any resource setting on the command line that you would otherwise put into a resource file, including font resources.

To change your pointer cursor from the command line, use the **-xrm** option when launching **scoterm** from a **scoterm** window:

scoterm -xrm 'ScoTerm*pointerShape: ***cursorname'*** **&**

If you specify a font resource from the command line, the pointer cursor font is only set for that instance of **scoterm**. Subsequent sessions return to the cursor specified in the resource files for **scoterm**.

NOTE If you are using **scosession** and you end your current session with a client that was run from the command line still open on the display, the client is restored when you resume your session. Any command line options that were used to define the client are also restored. In this way, command line options can define client behavior on a more permanent basis.

See also:

- "Using the -xrm option" (this page)

Using the -xrm option

The **-xrm** option allows you to set any resource value, including font specifications, from the command line. You must enter the resource specification as well as the desired font name when using this option. For example, to change the pointer cursor font for **scoterm** to a double arrow, enter the following at the command line:

scoterm -xrm 'ScoTerm*pointerShape: double_arrow' &

When using the **-xrm** option, you should be aware of the following:

- A resource specification must be quoted using the single quotation mark "'".
- You can specify more than one resource value from the command line at the same time. You must enter the **-xrm** option for each specification you make.

Resource values specified at the command line only exist for the **scoterm** session you are invoking. The resource values do not become part of the resource database and the values are not recreated the next time you run **scoterm** unless you use the **-xrm** option again.

Note that a font resource specified at the command line does not take effect if a font resource that takes precedence has already been specified. For example, say you loaded a resource file that includes the specification:

```
ScoTerm*pointerShape: heart
```

In this case, the following command line specification would not result in the **scoterm** client using the star pointer cursor font:

scoterm -xrm '*pointerShape: star' &

Because the **ScoTerm*pointerShape** designation is more specific, the ***pointerShape** entry on the command line does not override the font setting in the resource database.

To override the resource database, and get the star pointer cursor, you would need to use a resource equally or more specific than the default setting. For example, the resource **ScoTerm*pointerShape** would provide the desired pointer cursor change.

Example of changing cursor appearance

This section provides two examples that tie together many of the concepts and procedures discussed throughout this chapter. The examples in this section describe:

- how to change a user's Desktop cursor (this page)
- how to change a system-wide **scoterm** cursor (page 194)

Example 1: Changing Desktop cursor appearance

Let's assume you are a system administrator. One of the user's on your system has created a new bitmap with **scopaint**(XC) and wants it to display each time the Desktop processor is busy. The user has named the data pixmap file *waiting_d.xbm* and the mask pixmap file *waiting_m.xbm*. Both files are temporarily stored in the user's **$HOME** directory.

The following steps result in a new Desktop cursor that appears in the user's Graphical Environment when the Desktop processor is busy. Either *root* or the user can make this change. Let's assume you do this change for the user as the system administrator, logged in as *root*.

1. Change directories to the user's **$HOME** directory and locate the files named *waiting_d.xbm* and *waiting_m.xbm*. The user's **$HOME** directory is not a very appropriate place to store Desktop cursors, so you decide these files need to be moved to a more appropriate picture directory.
2. Create a picture subdirectory called */Picture* in the user's **$HOME**/*.xdt_dir/bitmaps* directory, and move *waiting_d.xbm* and *waiting_m.xbm* to this new directory.
3. Now look for the file named **$HOME**/*XDesktop3*. For the purposes of this example, let's assume this file already exists. (You will copy the relevant lines from the Desktop default resource file, */usr/lib/X11/app-defaults/XDesktop3*, into this *XDesktop3* file.)
4. Change directories to */usr/lib/X11/app-defaults*, then search the *XDesktop3* file for the lines that contain the resource that displays the busy cursor on the Desktop, ***busy.data** and ***busy.mask**.
5. Copy the lines that contain the cursor value for this resource to **$HOME**/*XDesktop3*, and comment out the the two lines containing the cursor value so that the file now looks like this:

```
! Name: xdt3.busy.data
! Class: XDesktop3.Cursor.Bitmap
! Default: Internal picture
!*busy.data : wait_d.xbm
```

   and

```
! Name: xdt3.busy.mask
! Class: XDesktop3.Cursor.Bitmap
! Default: Internal picture
!*busy.mask : wait_m.xbm
```

6. Now open a line immediately below both resources, and enter your new resource designations:

   XDesktop3*busy.data: $HOME/.xdt_dir/bitmaps/Pictures/waiting_d.xbm
   XDesktop3*busy.mask: $HOME/.xdt_dir/bitmaps/Pictures/waiting_m.xbm

7. Save and exit the *XDesktop3* file.
8. When the user runs the Desktop, and the Desktop processor is busy, the user's new cursor should appear.

Example 2: Changing scoterm cursor appearance

Let's assume you are an administrator for a system whose X server and clients are accessed by several users. Your company just completed a deal with a major boat manufacturer and, to celebrate, the president of your firm has requested that the pointer cursor in the **scoterm** window be changed to a boat-shape.

The following steps result in a system-wide boat shaped pointer cursor for the **scoterm** client.

1. Log into the system as *root*.
2. Change directories to */usr/lib/X11/app-defaults* and locate the file named *ScoTerm*. This is the default resource file that you need to edit to make your pointer cursor resource change.
3. Open the *ScoTerm* file for editing and search for the following line:

   ```
   ScoTerm*pointerShape:
   ```

   This is the resource that controls the font that is used to display the pointer cursor in the **scoterm** window.

   If this resource does not exist, skip to Step 5.
4. If this resource exists, comment out the line containing the font value for this resource so that it looks like this:

   ```
   !ScoTerm*pointerShape: cursorname
   ```

   By commenting out the default resource setting, instead of simply deleting the line, you leave yourself a safeguard. You can always return to this default if you make a mistake when setting a new font value.
5. Now open a line immediately below the resource you just commented out (or if a resource was not previously set, open a line anywhere in the file) and enter your new resource designation:

   ```
   ScoTerm*pointerShape:  boat
   ```
6. Save and exit the *ScoTerm* file.
7. As a final test, invoke **scoterm**. (If you are currently running a Graphical Environment session, exit **scoterm**, then invoke it again.) The pointer cursor that displays in the **scoterm** window should now look like a boat.

Chapter 10
Configuring mouse behavior

There are several aspects of your mouse's behavior that you can specify. You can:

- emulate a three-button mouse if your mouse only has two buttons (page 196)
- configure your mouse for right- or left-handed use (page 196)
- specify the rate of acceleration (page 198) at which the mouse cursor (also called the mouse pointer) moves across the screen and the number of pixels the cursor must move before the mouse begins accelerating (page 199)
- define a new duration that is allowed between the two clicks that constitute a double-click action (page 201)

There is also an example (page 206) at the end of this chapter that helps tie together many of the concepts and procedures discussed in this chapter.

The steps for performing these configuration tasks are discussed in this chapter. Note that a number of these tasks should be completed by individual users using the **scomouse** client. The super-user cannot configure mouse behavior on a system-wide level.

See also:

- "Changing mouse characteristics" (page 33)

Emulating a three-button mouse

In previous releases of SCO Open Desktop, if you wanted to emulate three-button mouse behavior with a two-button mouse, you had to remap mouse triggers by modifying several resource lines in your personal resource file. For this release, however, you do not have to make any configuration changes to obtain this functionality.

If you have a correctly installed two-button mouse on your system, the operating system automatically maps the mouse buttons so you can then perform the same tasks as a user with a three-button mouse.

When you use a two-button mouse, the left button on the mouse performs all the functions of mouse button 1, and the right button on the mouse performs the functions of mouse button 3. To simulate mouse button 2, click both buttons simultaneously. This technique is also referred to as "chording."

Switching to a left-handed mouse

By default, your mouse is configured for use by right-handed users. The left mouse button assumes the behavior of mouse button 1, the middle button behaves as mouse button 2, and the right button behaves as mouse button 3. If you are left-handed, however, you can reverse the button order of your mouse so that the mouse button on the right is treated as mouse button 1, and so forth.

To switch your mouse for left-handed use, perform the following steps. For more information on each of these steps, see the sections immediately following this procedure.

1. Use *one* of the following configuration tools:
   - Run the **scomouse** client by double-clicking on the **Mouse** icon, located in the **Preferences Editor**, in the *Controls* window, and click on the **Left Handed** button.
   - Run the **xmodmap** command from a **scoterm** window:

     xmodmap -e "pointer = 3 2 1"

2. List the new mouse button mappings to verify that your mouse is now configured for left-handed use:

   xmodmap -pp

Step 1: Configuring the mouse for left-handed use

The **scomouse** client is very easy to use and is generally the preferred method for configuring your mouse for left-handed use. When the client is running, simply click mouse button 1 on the **Left Handed** button. The remapping of the mouse buttons is immediate, and you can now use the right mouse button as mouse button 1. This setting stays in effect until you change it.

The **scomouse** client stores this configuration information in a file called *ScoMouse*, located in **$HOME**/*.odtpref*. If you examine this file after you make your selection, you see:

```
scomouse -s 2 -t 4 -l
```

The `-s` and `-t` entries refer to the acceleration and threshold values the mouse uses. These issues are described in "Configuring mouse acceleration" (page 198). The `-l` entry indicates the mouse is configured for left-handed use.

You can also switch to a left-handed mouse using the **xmodmap** command with the **-e** option.

NOTE If you set your mouse for left-handed use with the **xmodmap -e** command, the configuration is only in effect for the current session, unless you add the **xmodmap -e** command to an executable script in your **$HOME**/*.odtpref* directory.

The **scomouse** client is more intuitive to use than the **xmodmap -e** command, and is the preferred method for configuring your mouse for left- or right-handed use for multiple sessions.

To configure the mouse for left-handed use with **xmodmap**, enter:

xmodmap -e "pointer = 3 2 1"

The mouse button mappings are now reversed so that the button on the right behaves as mouse button 1, and so forth.

To return to a right-handed mouse using **xmodmap**, enter:

xmodmap -e "pointer = 1 2 3"

See also:

- **xmodmap**(X) manual page

Step 2: Listing the new mouse button mappings

If you used the **scomouse** client to configure your mouse, you know that your selection worked because the behavior of the mouse changes automatically.

However, if you switched your mouse with the **xmodmap** command, you may want to verify that the mouse buttons are now mapped correctly for left-handed use. Run the following command:

xmodmap -pp

If your configuration succeeded, you should see:

```
There are 3 pointer buttons defined.

    Physical        Button
     Button          Code
        1              3
        2              2
        3              1
```

The Physical Button column indicates the actual buttons on the mouse, where the button on the left is button 1, the middle button is button 2, and the right button is button 3. The Button Code column indicates the current mouse button mappings.

The example indicates that the mouse is now configured for left-handed use. The left button, or physical button 1, performs the function of mouse button 3, and so forth.

Configuring mouse acceleration

You can configure the rate at which the mouse cursor moves across the screen and the point at which the mouse cursor actually begins to accelerate. By configuring these parameters, you can use the mouse for precise positioning when it is moved slowly and you also have the flexibility to move the cursor quickly across the screen.

To configure the mouse's movement parameters, perform the following steps. For more information on each of these steps, see the sections immediately following this procedure.

1. Use one of the following configuration tools:
   - Run the **scomouse** client by double-clicking on the **Mouse** icon, located in the **Preferences Editor** in the *Controls* window, and adjust the "Acceleration" and "Distance" slider bars to the desired levels.
   - Run the **xset** command, with the **m** (mouse) option, from a **scoterm** window:

     xset m ***acceleration_value threshold_value***

2. Verify that your changes were successfully implemented by running the following command:

   xset q

Step 1: Setting the movement parameters

There are two parameters that control the movement of the mouse: acceleration and threshold. The acceleration parameter is a multiplier that is applied to the mouse cursor motion. If the acceleration parameter is set to "4", the mouse cursor moves across the screen four times faster than you move the mouse.

Setting a high acceleration is convenient for quickly moving the mouse cursor large distances on your screen. However, it can be awkward when you want to position the mouse precisely. The pointer can move too quickly and it becomes difficult to focus the mouse cursor on a small area on your screen. To overcome this problem, you can set the threshold parameter. The threshold controls the number of pixels the mouse cursor must move before the cursor motion accelerates.

For example, suppose you set an acceleration value of "6" and a threshold value of "8". The mouse is configured so that if you move the mouse cursor more than 8 pixels, the cursor can then move 6 times as fast on the screen as you actually move the physical mouse.

Use the following guidelines for deciding how to configure the mouse movement parameters:

- increase the acceleration parameter if you have a high resolution display or are using a monitor or X terminal with a large screen
- increase the threshold value if you increase the acceleration value
- set the threshold parameter fairly high if you have a high resolution display

You can use either the **scomouse** client or the **xset** command to configure these parameters.

The **scomouse** client is generally the preferred method for setting the mouse movement parameters:

- The acceleration parameter is set using the "Acceleration" slider bar. The further to the right you set the bar, the more the acceleration.
- The threshold parameter is set using the slider bar labeled "Distance moved before mouse accelerates." The left-most position allows the mouse to accelerate almost immediately upon being moved. Moving the slider further to the right results in a longer delay before the mouse cursor begins accelerating.

The parameters you set with the slider bars are implemented immediately and remain in effect until you change them again through the **scomouse** client. This allows you to test the mouse movement and fine-tune it to your personal preference.

The **scomouse** client stores the settings for these parameters in a file called *ScoMouse*, located in **$HOME/***.odtpref*. If you examine this file after you set new values for the movement parameters, you see an entry similar to this:

```
scomouse -s 7 -t 10 -r
```

The `-s` entry indicates that the acceleration of the mouse has been set to 7 and the `-t` entry indicates that the threshold has been set to 10. The `-r` entry indicates the mouse is configured for right-handed use.

You can also set the acceleration and threshold parameters using the **xset** command, with the **m** (mouse) option.

NOTE Mouse movement parameters configured with the **xset m** command are only in effect for the current session, unless you add the **xset m** command to an executable script in your **$HOME/***.odtpref* directory.

The **scomouse** client is more intuitive to use than the **xset m** command, and is the preferred method for configuring mouse movement behavior for multiple sessions.

To configure the mouse movement parameters with **xset**, enter:

xset m ***acceleration_value threshold_value***

acceleration_value refers to the value you want for the acceleration parameter, and ***threshold_value*** refers to the value you want for the threshold parameter. You must use positive integers to set these parameters.

If you only assign the **xset m** command one value, the value is assigned to the acceleration parameter. If you want to change the threshold parameter and leave the acceleration parameter unchanged, enter:

xset m default ***threshold_value***

If you want to return to the default acceleration and threshold values, enter:

xset m default

The system defaults specify that the mouse uses an acceleration value of 4 and a threshold value of 2.

See also:

- **xset**(X) manual page

Step 2: Listing the new mouse settings

Once you have configured the mouse movement parameters, you may want to verify that your changes were successfully implemented by the X server. Enter the following command:

xset q

The **q** option lists the current values of all of the **xset** preferences. The mouse movement parameters are included in this list, as shown in the following example:

```
Pointer Control:
  acceleration:   3 / 1     threshold:  4
```

This indicates that the mouse cursor on the screen moves three times as fast as you actually move the physical mouse (3/1), and that this occurs after the mouse pointer moves 4 pixels. The default is twice as fast (2/1).

Specifying the mouse double-click duration

You can modify the duration that is allowed between the two mouse clicks that constitute a double-click action. There is a specific period of time, in milliseconds, that is used to judge if two mouse button presses are a double-click or two single clicks. This behavior is defined through resources, and as such, can be modified, either by using **scomouse** or by editing the Desktop resource files.

You may find that the default settings require you to double-click the mouse button more quickly than you would prefer. Perhaps you commonly double-click on an icon, only to find that you have selected the icon (instead of executing it) because the Desktop interpreted your action as two separate single-click actions. Or, you may feel that the default settings allow for too much time between the clicks that constitute a double-click action.

The following sections describe how to define the double-click duration:

- using **scomouse** (this page)
- for the Desktop only (this page)
- for the window manager only (page 204)

You will probably need to experiment with different settings before you find a designation with which you are satisfied.

Defining the double-click duration with scomouse

Using the **scomouse** client, adjust the "double-click duration" slider bar. Moving the slider to the right increases the amount of time allowed between clicks. Moving the slider to the left decreases the amount of time allowed for a double-click.

Defining the double-click duration for the Desktop

The following steps describe how to modify the double-click duration by editing the appropriate resource files. For more information on each of these steps, see the sections immediately following this procedure.

1. Open the desired resource file for editing:
   - */usr/lib/X11/app-defaults/XDesktop3* for system-wide changes
   - **$HOME**/*XDesktop3* for individual changes
2. Add the Desktop double-click resource specification, using the following format:

   XDesktop3.triggers.maxUpTime: ***resource_value***

3. If desired, you can further customize mouse behavior on the Desktop with the following resources:
   - Set the **XDesktop3.triggers.thresholdDownTime** resource to control the time, in milliseconds, that a mouse button can be held down before it is considered a hold action instead of a click.
   - Set the **XDesktop3.triggers.maxMotion** resource to control the distance, in pixels, that the mouse cursor can move before a mouse button press is considered a drag action.

   When you have finished, save your changes and exit the resource file.
4. Restart the Desktop.

Step 1: Editing the resource file

You can change the duration allowed between the mouse button presses in a double-click action so that all users on your system can use the new behavior, or you can specify a new duration for an individual user only.

The majority of default Desktop resource settings are defined in the */usr/lib/X11/app-defaults/XDesktop3* file. The resources in this file are read by the resource manager when the Desktop is executed. If you want to modify the time allowed for double-clicks for all users, you should edit this file. You must have *root* privileges to edit this file. It is also a good idea to make a backup copy of this file before making changes to it.

If they do not use **scomouse**, individual users can also change this behavior using their personal Desktop resource file, **$HOME**/*XDesktop3*. When the user starts the Desktop, it checks to see if an *XDesktop3* file exists in **$HOME**. If the file does exist, the resource values specified in the user resource file take precedence over any values assigned to the same resource for the system, or in the resource database.

See also:

- "Methods for specifying resources" (page 87) for a more detailed explanation of the resource files

Step 2: Setting the maxUpTime resource

Use the **triggers.maxUpTime** resource to specify the duration allowed between clicks so the action is interpreted as a double-click. The default value of this resource is 500 milliseconds.

Use the following format to set this resource:

XDesktop3.triggers.maxUpTime: ***resource_value***

For example, if you want to specify a duration of "1,000" milliseconds, set the **triggers.maxUpTime** resource so it reads:

XDesktop3.triggers.maxUpTime: 1000

Step 3: Modifying other Desktop mouse resources

There are two other resources that you can set to refine the interaction of your mouse with the Desktop client:

- The **triggers.thresholdDownTime** resource controls the time, in milliseconds, that a mouse button can be held down before it is considered a hold action instead of a click. The default value for this resource is "700" milliseconds.

- The **triggers.maxMotion** resource controls the distance, in pixels, that the mouse cursor can move before a mouse button press is considered a drag action. The default value for this resource is "3" pixels.

Step 4: Restarting the Desktop

Once you have made the desired resource changes, you need to restart the Desktop so the newly defined values will be read. Select **Restart Desktop Session** from the main Desktop **File** menu. In the **Restart** dialog box, click on **Yes**. The Desktop starts again and reads your new resource values.

Defining the double-click duration for the window manager

To modify the time allowed between the clicks that constitute a double-click action for the window manager, perform the following steps. For more information on each of these steps, see the sections immediately following this procedure.

1. Open the desired resource file for editing.
   - **pmwm** mode:

     /usr/lib/X11/app-defaults/Pmwm for system-wide changes
   - **mwm** mode:

     /usr/lib/X11/app-defaults/Mwm for system-wide changes
   - both modes:

     $HOME/.*Xdefaults-**hostname*** for local changes
2. Add the window manager double-click resource specification, using the following format:

   Pmwm*doubleClickTime: ***resource_value***

   or

   Mwm*doubleClickTime: ***resource_value***

   When you have finished, save your changes and exit the resource file.
3. Restart the window manager.

Step 1: Editing the resource file

You can change the duration allowed between the mouse button presses in a double-click action so that all users on your system can use the new behavior, or you can modify the duration for an individual user.

The majority of window manager resource settings are defined in the */usr/lib/X11/app-defaults/Pmwm* (for **pmwm** mode) or the */usr/lib/X11/app-defaults/Mwm* (for **mwm** mode) resource file. The resources in this file are read by the resource manager when the window manager is executed. If you want to modify the window manager double-click duration for all users, you should edit this file. You must have *root* privileges to perform this step.

Individual users can also change this behavior. Individual resource settings are placed in a file called *.Xdefaults-**hostname***, where ***hostname*** is the name of the host, or machine, where the window manager is running.

NOTE The *.Xdefaults-**hostname*** file does not exist in the user's home directory by default. If this file is not currently present, you must create it before you can redefine the time allowed for mouse double-clicks.

If you create this file for a user from the *root* account, you must assign the file the correct ownership permissions. Run the **chown** command to assign the correct owner and the **chgrp** command to assign the correct group to the *.Xdefaults-**hostname*** file. If you created this file yourself, these steps are unnecessary.

When the user starts the window manager, it checks to see if an *.Xdefaults-**hostname*** file exists in **$HOME**. If the file does exist, the resource values specified in the user resource file take precedence over any values assigned to the same resource for the system, or in the resource database.

See also:

- "Methods for specifying resources" (page 87) for a more detailed explanation of resource files

Step 2: Setting the doubleClickTime resource

Use the **doubleClickTime** resource to specify the duration allowed between clicks so the action is interpreted as a double-click. The default value of this resource is 500 milliseconds.

Use the following format to set this resource:

Pmwm*doubleClickTime: ***resource_value***

or

Mwm*doubleClickTime: ***resource_value***

For example, if you want to specify a duration of "1,000" milliseconds, set the **doubleClickTime** resource so it reads:

Pmwm*doubleClickTime: 1000

or

Mwm*doubleClickTime: 1000

Step 3: Restarting the window manager

After you have specified the new window manager resource value, you must restart the window manager so your changes can take effect. Restart the window manager by selecting the **Restart Window Manager** option from the **Root** menu. The **Root** menu is accessed by pressing and holding mouse button 1 on the Desktop background or, if the Desktop is not running, in the Root window.

Verify that your new double-click designation is being used by the window manager.

Example of configuring your mouse

This section provides a comprehensive example that ties together some of the concepts and procedures discussed in this chapter.

Let's assume you are temporarily working on a system, using a departmental account that is accessed by several users. While most of these users are right-handed, you are left-handed. Let's also assume the system uses a 20-inch, high resolution display, but your colleagues have configured the mouse so it moves more slowly over the large screen than you would prefer.

You want to reconfigure the mouse for left-handed use and speed up the mouse cursor movement but you do not want to overwrite the mouse settings that satisfy the majority of users who access the departmental account.

This example covers how to make these changes for your current session only. The commands used affect only the current session, unlike using **scomouse**, which affects not only the current session but also all subsequent sessions.

NOTE If you want to configure the behavior of your mouse on a more permanent basis, you should use the **scomouse** utility. See "Changing mouse characteristics" (page 33) for more information on using **scomouse**.

The following steps result in a mouse that is temporarily configured for left-handed use and faster movement across the display.

1. Log into the system. If logging in does not automatically start the X server, do so now by entering the following command at the prompt:

   startx &

2. To configure the mouse for left-handed use, enter the following command from a **scoterm** window:

   xmodmap -e "pointer = 3 2 1"

3. When the prompt returns, verify that the mouse buttons are now mapped for left-handed use by entering:

   xmodmap -pp

   You should see the following:

```
There are 3 pointer buttons defined.

    Physical        Button
     Button          Code
        1              3
        2              2
        3              1
```

   You can now use the mouse button on the right to perform all mouse button 1 actions.

4. Now you are ready to increase the rate at which the mouse cursor moves across the screen. For this example, we'll assume you want the mouse cursor to move six times as fast as you move the mouse.

   Because you also want to precisely position the mouse cursor, you need to increase the rate of the mouse's threshold. For this example, we'll set the threshold so the cursor must move over 8 pixels on the screen before the mouse cursor actually accelerates.

   To make these settings, enter the following command from a **scoterm** window:

   xset m 6 8

5. Verify that the X server successfully implemented these new mouse settings by entering:

   xset q

   You see a list of all the current **xset** preferences, including the current mouse movement parameter values:

   ```
   Pointer Control:
     acceleration:   6 / 1     threshold:  8
   ```

   The mouse is now configured to suit your preferences. When you end your session and log out, your temporary mouse configuration settings are removed.

Chapter 11

Configuring the keyboard for the server

When you run an SCO OpenServer Graphical Environment session from the console, all keystrokes are interpreted by the X server before they are passed along to any clients that you are running. Consequently, the keyboard may behave in a completely different manner when your are in a Graphical Environment session than when you use a console multiscreen in text mode.

This chapter describes:

- background on the server keyboard (this page)
- configuring the X server to accommodate different keyboards (page 211)
- modifying the keyboard layout (page 213)

There is also an example (page 216) at the end of this chapter that helps tie together many of the concepts and procedures discussed in this chapter.

About the server keyboard

Every time you press a key on your keyboard while running the X server, the following signals are exchanged:

scancodes — hardware-dependent codes generated by keystrokes and received by the X server

keycodes — codes sent by the X server to the client indicating which key was pressed. Keycodes by themselves do not indicate what the keystroke means; the client must request that information from the X server.

modifiers — flags, such as "shift" and "control," that the server sends with each keycode to the client. The client's interpretation of the keycode may depend on the state of the modifiers. For

example, the state of the shift modifier determines whether an alphabetic character should be lowercase or uppercase. Because the X server encodes the states of the eight modifiers (shift, lock, control, mod1, mod2, mod3, mod4, and mod5) in a single byte of data, modifiers are often referred to as "modifier bits."

keysyms — codes that specify the glyphs appearing on the keys. The X server also maintains a list of strings that describe the keysyms, such as "a", "B", and "Control."

The X server keeps track of the mappings of keycodes to keysyms, and mappings of keysyms to modifiers in two tables in its memory:

keymap table — contains a list of keycodes that the X server sends, and the keysyms and strings that correspond to them. The order of keysyms in the keymap table determines which keysym corresponds to the shifted or unshifted key.

modifier map — contains a list of keysyms and keycodes to which each of the eight modifiers are "attached." For example, the default configuration attaches the mod1 modifier to the Alt_L and Alt_R keysyms and to the keycodes corresponding to the left and right ⟨Alt⟩ keys. When either ⟨Alt⟩ key is pressed, or when any key mapped to the Alt_L or Alt_R keysyms is pressed, the mod1 modifier is on.

To change the behavior of your keyboard while the X server is running, you need only modify these two tables.

Because many non-U.S. keyboards have more than two symbols on each key, the Shift modifier is supplemented by a modifier known as the "group modifier," and the keysym, Mode_Switch. If the keymap table specifies more than two keysyms for a key, the state of the group modifier determines whether the Shift modifier toggles between the first and second keysyms, which are referred to as "group 1," or between the third and fourth keysyms, which are referred to as "group 2." When you configure the X server for a non-U.S. keyboard, **xsconfig.sh** (see the **xsconfig**(X) manual page) maps Mode_Switch to the mod3 modifier. Consequently, any key mapped to Mode_Switch in the keymap table acts as the group modifier.

The X server's initial keyboard configuration is read from a configuration file, *.Xsco.cfg*. If the X server finds *.Xsco.cfg* in the user's home directory, it reads that file. Otherwise, it reads the system-wide default configuration file, */usr/lib/X11/.Xsco.cfg*.

You can modify the contents of the keymap table and modifier map while the X server is running, but the initial mapping of scancodes and keycodes, and of keysyms to strings, can only be modified by compiling a new configuration file. You create keyboard configuration files with the **xsconfig.sh** script, which allows you to create default keyboard configurations for a variety of character sets, languages, and keyboards.

See also:

- **xsconfig**(X) manual page

Changing the modifier map

To change the X server's modifier map while it is running, perform the following steps. For more information on each of these steps, see the sections immediately following this procedure.

1. Change the keysyms attached to a specific modifier by entering the following command in a **scoterm** window:

   xmodmap -e ***expression***

   Repeat this step for every modifier map change you want to make.

2. Check the current modifier map by entering the following command at the prompt in a **scoterm** window:

   xmodmap -pm

Step 1: Changing a modifier map

To change modifier mappings run the following command in a **scoterm** window:

xmodmap -e ***expression***

expression is a quoted string that tells **xmodmap** how to change the modifier map.

To add a keysym to a specific modifier, ***expression*** takes the following form:

"add ***modifier*** **=** ***keysym_name*****"**

Modifiers can be attached to up to two keysyms. ***modifier*** is one of the following: shift, lock, control, mod1, mod2, mod3, mod4, or mod5. ***keysym_name*** is the name of a keysym to which you want to attach ***modifier***. For a list of valid keysyms, examine */usr/lib/X11/xsconfig/keysymdef.h*. Note that you must not include the keysym's "XK_" prefix in ***keysym_list***. Note also that this command does not override any existing modifier attachments; it only adds an attachment.

NOTE You must leave spaces around the " = " character.

For example, if the shift modifier is only mapped to the left ⟨Shift⟩ key, you can attach the shift modifier to the right ⟨Shift⟩ key with the following command:

xmodmap -e "add shift = Shift_R"

To remove a keysym from a specific modifier, ***expression*** takes the following form:

"remove *modifier* = *keysym_name*"

modifier is one of the following: shift, lock, control, mod1, mod2, mod3, mod4, or mod5. ***keysym_name*** is the name of a keysym to which ***modifier*** is already attached. For example, to remove the shift modifier's attachment to the left ⟨Shift⟩ key, use the following command:

xmodmap -e "remove shift = Shift_L"

The above command does not affect the shift modifier's attachment to the right ⟨Shift⟩ key.

Modifiers are usually attached not only to keysyms but also to specific scancodes generated by the keyboard. Although you can change the keysyms to which a modifier is attached, you cannot change the keys to which modifiers are directly attached. If a modifier's attachment to a scancode prevents the modifier mapping you desire, you can clear all of the modifier's attachments with the following command:

xmodmap -e "clear *modifier*"

This command removes all of ***modifier***'s attachments to keysyms and keyboard scancodes. You can then attach the modifier to any desired keysyms. If you need to attach the modifier to a keyboard scancode, however, you must create a new *.Xsco.cfg* file with **xsconfig**(X).

NOTE The modifier map listing generated by **xmodmap -pm** only shows the keysyms, not the keyboard scancodes, to which modifiers are attached. If you want to remap modifiers, be sure to determine which keys the modifiers are attached to by examining the */usr/lib/X11/xsconfig/default.kbd* file or, if you are using a non-U.S. keyboard, the keyboard file from which the configuration file was compiled, */usr/lib/X11/xsconfig/mapkey.kbd*. Note that the scancodes in *default.kbd* correspond to server keycodes minus 7.

xmodmap can also accept commands from a file or standard input.

See also:

- **xmodmap**(X) manual page
- **xsconfig**(X) manual page

Step 2: Examining the modifier map

To view the current modifier map, run the following command at the prompt in a **scoterm** window:

xmodmap -pm

A list of modifiers and the keysyms and keycodes to which they are attached appears. The following is an example modifier map listing:

```
xmodmap:  up to 2 keys per modifier, (keycodes in parentheses):

shift       Shift_L (0x31),  Shift_R (0x3d)
lock        Caps_Lock (0x41)
control     Control_L (0x24),  Control_R (0x87)
mod1        Alt_L (0x3f),  Alt_R (0x88)
mod2        Num_Lock (0x4c)
mod3
mod4
mod5
```

This listing indicates that the shift modifier is attached to the keysyms for both the left and right ⟨Shift⟩ keys.

Changing the keymap table

To change the X server's keymap table while it is running, perform the following steps. For more information on each of these steps, see the sections immediately following this procedure.

1. Check the current keymap table by entering the following command at the prompt in a **scoterm** window:

   xmodmap -pk

2. Change the keysyms attached to a specific keycode by entering the following command in a **scoterm** window:

   xmodmap -e ***expression***

Step 1: Examining the current keymap table

To view the current keymap table, run the following command in a **scoterm** window:

xmodmap -pk

A list of keycodes and the keysyms that are associated with them appears. The following is a portion of a typical keymap listing generated by **xmodmap -pk**:

```
There are 2 KeySyms per KeyCode; KeyCodes range from 8 to 150.

KeyCode     Keysym (Keysym) ...
Value       Value   (Name)  ...

  8         0xff1b (Escape)
  9         0x0031 (1)      0x0021 (exclaim)
 10         0x0032 (2)      0x0040 (at)
 11         0x0033 (3)      0x0023 (numbersign)
 12         0x0034 (4)      0x0024 (dollar)
 13         0x0035 (5)      0x0025 (percent)
 14         0x0036 (6)      0x005e (asciicircum)
 15         0x0037 (7)      0x0026 (ampersand)
 16         0x0038 (8)      0x002a (asterisk)
 17         0x0039 (9)      0x0028 (parenleft)
 18         0x0030 (0)      0x0029 (parenright)
 19         0x002d (minus)  0x005f (underscore)
 20         0x003d (equal)  0x002b (plus)
 21         0xff08 (BackSpace)
 22         0xff09 (Tab)
 23         0x0051 (Q)
 24         0x0057 (W)
 25         0x0045 (E)
 26         0x0052 (R)
 27         0x0054 (T)
 28         0x0059 (Y)
  .
  .
  .
```

Make a note of the keycodes and keysyms that you want to remap.

In the above listing, the keysyms for "equal" and "plus" are associated with keycode 20. Their order in the table specifies that the "plus" keysym corresponds to the shifted key. If you swap their order in the table, however, the "equal" keysym corresponds to the shifted key.

Step 2: Specifying keymap table changes

To change the mapping of keysyms to keycodes, run the following command in a **scoterm** window:

xmodmap -e ***expression***

expression is a quoted string that tells **xmodmap** how to change the keymap table.

To redefine an entry in the keymap table, ***expression*** takes the following form:

keycode *keycode_number* = *keysym_list*

keycode_number specifies the keycode of the key you want to remap in the keymap table. ***keysym_list*** is a list of keysym names, delimited by spaces, that are mapped to ***keycode_number***. For a list of valid keysyms, examine */usr/lib/X11/xsconfig/keysymdef.h*. Note that you must not include the keysym's "XK_" prefix in ***keysym_list***.

NOTE You must leave spaces around the " = " character.

For example, to attach the bracketleft and braceleft keysyms to keycode 34 (to which bracketright and braceright are currently attached), run the following command in a **scoterm** window:

xmodmap -e "keycode 34 = bracketleft braceleft"

In this example, the keysyms bracketleft and braceleft are attached to keycode 34, and the keysyms bracketright and braceright are no longer attached.

To replace one keysym with another keysym or list of keysyms, ***expression*** takes the following form:

keysym *keysym_name* = *keysym_list*

keysym_name is the name of a keysym that is currently mapped to a keycode in the keymap table. ***keysym_list*** is a list of keysym names, delimited by spaces, that replace ***keysym_name*** in the keymap table. For a list of valid keysyms, examine */usr/lib/X11/xsconfig/keysymdef.h*. Note that you must not include the "XK_" prefix in ***keysym_list***.

This command is very useful if you want to replace keysym attachments but do not know the keycodes to which they are currently attached. Only the first occurrence of this ***keysym_name*** in the keymap table is replaced. If the keysym was mapped to multiple keycodes, you must use this command to find each entry. For example, to attach the keysym Control_L to the keycode to which the keysym CapsLock is attached, run the following command:

xmodmap -e "keysym CapsLock = Control_L"

Example of configuring the keyboard

This section provides a comprehensive example that ties together many of the concepts and procedures discussed in this chapter.

For the purposes of this example, let's assume you are accustomed to working from a terminal on which the left ⟨Ctrl⟩ key is directly above the left ⟨Shift⟩ key. However, you run your system from the console of a machine on which the ⟨CapsLock⟩ key is located where you expect the ⟨Ctrl⟩ key, and the ⟨Ctrl⟩ key is where you expect to find the ⟨CapsLock⟩ key. This example explains how you swap the function of these two keys.

1. Log in to the system. If logging in does not automatically start a Graphical Environment session, do so now by entering the following command at the prompt:

   startx &

2. Start a **scoterm** window and run the following command:

   xmodmap -pk | grep Caps_Lock

   This displays the keycode(s) to which the Caps_Lock keysym is attached. The output is similar to the following:

   ```
   65          0xffe5 (Caps_Lock)
   ```

3. To display the keycode(s) to which the Control_L keysym is attached, type the following command:

   xmodmap -pk | grep Control_L

   The output is similar to the following:

   ```
   36          0xffe3 (Control_L)
   ```

4. Attach the Control_L keysym to the keycode to which the Caps_Lock keysym is assigned with the following command:

   xmodmap -e "keysym Caps_Lock = Control_L"

5. Now check the keymap table with the following command:

   xmodmap -pk | grep Control_L

   The output indicates that the Control_L keysym is now attached to the keycode to which the Caps_Lock keysym was attached. The output is similar to the following:

   ```
   36          0xffe3 (Control_L)
   65          0xffe3 (Control_L)
   ```

   Note that you now need to attach the Caps_Lock keysym to the keycode to which the Control_L keysym was originally attached, which in this case is keycode 36.

6. Attach the Caps_Lock keysym to the keycode 36 with the following command:

   xmodmap -e "keycode 36 = Caps_Lock"

7. Verify that the Caps_Lock keysym has been mapped to the ⟨Ctrl⟩ key successfully with the following command:

   xmodmap -pk | grep Caps_Lock

   The output should show the Caps_Lock keysym attached to the keycode to which the Control_L keysym was originally attached. The output should be similar to the following:

   ```
   36          0xffe5 (Caps_Lock)
   ```

8. Verify that the Control_L keysym has been mapped to the ⟨CapsLock⟩ key successfully with the following command:

   xmodmap -pk | grep Control_L

   The output should show the Control_L keysym attached to the keycode to which the Caps_Lock keysym was originally attached. The output should be similar to the following:

   ```
   65          0xffe3 (Control_L)
   ```

9. Test the new keyboard configuration.

Chapter 12

Customizing the window manager

The SCO Panner window manager (**pmwm**), an enhanced version of the OSF/Motif window manager (**mwm**), provides a wide variety of methods for managing windows (for example, moving, resizing, iconifying, and so forth). Virtually all of the window manager features can be customized.

The default window manager operation is largely controlled by a system-wide configuration file, */usr/lib/X11/system.pmwmrc* (or, if you opt to run the window manager in **mwm** mode, */usr/lib/X11/system.mwmrc*). This file establishes the contents of the **Root** and **Window** menus, how menus and menu options are invoked, and what key and mouse button combinations can be used to manage windows.

Individual users can customize this behavior, using their personal window manager configuration file, **$HOME/***.pmwmrc* or **$HOME/***.mwmrc*.

Other window manager features that you can change, such as the appearance of window frames, icons, and menus, the keyboard focus policy, and how icons are arranged on the display, are controlled through resources.

This chapter discusses:

- changing the default **pmwm** mode of the SCO Panner window manager to the OSF/Motif **mwm** mode (page 220)
- creating a personal window manager configuration file (page 221)
- the syntax, structure, and content of the window manager configuration file (page 222)
- using window manager functions (page 223)

See also:

- Chapter 13, "Customizing window manager menus" (page 235)
- Chapter 14, "Configuring window manager button bindings" (page 253)
- Chapter 15, "Configuring window manager key bindings" (page 269)
- Appendix A, "OSF/Motif window manager resources" (page 377)
- Chapter 1, "Introduction to SCO Panner" in *Using SCO Panner*
- "Setting SCO Panner resources" in *Using SCO Panner*
- **mwm**(XC) manual page

Selecting between SCO Panner and OSF/Motif modes

When the SCO Panner window manager runs in the default **pmwm** mode, it provides a virtual workspace that is much larger than the size of the physical screen. A map of the entire workspace, divided into screen-size work areas, is displayed in a panner window. You can manipulate both the actual windows in the virtual workspace and the representations of the windows in the panner.

While the virtual workspace aspects of **pmwm** are very useful, you may also find that it places higher demands on your system, especially with regard to memory usage.

You have the option of running the SCO Panner window manager in **mwm** mode, which provides standard OSF/Motif window manager functionality.

To switch to **mwm** mode, you can change the **scosession *windowManager** resource in */usr/lib/X11/app-defaults/ScoSession* to:

***windowManager: /usr/bin/X11/mwm**

This tells the session manager to run the window manager in **mwm** mode (*/usr/bin/X11/mwm* is linked to */usr/bin/X11/pmwm*, specifying the **-mwm** option).

Alternatively, you can run:

/usr/bin/X11/pmwm -mwm

You could specify this command in a *.startxrc* file.

If you opt to use **mwm** mode, the window manager uses the */usr/lib/X11/system.mwmrc* configuration file and the */usr/lib/X11/app-defaults/Mwm* resource file to determine its appearance and behavior.

See also:

- "Creating a personal window manager configuration file" (this page)
- "Running the startx script" (page 46)
- Chapter 1, "Introduction to SCO Panner" in *Using SCO Panner* for information on how to use the SCO Panner window manager features

Creating a personal window manager configuration file

If you want to customize your window manager menus, key bindings and mouse button bindings, you can make your changes in either the */usr/lib/X11/system.pmwmrc* or the */usr/lib/X11/system.mwmrc* file. However, if you want to make customizations for individual users, you should add these changes to the user's personal *.pmwmrc* or *.mwmrc* file, located in the user's home directory, instead.

The *.pmwmrc* and *.mwmrc* files do not exist in **$HOME** by default. To create either of these files, copy the corresponding system file to *.pmwmrc* or *.mwmrc* in your home directory.

If you have created a personal window manager configuration file, you should note the following:

- The system configuration file is ignored by the system if you have your own *.pmwmrc* or *.mwmrc* file. Therefore, your personal file must include the complete sections for menu specifications and button and key binding specifications.
- If you change the default names of any menu, button, or key settings, you must modify the appropriate X resources that define the names.
- Any changes that you designate in your personal window manager configuration file only take effect after you restart the window manager. To do this, select the **Restart Window Manager** option from the **Root** menu.

The window manager configuration file is a standard text file containing items of information separated by blanks, tabs, and new-line characters. The following guidelines apply to either a personal or system-wide window manager configuration file:

- Blank lines are ignored.
- The "#" character at the beginning of a line is regarded as a comment. If the "!" character is the first character in a line, the line is also regarded as a comment.

- Items or characters that have special meaning are interpreted literally when quoted. For example, if you quote the comment character, it is not interpreted as the comment character.
- Items longer than one character are quoted with double quotes (").
- A single character is quoted by preceding it with a backslash (\).

Examining the window manager configuration file

The window manager configuration file is divided into the following sections:

- The Menu specifications section defines the contents of window manager menus.
- The Button bindings section binds mouse button events to window manager functions.
- The Key bindings section binds key events to window manager functions.

The following pseudo-code illustrates the syntax of the window manager configuration file:

```
Menu menu_name
{
    label   [mnemonic]   [accelerator]   function
    ...
}

Buttons bindings_set_name
{
    button  context          function
    ...
}

Keys bindings_set_name
{
    key     context          function
    ...
}
```

By default, the menu section defines the contents of the **Window** and **Root** menus. You can also specify submenus in this section, if desired. Menu items are paired with predefined window manager functions. (page 223)

A "binding" is a mapping between a user action (such as a keystroke) and a window manager function. The button bindings section specifies mouse buttons or key/button combinations that can be used to invoke various window manager functions. The key bindings section specifies keyboard keys that can be used to invoke the predefined functions.

See also:

- "Using window manager functions" (this page)
- Chapter 13, "Customizing window manager menus" (page 235)
- Chapter 14, "Configuring window manager button bindings" (page 253)
- Chapter 15, "Configuring window manager key bindings" (page 269)

Using window manager functions

Window manager functions are used to define the actions that occur when a menu item, or a mouse or key event is selected. For example, binding mouse button 1 to a client window with the **f.raise** function allows you to raise the window to the top of your screen whenever you press mouse button 1 within the boundary of that window.

The syntax for naming a function is the same for the three different sections in the *.mwmrc* file. The syntax is:

function = ***function_name*** [***function_args***]

where ***function_name*** is one of the defined window manager functions, and ***function_args*** is a valid argument to the function. If you do not specify a ***function_args*** for a function that accepts arguments, all of the possible arguments are applied to the function.

See also:

- "Function descriptions" (this page)
- "Function constraints" (page 230)

Function descriptions

The following list describes all of the window manager functions and includes the valid arguments for functions that accept arguments.

f.beep
This function causes a beep.

f.circle_down [icon | window]
This function places the window or icon that is on the top of the window stack to the bottom of the window stack (so that it no longer obscures any other window or icon). This function affects only those windows and icons that are obscuring other windows and icons, or that are obscured by other windows and icons. Secondary windows (that is, transient windows) are restacked with their associated "primary window." Secondary windows always

stay on top of the associated primary window, and there can be no other primary windows between the secondary windows and their primary window.

If the function argument is "icon," the function applies only to icons. If the function argument is "window," the function applies only to windows.

f.circle_up [icon | window]
This function raises the window or icon on the bottom of the window stack (so that it is not obscured by any other windows). This function affects only those windows and icons that are obscuring other windows and icons, or that are obscured by other windows and icons. Secondary windows (that is, transient windows) are restacked with their associated primary window.

If the function argument is "icon," the function applies only to icons. If the function argument is "window," the function applies only to windows.

f.exec or **!**
This function causes a command to be executed (using the value of the **$SHELL** environment variable if it is set, otherwise **/bin/sh** is used). This allows you to execute any shell command from a keystroke, button press, or menu item. The **!** notation can be used in place of the **f.exec** function name.

f.focus_color
This function sets the colormap focus to a client window. If this function is executed in the Root window context, then the default colormap (set up by the X server for the screen where the window manager is running) is installed, and there is no specific client window colormap focus. This function is treated as **f.nop** if the **colormapFocusPolicy** resource is not set to "explicit".

f.focus_key
This function sets the keyboard input focus to a client window or icon. This function is treated as **f.nop** if the **keyboardFocusPolicy** resource is not explicit or the function is executed in the Root window context.

f.hide_iconbox
This function unmaps the icon box, if mapped. (This function does not apply in **mwm** mode.)

f.hide_panner
This function unmaps the panner window, if mapped. (This function does not apply in **mwm** mode.)

f.identify
This function pops up a dialog box that provides useful information about the window from which the dialog box was initiated. If on the Root window, information about the operating system and the compile environment is displayed. (This function does not apply in **mwm** mode.)

f.kill
If the **WM_DELETE_WINDOW** protocol is set up, the client is sent a client message event indicating that the client window should be deleted. If the **WM_SAVE_YOURSELF** protocol is set up and the **WM_DELETE_WINDOW** protocol is not set up, the client is sent a client message event indicating that the client should prepare to be terminated. If the client does not have either the **WM_DELETE_WINDOW** or **WM_SAVE_YOURSELF** protocol set up, this function causes a client's X connection to be terminated (usually resulting in termination of the client). Refer to the description of the **quitTimeout** resource and the **WM_PROTOCOLS** property in the **mwm**(XC) manual page.

f.lower [*-client*]
This function lowers a client window to the bottom of the window stack (where it obscures no other window). Secondary windows (that is, transient windows) are restacked with their associated primary window.

The *-client* argument indicates the name or class of a client to lower. If the *-client* argument is not specified, the context in which the function was invoked indicates the window or icon to lower.

f.maximize
This function displays a client window at its full size. This is also know as maximizing a window.

f.menu *menu_name*
This function associates a submenu (or "cascading" menu) with a menu entry or associates a menu with a button or key binding.

The *menu_name* argument identifies the menu to be used; this argument is not optional.

f.minimize
This function minimizes (iconifies) a client window. A window is minimized when no icon box is used, and its icon is placed on the bottom of the window stack (such that it obscures no other window). If an icon box is used, then the client's icon changes to its iconified form inside the icon box and the normal window is removed from the screen. Secondary windows (that is, transient windows) are minimized with their associated primary window. There is only one icon for a primary window and all its secondary windows.

f.move
This function provides for interactive movement of a client window.

f.move_screen_to_client [*-client*]
This function moves the active screen to a work area displaying the named client or the activated icon. (This function does not apply in **mwm** mode.)

f.nail
This function provides the behavior for the **Toggle Nail** option on the **Window** menu. It acts as a toggle for either nailing or unnailing the current window. (This function does not apply in **mwm** mode.)

f.next_cmap
This function installs the next colormap in the list of colormaps for the window with the colormap focus. A client can install multiple colormaps. (See the books listed in "For further reading" (page 7) for more information.) The **f.next_cmap** function provides a mechanism that enables you to shuffle through the colormaps.

f.next_key [icon | window | transient]
This function sets the keyboard input focus to the next window or icon in the set of windows or icons managed by the window manager (the ordering of this set is based on the stacking of windows on the screen). This function is treated as **f.nop** if the **keyboardFocusPolicy** resource is not set to "explicit". The keyboard input focus is only moved to windows that do not have an associated secondary window that is application modal.

If the "transient" argument is specified, then transient (secondary) windows are traversed. Otherwise, if only "window" is specified, traversal is done only to the last focused window in a transient group.) If an "icon" function argument is specified, then the function applies only to icons. If a "window" function argument is specified, then the function applies only to windows.

f.nop
This function is a null function; no action is performed. When you want to include a command line that temporarily causes no action, you can use **f.nop** to satisfy the syntax requirement that a function of some type be named.

f.normalize
This function displays a client window in its normal size. Secondary windows (that is, transient windows) are placed in their normal state along with their associated primary window.

f.normalize_and_raise
This function displays a window in normal size and raises it to the top of the stack.

f.pack_icons
This function redraws icons on the Root window or in the icon box, based on the layout policy in use. In general, this causes icons to be "packed" into the icon grid.

f.pan_activescreen *±x±y-percent*
This function moves the active work area across the workspace in the direction specified by a percentage of the screen. As an example, *±x±y-percent* could be specified as "+50-50". This would move the active workarea half a screen (50 percent) to the right and half a screen (50 percent) up the workspace. The following values can also be used:

`+0+100`	up
`+0-100`	down
`-100+0`	left
`+100-0`	right

(This function does not apply in **mwm** mode.)

f.pass_keys
This function enables or disables (toggles) the processing of key bindings for window manager functions. When it disables key binding processing, all keys are passed on to the window with the keyboard input focus, and no window manager functions are invoked. If the **f.pass_keys** function is invoked with a key binding to disable key binding processing, the same key binding can be used to enable key binding processing.

f.post_wmenu
This function posts the **Window** menu that is defined by the **windowMenu** resource (see Chapter 13, "Customizing window manager menus" (page 235) for more details). If a key posts the **Window** menu and a window menu button is present, the Window menu is automatically placed with its top-left corner at the bottom-left corner of the window menu button for the client window. If no window menu button is present, the **Window** menu is placed at the top-left corner of the client window.

f.prev_cmap
This function installs the previous colormap in the list of colormaps for the window with the colormap focus. See **f.next_cmap** for more information.

f.prev_key [icon | window | transient]
This function sets the keyboard input focus to the previous window or icon in the set of windows or icons managed by the window manager (the ordering of this set is based on the stacking of windows on the screen). This function is treated as **f.nop** if the **keyboardFocusPolicy** resource is not explicit. The keyboard input focus is only moved to windows that do not have an associated secondary window that is application modal.

If the "transient" argument is specified, then transient (secondary) windows are traversed. Otherwise, if only "window" is specified, traversal is done only to the last focused window in a transient group. If an "icon" function argument is specified, the function applies only to icons. If a "window" function argument is specified, the function applies only to windows.

f.quit_mwm
This function terminates the window manager but not necessarily the X server.

f.raise [*-client*]
This function raises a client window to the top of the window stack (where it is obscured by no other window). Secondary windows (that is, transient windows) are restacked with their associated primary window.

The *-client* argument indicates the name or class of a client to raise. If the *-client* argument is not specified, the context in which the function was invoked indicates the window or icon to raise.

In **pmwm** mode only, this function also moves the current view area to the work area in which the client window is visible.

f.raise_lower
This function raises a client window to the top of the window stack if it is partially obscured by another window; otherwise, it lowers the window to the bottom of the window stack. Secondary windows (that is, transient windows) are restacked with their associated primary window.

f.refresh
This function redraws all windows on the display.

f.refresh_win
This function redraws a client window.

f.resize
This function allows interactive resizing of a client window.

f.restart
This function restarts the window manager, effectively terminating and re-executing **pmwm** or **mwm**.

f.send_msg *message_number*
This function sends a client message of the type **_MOTIF_WM_MESSAGES**, with the *message_number* function argument that indicates the type of message. The client message is sent only if *message_number* is included in the client's **_MOTIF_WM_MESSAGES** property. A menu item label is grayed if the menu item is used to do **f.send_msg** of a message that is not included in the client's **_MOTIF_WM_MESSAGES** property.

f.separator
This function causes a menu separator to be placed in a menu, at the specified location. You should use the "no-label" value for the ***label*** clause when you use this function.

f.set_activescreen $\pm x \pm y$ | **home**
This function sets the active screen to $\pm x \pm y$ geometry coordinates. "home" specifies the $\pm 0 \pm 0$ coordinates. Negative x and y coordinates are relative to the opposite side of the workspace. (This function does not apply in **mwm** mode.)

f.set_behavior
This function causes the window manager to restart with the default OSF behavior (if a custom behavior has been configured) or with a custom behavior (if an OSF default behavior has been configured). The default behavior is the internal appearance and behavior of the window manager, including the contents of the **Root** and **Window** menus. A custom behavior encompasses any resources that have been specified by a user, or user customizations to the window manager menus. You can toggle between these two behaviors by pressing ⟨Shift⟩⟨Ctrl⟩⟨Alt⟩**!**.

f.show_iconbox
This functions remaps the icon box. (This function does not apply in **mwm** mode.)

f.show_panner
This function remaps the panner window. (This function does not apply in **mwm** mode.)

f.snap
This function snaps the active screen to the closest specified grid position. (This function does not apply in **mwm** mode.)

f.sort_icons [icontitle | name | clienttitle | disable]
This function sorts and displays the icons on the sort criteria specifies. If no values are given, defaults to the value of the **iconSortOrder** resource. (This function does not apply in **mwm** mode.)

f.title
This function inserts a title in a menu, at the specified location.

f.toggle_autopan [on | off]
This function turns the autopan functionality on or off, temporarily.

Function constraints

Some functions cannot be used within some of the sections in the window manager configuration file. For example, you cannot use the **f.title** function to define a button or key binding; you can only use it in the menu specification section.

There are also constraints regarding the context in which a function can be used. For example, the **f.minimize** function only applies to windows ; it does not work when the pointer is on the **Root** menu or an icon.

You can configure a function's context as long as you stay within the limits of that function's constraints. For example, you can configure the **f.kill** function to work with icons, windows, or both. However, because the Root window is not an available context in which to use **f.kill**, you cannot configure it for use here.

Table 12-1, "Function contexts" describes the seven contexts in which functions can be used.

Table 12-1 Function contexts

Context	Description
app	The function can be performed when the pointer is on the application window, not including the window frame.
border	The function can be performed when the pointer is on the border of the window frame, not including the title bar.
frame	The function can be performed when the pointer is on the window frame around a client window, including the border and title bar.
icon	The function can be performed when the pointer is on an icon. Note that icon refers to window manager icons only, not Desktop icons.
root	The function can be performed when: 1. the pointer is on the **Root** menu, and 2. neither a client window nor an icon is to be acted upon by the function.
title	The function can be performed when the pointer is on the title area of the window frame.
window	The function can be performed when the pointer is on a client window, including the title bar and frame. Some functions, such as **f.maximize**, apply only when the window is normalized. Others, such as **f.normalize**, apply only when the window is maximized or minimized.

Table 12-2, "Where functions can be used" describes the contexts that are available to each of the window manager functions, as well as the sections of the window manager configuration file in which you can use the functions. Note that "button" represents the button binding section, "key" represents the key binding section, and "menu" represents the menu specification section.

Table 12-2 Where functions can be used

Function	Contexts	.mwmrc sections
f.beep	root,icon,window	button,key,menu
f.circle_down	root,icon,window	button,key,menu
f.circle_up	root,icon,window	button,key,menu
f.exec	root,icon,window	button,key,menu
f.focus_color	root,icon,window	button,key,menu
f.focus_key	root,icon,window	button,key,menu
f.hide_iconbox	root,icon,window	button,key,menu
f.hide_panner	root,icon,window	button,key,menu
f.identify	root,icon,window	button,key,menu
f.kill	icon,window	button,key,menu
f.lower	root,icon,window	button,key,menu
f.maximize	icon,window(normal)	button,key,menu
f.menu	root,icon,window	button,key,menu
f.minimize	window	button,key,menu
f.move	icon,window	button,key,menu
f.move_screen_to_client	icon	button,key,menu
f.nail	window	button,key,menu
f.next_cmap	root,icon,window	button,key,menu
f.next_key	root,icon,window	button,key,menu
f.nop	root,icon,window	button,key,menu
f.normalize	icon,window(maximized)	button,key,menu
f.normalize_and_raise	icon,window(maximized)	button,key,menu
f.pack_icons	root,icon,window	button,key,menu
f.pan_activescreen	root	button,key,menu
f.pass_keys	root,icon,window	button,key,menu
f.post_wmenu	root,icon,window	button,key
f.prev_cmap	root,icon,window	button,key,menu
f.prev_key	root,icon,window	button,key,menu
f.quit_mwm	root	button,key,menu
f.raise	root,icon,window	button,key,menu
f.raise_lower	icon,window	button,key,menu
f.refresh	root,icon,window	button,key,menu
f.refresh_win	window	button,key,menu

(Continued on next page)

Table 12-2 Where functions can be used
(Continued)

Function	Contexts	.mwmrc sections
f.resize	window	button,key,menu
f.restart	root	button,key,menu
f.send_msg	icon,window	button,key,menu
f.separator	root,icon,window	menu
f.set_activescreen	root	button,key,menu
f.set_behavior	root,icon,window	button,key,menu
f.show_iconbox	root,icon,window	button,key,menu
f.show_panner	root,icon,window	button,key,menu
f.snap	root,icon,window	button,key,menu
f.sort_icons	root,icon,window	button,key,menu
f.title	root,icon,window	menu
f.toggle_autopan	root,icon,window	button,key,menu

NOTE If a function is specified in a context that does not apply, or specified in an incompatible section of the window manager configuration file, the function is treated as **f.nop**.

Chapter 13

Customizing window manager menus

This chapter explains how you can create new window manager menus and customize existing window manager menus for your entire system or for a single user.

Specifically, this chapter describes:

- background about window manager menus (this page)
- adding or modifying window manager menus (page 237)
- changing the menu accessed by the window manager button (page 245)

There is also an example (page 249) at the end of this chapter that helps tie together many of the concepts and procedures discussed in this chapter.

See also:

- Chapter 24, "Configuring Desktop menus" (page 341)
- Chapter 12, "Customizing the window manager" (page 219)
- Chapter 5, "Understanding resources" (page 79)

About window manager menus

By default, the window manager provides two menus: the **Window** menu and the **Root** menu. The **Window** menu appears when you click on the window menu button, which is on the top left of a window frame. The **Root** menu appears when you press and hold mouse button 1 anywhere on the Desktop background or, if the Desktop is not running, in the Root window.

The functionality of these menus is defined in the menu section of the window manager configuration file (the */usr/lib/X11/system.pmwmrc* or */usr/lib/X11/system.mwmrc* system-wide files or the **$HOME/***.pmwmrc* or **$HOME/***.mwmrc* local files).

Depending on the window manager configuration file you use, you can modify the default menus or create new window manager menus that affect all users on your system or individual users only.

A window manager menu definition uses the following format:

```
Menu menu_name
{
    label   [mnemonic]   [accelerator]   function
    label   [mnemonic]   [accelerator]   function
    .
    .
    label   [mnemonic]   [accelerator]   function
}
```

The action of a window manager menu is defined through window manager functions. These functions define behavior, such as moving or iconifying a window, or displaying a submenu. Window manager functions are contextual; if an action is inappropriate for a circumstance, the menu item is dimmed.

Because window manager menus are not accessed from a menu bar, you also need to define the mouse button or key events that display the menus, and the context in which the menus are available. For example, by default, the **Root** menu is only available when you click and hold mouse button 1 in the Root window. If you click and hold mouse button 1 over an iconified window, the **Root** menu is not displayed.

See also:

- Chapter 12, "Customizing the window manager" (page 219) for more information on the window manager configuration files and the SCO Panner and OSF/Motif modes of operation

Adding or modifying window manager menus

You can create completely new window manager menus that you can call to the screen by pressing a mouse button or a key on the keyboard, or by selecting it from an existing menu. You can also modify the existing window manager menus (the **Root** and **Window** menus) to include additional menu items or submenus.

To create a new window manager menu, perform the following steps. For more information on each of these steps, see the sections immediately following this procedure.

1. Open the desired window manager configuration file for editing.
   - **pmwm** mode:

     /usr/lib/X11/system.pmwmrc for system-wide changes

     $HOME/*.pmwmrc* for individual changes

   - **mwm** mode:

     /usr/lib/X11/system.mwmrc for system-wide changes

     $HOME/*.mwmrc* for individual changes

2. Within the section of the configuration file that contains menu definitions, begin the new menu with the section type and section title, using the following syntax:

   Menu ***menu_name***

3. Enter your menu items, using the following syntax:

```
{
    label    [mnemonic]    [accelerator]    function
    label    [mnemonic]    [accelerator]    function
    .
    .
    label    [mnemonic]    [accelerator]    function
}
```

4. Define the method by which you want to access the new menu:
   - To access the menu as a submenu, add a new **menu_item** clause to the parent menu's definition section, referencing your new menu's *menu_name* with the **f.menu** function:

     label [*mnemonic*] [*accelerator*] **f.menu** "*menu_name*"

   - To use the menu through a mouse button or key sequence, define the mouse button or keystroke that you want to use in the key and button bindings sections of the configuration file:

     button_event context **f.menu** *menu_name*

     or

     key_event context **f.menu** *menu_name*

     See Chapter 14, "Configuring window manager button bindings" (page 253) and Chapter 15, "Configuring window manager key bindings" (page 269) for more information on button and key bindings.

   When you have finished, save and exit the configuration file.
5. Restart the window manager.

You can also add new menu items to the existing **Window** and **Root** menus by following these steps. In particular, pay attention to Step 1 (this page) and Step 3. (page 240) If you are modifying existing window manager menus. Step 4 (page 243) is unnecessary.

Step 1: Editing a window manager configuration file

The default operation of the window manager is largely controlled by a system-wide file, called *system.pmwmrc* if you are using **pmwm** mode or *system.mwmrc* if you are using **mwm** mode. This file's functions include defining the contents of the **Root** and **Window** menus and how menu functions are invoked. Individual users can also have a version of this file, located in their home directories, called *.pmwmrc* (for **pmwm** mode) or *.mwmrc* (for **mwm** mode). This file can be used to customize window manager menus for an individual user without affecting other users on the system.

Open one of the following files for editing:

- the system-wide configuration file (*/usr/lib/X11/system.pmwmrc* or */usr/lib/X11/system.mwmrc*) if you want to create a new menu or modify an existing menu so that all users are exposed to these customizations. You must have *root* permissions to edit this file.

- the local configuration file (**$HOME/***.pmwmrc* or **$HOME/***.mwmrc*) if you want to customize window manager menus for an individual user.

  The **$HOME/***.pmwmrc* and **$HOME/***.mwmrc* files do not exist by default. If a local configuration file does not already exist in your home directory, copy the appropriate system-wide window manager configuration file to **$HOME** and rename the file either *.pmwmrc* or *.mwmrc*.

NOTE Once the *.pmwmrc* or *.mwmrc* file exists in **$HOME**, it completely overrides the system-wide window manager configuration file. Therefore, make sure you copy the entire system file to your home directory, to avoid losing critical functionality.

See also:

- Chapter 12, "Customizing the window manager" (page 219) for more information on the system-wide and local window manager configuration files

Step 2: Starting a new menu

The first section of the window manager configuration file defines the contents of the **Root** and **Window** menus. If you want to create a new widow manager menu, add the menu definition to the end of this section.

A menu definition starts with the **Menu** section type and a title for the section, designated by the ***menu_name*** specification:

Menu *menu_name*

The ***menu_name*** specification is an internal reference and does not appear as the title of the menu. The string value that you assign to ***menu_name*** is often referenced elsewhere in the window manager configuration file. In particular, ***menu_name*** can be paired with a button action (in the button bindings section of the configuration file) so you can press a particular mouse button to display the menu. This concept is covered in greater detail in Step 4. (page 243)

NOTE The ***menu_name*** can also be used as the value for the **rootMenu** and the **windowMenu** resources. See "Changing the menu associated with the window menu button" (page 245) and Appendix A, "OSF/Motif window manager resources" (page 377) for more information on specifying these resources.

The following example shows the section type and title for the default **Root** menu:

```
Menu RootMenu
```

Step 3: Creating menu items

The syntax for defining items on a window manager menu is very simple. Each item is defined by a line that uses the following format:

```
{
  label  [mnemonic]  [accelerator]  function
}
```

The syntax for items on the **Root** menu is slightly different because mnemonics and accelerators are not available.

When creating menu options, note the following:

- The ***label*** specification is used as the text for the menu option, and has the following syntax:

  label = ***character_strings*** or ***bitmap_file***

  label can consist of a character string or a graphic representation (bitmap file). A character string must be compatible with the menu font that is used. Character strings must be typed precisely, using one of the following approaches:

  – strings containing a space must be enclosed in quotation marks "" "". For example: "Menu Name."

  – single-word strings do not have to be enclosed in quotation marks. However, it is preferable to do so for consistency. For example: "MenuName."

  If you are using a bitmap file instead of a character string, you need to tell the window manager the full path of the file. There are several methods for indicating a file's path:

Character	Function	Example
@	the following string is a pathname	@/u/tammyr/bitmaps/root_weave
~	the user's home directory	~/bitmaps/root_weave

NOTE You can also set the **bitmapDirectory** resource to the path of the directory containing the bitmaps. For example, if you set **bitmapDirectory** to **/u/tammyr/bitmaps**, the bitmap file could be specified as: **@root_weave**. For more information on setting resources, see Chapter 5, "Understanding resources" (page 79).

- The ***mnemonic*** specification indicates the character, or mnemonic, that can be used to select a menu item once the menu is open and displayed on the screen. A ***mnemonic*** specification has the following syntax:

  mnemonic = ***_character***

  The first character in the ***label*** specification that matches the designated ***mnemonic*** is underlined on the menu. Note that the ***mnemonic*** specification is case sensitive.

  If there is no matching character in ***label,*** no mnemonic is registered. The character must match a character in ***label*** exactly; the mnemonic cannot execute if any modifier (such as the ⟨Shift⟩ key) is pressed with the character.

  This specification is optional, and is not available on the **Root** menu.

- The ***accelerator*** specification is an accelerator key event, with the same syntax as the window manager key bindings. By default, an accelerator key sequence is a Meta key, usually the ⟨Alt⟩ key, plus a function key.

  The ***accelerator*** specification has the following syntax:

  accelerator = **Meta<key>*funct_key***

  The ***accelerator*** sequence works whether or not the menu is displayed. Because of this, you should be careful not to use key actions that are already defined in the key bindings section of the window manager configuration file. Key bindings are discussed in greater detail in Chapter 15, "Configuring window manager key bindings" (page 269).

  Because the key-action combination must be unique, accelerators use modifiers to provide more key-action combinations. Modifiers are keys that users must press simultaneously with the existing key event. All modifiers specified are exclusive; that is, only the specified modifiers can be present when the key event occurs.

The following list indicates the keys that can be used for modifiers:

modifier_name	Description
Ctrl	Control Key
Shift	Shift Key
Alt	Alt/Meta Key
Meta	Meta/Alt Key
Lock	Lock Key
Mod1	Modifier1
Mod2	Modifier2
Mod3	Modifier3
Mod4	Modifier4
Mod5	Modifier5

This ***accelerator*** specification is optional, and is not available on the **Root** menu.

- The ***function*** specification defines the action to be performed when a menu item is selected. The syntax for naming a function is:

  function = ***function_name*** [***function_args***]

  where ***function_name*** is one of the defined window manager functions, and ***function_args*** is a valid argument to the function.

  There are a number of predefined window manager functions that you can use for the ***function_name*** clause. Each of the window manager functions has a name beginning with "f.". These functions define behavior, such as resizing windows (**f.resize**), moving a window (**f.move**), and iconifying a window (**f.minimize**). For a complete list of the available window manager functions and a description of their behavior, see "Using window manager functions" (page 223).

  There are several window manager functions that are particularly relevant to the task of creating menus. These functions include:

Function	Behavior
f.menu	associates a submenu entry with a menu definition
f.title	inserts a title in the menu at the specified location
f.separator	inserts a dividing line in the menu at the specified location. An **f.separator** is automatically inserted after an **f.title**.

Several of the window manager functions can accept arguments. The nature of ***function_args*** depends on the specific function. In some cases, you can define the context in which the action should happen (for example, perform the action only on icons, or on windows and icons). Sometimes the argument is a specific client name, or a menu name, as with the **f.menu** function. Table 12-1, "Function contexts" (page 231) discusses the issue of context and arguments for window manager functions in detail.

If a ***function_args*** contains any spaces, the argument must be contained within double quotations "" "".

The following example shows how the ***label, mnemonic, accelerator,*** and ***function*** clauses are defined for all of the items on the **Window** menu that is used by default for **pmwm** mode:

```
"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
"Lower"         _L   Alt<key>F3          f.lower
"Raise"         _a   Alt Shift<key>F3    f.raise
"Toggle Nail"   _a   Alt Shift<key>F2    f.nail
"Hide"                                   f.hide_panner
```

NOTE Menu options are grayed if an entry performs the **f.nop** function, an invalid function, or a function that is not available in the current context.

Step 4: Specifying how to access the new menu

Before you can use your new menu, you must specify the method by which it can be accessed. You can attach the menu to an existing menu so it is available as a "cascading" or submenu. You can also configure the menu so a mouse button or key event displays the menu. These approaches are discussed here. You can also replace the default **Window** menu with your new menu, so it is automatically available by clicking on the window menu button. This approach is discussed in "Changing the menu associated with the window menu button" (page 245).

- To make your menu a submenu of an existing menu (perhaps the **Root** or **Window** menus), locate the intended parent menu's definition section in the local or system-wide window manager configuration file and add an **f.menu** function. This function must be coupled with the ***menu_name*** you used in your menu's definition section:

  label [***mnemonic***] [***accelerator***] **f.menu** "***menu_name***"

For example, if you created a menu that you defined as "GamesMenu", and you want to access it from the **Root** menu, you would enter the following line within the **Root** menu definition section:

```
"Games Menu"        f.menu "GamesMenu"
```

- To access your new menu directly through a mouse button or key sequence, you must add an **f.menu** function to the button or key binding section of the window manager configuration file. The **f.menu** function must be coupled with the *menu_name* you used in your menu's definition section. You also need to specify the context in which the menu is available.

  [*modifier_key*]*button_event context* **f.menu** "*menu_name*"

  or

  [*modifier_keys*]<*Key*>*key_name context* **f.menu** "*menu_name*"

  For example, if you created a menu that you defined as "GamesMenu", and you want to access it by pressing the second mouse button on the Root window, enter the following line within the button binding section of the window manager configuration file:

  ```
  <Btn2Down>    root    f.menu "GamesMenu"
  ```

  You can also use the **f.post_wmenu** function if you want to post the currently defined **Window** menu with a mouse button or key event. This function automatically displays the menu that is defined by the window manager **windowMenu** resource. If you intend to configure your new menu so it functions as the **Window** menu, use the **f.post_wmenu** function.

  For detailed information on the button and key binding sections, see Chapter 14, "Configuring window manager button bindings" (page 253), and Chapter 15, "Configuring window manager key bindings" (page 269).

When you have finished specifying how you can access your new menu, save and exit the window manager configuration file.

Step 5: Restarting the window manager

After you have added your new menu information to either the local window manager configuration file (**$HOME**/*.pmwmrc* or **$HOME**/*.mwmrc*) or the system-wide file (*/usr/lib/X11/system.pmwmrc* or *system.mwmrc*), you must restart the window manager before your changes can take effect and your new menu can be displayed.

Restart the window manager by selecting the **Restart Window Manager** option on the **Root** menu. The **Root** menu is accessed by pressing and holding mouse button 1 anywhere on the Desktop background or, if the Desktop is not running, in the Root window. After the window manager restarts, you can use your new menu.

Changing the menu associated with the window menu button

You can use the window manager **windowMenu** resource to change the menu that appears when the user clicks on the window menu button. This allows you to display a menu of your choice from the window menu button without having to extensively remodel the default **Window** menu.

To associate a new menu with the window menu button, create the new menu, as described in "Adding or modifying window manager menus" (page 237), and then perform the following steps. For more information on each of these steps, see the sections immediately following this procedure.

1. Open the desired resource file for editing.
   - **pmwm** mode:

     /usr/lib/X11/app-defaults/Pmwm for system-wide changes
   - **mwm** mode:

     /usr/lib/X11/app-defaults/Mwm for system-wide changes
   - both modes:

     $HOME/*.Xdefaults-**hostname*** for local changes
2. Add the **Window** menu resource specification, using the following format:

   Pmwm*windowMenu: ***menu_name***

   or

   Mwm*windowMenu: ***menu_name***
3. If desired, you can further customize the **Window** menu with the following resources:
   - Set the **wMenuButtonClick** resource so the **Window** menu stays posted on the screen or closes automatically when you finish clicking on the window menu button:

     Pmwm*wMenuButtonClick: ***boolean_value***

     or

     Mwm*wMenuButtonClick: ***boolean_value***

- Set the **wMenuButtonClick2** resource to determine if double-clicking on the window menu button closes the **Window** menu:

  Pmwm*wMenuButtonClick2: ***boolean_value***

  or

  Mwm*wMenuButtonClick2: ***boolean_value***

  When you have finished, save your changes and exit the resource file.

4. Restart the window manager.

Step 1: Editing the resource file

You can change the default **Window** menu so that all users on your system access the new menu, or you can change the **Window** menu for an individual user.

The majority of window manager resource settings are defined in the */usr/lib/X11/app-defaults/Pmwm* (for **pmwm** mode) or the */usr/lib/X11/app-defaults/Mwm* (for **mwm** mode) resource file. The resources in this file are read by the resource manager when the window manager is executed. If you want to define the new **windowMenu** resource for all users on your system, you should edit this file. You must have *root* privileges to perform this step.

Individual users can also change the **Window** menu that is displayed. Individual resource settings are placed in a file called *.Xdefaults-**hostname***, where ***hostname*** is the name of the host, or machine, where the window manager is running.

NOTE The *.Xdefaults-**hostname*** file does not exist in the user's home directory by default. If this file is not present, you must create it before you can specify a different **Window** menu.

If you create this file for a user from the *root* account, you must assign the file the correct ownership permissions. Run the **chown** command to assign the correct owner and the **chgrp** command to assign the correct group to the *.Xdefaults-**hostname*** file. If you created this file yourself, these steps are unnecessary.

When the user invokes a client, it checks to see if an *.Xdefaults-**hostname*** file exists in **$HOME**. If such a file does exist, the resource values specified in the user resource file take precedence over any values assigned to the same resource for the system, or in the resource database.

See also:

- "Methods for specifying resources" (page 87) for a more detailed explanation of resource files

Step 2: Setting the windowMenu resource

Use the window manager **windowMenu** resource to specify the menu that you would like to use instead of the default **Window** menu. Note that it is unlikely that this resource is actually defined in any of the window manager resource files on your system. The default value of the **windowMenu** resource is the name of the built-in **Window** menu specification, "DefaultWindowMenu".

Use the following format to set this resource:

Pmwm*windowMenu: ***menu_name***

or

Mwm*windowMenu: ***menu_name***

menu_name is the name you assigned your menu in the menu definition section of the window manager configuration file (*/usr/lib/X11/system.pmwmrc* and **$HOME**/*.pmwmrc* for **pmwm** mode or */usr/lib/X11/system.mwmrc* and **$HOME**/*.mwmrc* for **mwm** mode).

For example, if you created a menu that you defined as "MyMenu," and you want to use this menu in place of the default **Window** menu, set the **windowMenu** resource so it reads:

```
Pmwm*windowMenu: MyMenu
```

or

```
Mwm*windowMenu: MyMenu
```

Window menus can also be assigned on a client class basis. This means that the menu you specify is used as the **Window** menu for the designated client(s) only, and the default Window menu is used for all other clients. To do this, use the following format:

Pmwmclient*windowMenu:** ***menu_name***

or

Mwmclient*windowMenu:** ***menu_name***

client can be specified using either the application's binary or class name. ***menu_name*** is the name you assigned the menu in the menu definition section of the window manager configuration file.

For example, suppose you want to configure a particular menu of your own creation, defined as "EditorMenu" in the window manager configuration file. To have it function as the **Window** menu only in a **scoedit** graphical editor window, specify the following:

```
Pmwm*ScoEdit*windowMenu: EditorMenu
```

or

```
Mwm*ScoEdit*windowMenu: EditorMenu
```

Step 3: Modifying other Window menu resources

There are two other resources that you can set to customize the **Window** menu. These resources can be set on their own, or in conjunction with assigning a custom menu to the window menu button.

- The **wMenuButtonClick** resource controls whether clicking the mouse with the pointer over the window menu button posts (and leaves posted) the **Window** menu. If this resource has a value of "true," the menu remains displayed. "True" is the default value for this resource.
- The **wMenuButtonClick2** resource indicates whether double-clicking the mouse on the window menu button activates an **f.kill** function, thereby terminating the client. The default value, "true", causes an **f.kill** function.

Step 4: Restarting the window manager

After you have specified the new window manager resource values, you must restart the window manager so your changes can take effect. Restart the window manager by selecting the **Restart Window Manager** option from the **Root** menu. The **Root** menu is accessed by pressing and holding mouse button 1 on the Desktop background or, if the Desktop is not running, in the Root window.

Verify that your new menu is now being used as the **Window** menu.

Example of creating a window manager submenu

This section provides an example that ties together many of the concepts and procedures discussed in this chapter.

Let's assume that you use the SCO Panner window manager in the default **pmwm** mode. There are several clients that you run frequently and you would like to be able to access these commands from the Root window without having to first run a **scoterm** window or the Desktop. You can create a menu from which you can launch these commands, and you can make this menu accessible from the **Root** menu.

The following steps result in a **Client** submenu that is available from the **Root** menu:

1. Open the local **pmwm** configuration file, *.pmwmrc*, for editing. This file is located in your home directory.

   If this file does not currently exist, copy the entire system-wide **pmwm** configuration file, */usr/lib/X11/system.pmwmrc*, to your home directory and rename it *.pmwmrc*.

2. At the end of the menu section, open a line and begin your new menu definition with the following:

   ```
   Menu ClientMenu
   ```

3. Assign your menu a title by entering the following lines:

   ```
   {
       "Client Menu"      f.title
   ```

4. Create your menu items by entering the following lines:

```
    "xclock"        f.exec "xclock"
    "xload"         f.exec "xload"
    "xcalc"         f.exec "xcalc"
    "xbiff"         f.exec "xbiff"
    "xmag"          f.exec "xmag"
    "xeyes"         f.exec "xeyes"
    no-label        f.separator
    "scoterm"       f.exec "scoterm"
    "xterm"         f.exec "xterm"
}
```

When you have finished entering the menu items, the entire menu definition should look like this:

```
Menu ClientMenu
{
    "Client Menu"   f.title
    "xclock"        f.exec "xclock"
    "xload"         f.exec "xload"
    "xcalc"         f.exec "xcalc"
    "xbiff"         f.exec "xbiff"
    "xmag"          f.exec "xmag"
    "xeyes"         f.exec "xeyes"
    no-label        f.separator
    "scoterm"       f.exec "scoterm"
    "xterm"         f.exec "xterm"
}
```

5. Now you need to add a menu item for your **Client** menu to the **Root** menu definition. Let's add the submenu so it appears after the **Refresh** option and before the **Restart Window Manager** option. And let's separate the submenu option from the other options on the **Root** menu with a dividing line.

   Within the definition section of the **Root** menu, open a line below the **Refresh** option and enter the following:

```
    no-label        f.separator
    "Client Menu"   f.menu ClientMenu
```

When you have finished, the **Root** menu definition section should look like this:

```
Menu RootMenu
{
  @sco-logosm.xbm          f.title
  "Unix Window"            f.exec "/usr/bin/X11/scoterm"
  "Desktop"                f.exec "/usr/bin/X11/xdt3"
  "Mail"                   f.exec "/usr/bin/X11/scomail"
  "Calendar"               f.exec "/usr/bin/X11/scocal"
  "Screen Lock"            f.exec "/usr/bin/X11/scolock"
  no-label                 f.separator
  "Help on SCO Panner"     f.exec "/usr/bin/X11/scohelp ..."
  "Help"                   f.exec "/usr/bin/X11/scohelp"
  no-label                 f.separator
  "Panner/Icon Menu"       f.menu "PanMenu"
  "Refresh"                f.refresh
  no-label                 f.separator
  "Client Menu"            f.menu ClientMenu
  no-label                 f.separator
  "Restart Window Manager" f.restart
  "Quit Window Manager"    f.quit_mwm
  no-label                 f.separator
  "Exit Session"           f.exec "/usr/bin/X11/scosession -stop"
}
```

6. When you have finished writing the definition for your new menu and adding the submenu to the **Root** menu, save and exit the *.pmwmrc* file. Make sure you leave the remaining sections of the *.pmwmrc* file intact.

7. Access the **Root** menu and select the **Restart Window Manager** option. The **Root** menu is available by pressing and holding mouse button 1 on the Desktop background or, if the Desktop is not running, in the Root window.

   The window manager is restarted; the new menu information in your *.pmwmrc* file is read, and your new **Client** submenu is now available from the **Root** menu.

Chapter 14

Configuring window manager button bindings

"Button bindings" provide a way to define the action that is performed when you press a mouse button in various window manager contexts. You can use different mouse buttons, alone or in combination with keystrokes, to invoke many different actions. The actions performed depend on where the pointer (mouse cursor) is focused on the screen.

This chapter describes:

- background information about window manager button bindings (page 254)
- configuring new button bindings (page 257)
- creating new button binding sets (page 263)

There is also an example (page 267) at the end of this chapter that helps to tie together many of the concepts and procedures discussed in this chapter.

See also:

- "Using window manager functions" (page 223)
- Chapter 5, "Understanding resources" (page 79)

Default button bindings

The button bindings that are provided with your system are referred to as default button bindings. Most default button bindings can be reconfigured.

All bindings are located in the */usr/lib/X11/system.pmwmrc* file (for **pmwm** mode) or the */usr/lib/X11/system.mwmrc* file (for **mwm** mode) in the **Buttons** section type. The default bindings are defined by a set named "DefaultButtonBindings." System administrators can customize the functionality of some button bindings and make system-wide changes by editing this file. Users can customize the functionality of button bindings in their local environment by copying the appropriate system-wide window manager configuration file to either *.pmwmrc* (for **pmwm** mode) or *.mwmrc* (for **mwm** mode) in their **$HOME** directory.

The entries in the "DefaultButtonBindings" section of the system-wide window manager configuration file look similar to these:

```
Buttons  DefaultButtonBindings
{
        .          .          .
   <Btn1Down>   frame|icon   f.raise
   <Btn2Down>   frame|icon   f.post_wmenu
        .          .          .
        .          .          .

}
```

Each line in the "DefaultButtonBindings" section represents a button binding. A button binding consists of a button action (such as **Btn1Down**), the window manager context in which the action is valid (such as **frame**), and the function the action provides (such as **f.raise**). Button actions, contexts, and functions are discussed in more detail later in this chapter.

Table 14-1, "Default button bindings" lists the default button bindings.[1]

Table 14-1 Default button bindings

Button action	Context	Function
For Pmwm mode:		
Btn1Down	frame | icon	f.raise
Btn2Down	frame | icon	f.post_wmenu
Alt Btn1Down	title	f.move
Alt Btn1Down	frame	f.resize
Btn1Down	root	f.menu "RootMenu"
Btn2Down	root	f.menu "RootMenu"
Btn3Down	root	f.menu "PanMenu"
# post menus		
Shift Btn1Click	root	f.menu "RootMenu"
Shift Btn2Click	root	f.menu "RootMenu"
Shift Btn3Click	root	f.menu "PanMenu"
# menus that pop-up	under xdt	
Ctrl Btn1Down	root	f.menu "RootMenu"
Ctrl Btn2Down	root	f.menu "RootMenu"
Ctrl Btn3Down	root	f.menu "PanMenu"
Shift Alt Btn1Down	window | icon	f.raise
Alt Btn1Down	icon | window	f.lower
Alt Btn2Down	window | icon	f.resize
Alt Btn3Down	window	f.move
For Mwm mode:		
Btn1Down	frame | icon	f.raise
Btn2Down	frame | icon	f.post_wmenu
Alt Btn1Down	title	f.move
Alt Btn1Down	frame	f.resize
Btn1Down	root	f.menu "RootMenu"
Btn2Down	root	f.menu "RootMenu"
# post menus		
Shift Btn1Click	root	f.menu "RootMenu"
Shift Btn2Click	root	f.menu "RootMenu"
# menus that pop-up	under xdt	
Ctrl Btn1Down	root	f.menu "RootMenu"
Ctrl Btn2Down	root	f.menu "RootMenu"
Shift Alt Btn1Down	window | icon	f.raise
Alt Btn1Down	icon | window	f.lower
Alt Btn2Down	window | icon	f.resize
Alt Btn3Down	window	f.move

1. A context of *icon* indicates window manager icons, not icons on the Desktop.

You can reconfigure the default button bindings or create new button bindings and add them to the default binding set. In addition, if you want to underscore to yourself that you are using your own bindings rather than the default bindings, you can create your own sets of button bindings.

See also:

- "About window manager functions" (this page)
- "Configuring button bindings" (page 257)
- "Creating a new button binding set" (page 263)

About window manager functions

Window manager functions are a component of every section of the system-wide window manager configuration file (*system.pmwmrc* or *system.mwmrc*) and the local configuration file (*.pmwmrc* or *.mwmrc*). The system-wide and local configuration files use window manager functions to define the behavior of mouse buttons, keys, and menu panes.

You configure button bindings by associating each mouse button with at least one window manager function. For example, the **f.move** function is defined to allow a client window to be interactively moved. If you bind mouse button 3 to a client window with the **f.move** function, the window moves with the mouse each time you press and hold down mouse button 3. This button binding definition would look like this:

```
Buttons  bindings_set_name
{
        .          .          .
    <Btn3Down>  window     f.move
        .          .          .
        .          .          .
}
```

See also:

- "Using window manager functions" (page 223) for a complete list and a detailed explanation of all of the valid window manager functions

Configuring button bindings

To modify an existing button binding or to create a new button binding, perform the following steps. For more information on each of these steps, see the sections immediately following this procedure.

1. Open the desired window manager configuration file for editing.
   - **pmwm** mode:

     /usr/lib/X11/system.pmwmrc for system-wide changes

     $HOME/*.pmwmrc* for individual changes

   - **mwm** mode:

     /usr/lib/X11/system.mwmrc for system-wide changes

     $HOME/*.mwmrc* for individual changes

2. Locate the button binding section in the window manager configuration file. The button binding section uses the following syntax:

```
Buttons  bindings_set_name
{
    button context   function
    button context   function
    button context   function
                .
                .
    button context   function
}
```

3. Configure the button binding specification, if desired.
4. Configure the function specification, if desired.
5. Configure the context specification, if desired.
6. Restart the window manager and test your new button binding.

See also:

- "Example of creating a new button set" (page 267) for sample button binding definitions

Step 1: Editing a window manager configuration file

If you want to make system-wide changes to the default button bindings, open the system-wide window manager configuration file, */usr/lib/X11/system.pmwmrc* if you are using **pmwm** mode or *system.mwmrc* if you are using **mwm** mode. The system-wide file contains the default functionality for the window manager button bindings.

If you want to customize the button bindings in your local Graphical Environment, edit your personal window manager configuration file, **$HOME/***.pmwmrc* if you are using **pmwm** mode or **$HOME/***.mwmrc* if you are using **mwm** mode. The personal configuration file is not provided with the system by default. If it does not already exist, you must create your own by copying the appropriate system-wide window manager configuration file to *.pmwmrc* or *.mwmrc* in your **$HOME** directory.

NOTE Once the *.pmwmrc* or *.mwmrc* file exists in **$HOME**, it completely overrides the system-wide window manager configuration file. Therefore, make sure you copy the entire system file to your home directory, to avoid losing critical functionality.

See also:

- Chapter 12, "Customizing the window manager" (page 219) for more information on the system-wide and local window manager configuration files

Step 2: Locating the button binding section

When you locate the appropriate button binding section, you see entries with the following syntax:

```
Buttons  bindings_set_name
{
    button context  function
    button context  function
                .
                .
}
```

Buttons — the type of the binding set being defined.

bindings_set_name — the name assigned to a set of bindings. The default name is "DefaultButtonBindings".

button — the specification that defines the mouse button action or the mouse button action plus a key sequence.

context — the specification that defines the window manager context in which the button specification becomes active. For example, the context of a **window** indicates that the pointer must be on a client window or a window management frame for the button specification to be effective.

function — the specification that defines one of the many window manager functions or actions. See "Using window manager functions" (page 223) for a list of these functions.

Step 3: Configuring the button binding specification

The window manager configuration files use window manager functions to define the behavior and control the functionality of button events. A button event describes an action that you take, such as pressing mouse button 1, to execute a function (for example: raising a window). The button event specification has the following syntax:

button = [***modifier_list***]***button_event_name***

where:

modifier_list = ***modifier_name*** {***modifier_name***}

Table 14-2, "Button event definitions" lists the values that can be used for ***button_event_name***. Table 14-3, "Modifiers" (page 260) lists the values that can be used for ***modifier_name***.

Table 14-2 Button event definitions

button_event_name	Description
Btn1Down	Button 1 press
Btn1Up	Button 1 release
Btn1Click	Button 1 press and release
Btn1Click2	Button 1 double click
Btn2Down	Button 2 press
Btn2Up	Button 2 release
Btn2Click	Button 2 press and release
Btn2Click2	Button 2 double click
Btn3Down	Button 3 press
Btn3Up	Button 3 release
Btn3Click	Button 3 press and release
Btn3Click2	Button 3 double click
Btn4Down	Button 4 press
Btn4Up	Button 4 release
Btn4Click	Button 4 press and release
Btn4Click2	Button 4 double click
Btn5Down	Button 5 press
Btn5Up	Button 5 release
Btn5Click	Button 5 press and release
Btn5Click2	Button 5 double click

To make sure you are not selecting already existing button-action combinations, you might want to modify the button event by requiring a simultaneous key press with the button action. These required key presses are called "modifiers". All modifiers specified are exclusive; that is, only the specified modifiers can be present when the button event occurs. Table 14-3, "Modifiers" (this page) indicates the values that can be used for ***modifier_name***.

NOTE The ⟨Alt⟩ key is frequently labeled **Extend** or **Meta**. **Alt** and **Meta** can be used interchangeably for an event specification.

Table 14-3 Modifiers

modifier_name	Description
Ctrl	Control Key
Shift	Shift Key
Alt	Alt/Meta Key
Meta	Meta/Alt Key
Lock	Lock Key
Mod1	Modifier1
Mod2	Modifier2
Mod3	Modifier3
Mod4	Modifier4
Mod5	Modifier5

For example, in the following sample button binding definition, the button specification is Meta⟨Btn3Down⟩. When the pointer is in a client window and you press the Meta key simultaneously with mouse button 3, you execute the **f.move** function.

```
Buttons  bindings_set_name
{
    Meta<Btn3Down>  window      f.move
          .
          .
}
```

If you are editing an existing button binding, replace the old button specification with the new one. Make sure you do not try to reconfigure the default button bindings listed in Table 14-1, "Default button bindings" (page 255).

If you are adding a new button binding, put the new button specification on a new line.

Step 4: Configuring the function specification

The syntax for naming a function is the same, no matter what type the function describes. The syntax for naming a function is:

function = *function_name* [*function_args*]

function_name is one of the valid window manager functions, and ***function_args*** is a valid argument to the function. If ***function_args*** contains more than one word, the argument must be contained in quotes.

If you are editing an existing button binding, replace the old function specification with the new one.

If you are adding a new button binding, put the new function specification on a new line, after the related button specification.

See also:

- "Using window manager functions" (page 223) for a complete list and a detailed explanation of all the valid window manager functions

Step 5: Configuring the context specification

The syntax for the context specification is:

context = **app | border | frame | icon | root | title | window**

The ***context*** specification defines the window manager context in which the button specification becomes active; it indicates where the pointer must be for the button specification to be effective. For example, the context of a **window** indicates that the pointer must be on a client window or a window management frame for the button binding to be effective. The button specification can be active in more than one context.

Table 14-4, "Button binding contexts" lists and describes the values that can be used for *context*.

Table 14-4 Button binding contexts

Context	Description
app	The button binding is effective when the pointer is on the application window, not including the window management frame.
border	The button binding is effective when the pointer is on the border of the window management frame, not including the title bar.
frame	The button binding is effective when the pointer is on the window management frame around a client window, border, and title bar.
icon	The button binding is effective when the pointer is on an icon. Note that icon refers to window manager icons only, not Desktop icons.
root	The button binding is effective when: 1. the pointer is on the Root window, and 2. neither a client window nor an icon is to be acted upon by the function.
title	The button binding is effective when the pointer is on the title area of the window management frame.
window	The button binding is effective when the pointer is on a client window, title bar, or a window management frame.

If you are editing an existing button binding, replace the old context specification with the new one.

If you are adding a new button binding, put the context specification on a new line, after the related button and function specifications. See "Function constraints" (page 230) and Table 12-2, "Where functions can be used" (page 232) for a list of the contexts that are available to each of the window manager functions.

Step 6: Restarting the window manager

After you configure your new button bindings, you must restart the window manager before your changes can take effect. Restart the window manager by selecting the **Restart Window Manager** option from the **Root** menu. The **Root** menu is accessed by pressing and holding mouse button 1 on the Desktop background or, if the Desktop is not running, in the Root window.

Verify that the new button bindings are configured correctly by testing your new button sequences in the appropriate contexts. The button bindings are effective immediately after the window manager is restarted.

Creating a new button binding set

To create a new button binding set, perform the following steps. For more information on each of these steps, see the sections immediately following this procedure.

1. Open the desired window manager configuration file for editing.
   - **pmwm** mode:

     /usr/lib/X11/system.pmwmrc for system-wide changes

     $HOME/*.pmwmrc* for individual changes

   - **mwm** mode:

     /usr/lib/X11/system.mwmrc for system-wide changes

     $HOME/*.mwmrc* for individual changes

2. Locate the "DefaultButtonBindings" section in the window manager configuration file, copy it, then use it as a template for your new set.
3. Configure the new button, function, and context specifications.
4. Configure the new button binding set for use by the window manager with the **buttonBindings** resource. Add this resource to */usr/lib/X11/app-defaults/Pmwm* (for **pmwm** mode), */usr/lib/X11/app-defaults/Mwm* (for **mwm** mode), or to the **$HOME/***.Xdefaults-**hostname*** file, where ***hostname*** is the name of the machine on which the window manager is running. Use the following syntax:

   Pmwm*buttonBindings: ***bindings_set_name***

   or

   Mwm*buttonBindings: ***bindings_set_name***

5. Restart the window manager and test your new button binding set.

See also:

- "Example of creating a new button set" (page 267) to see sample button binding definitions

Step 1: Editing a window manager configuration file

If you want to make system-wide changes to the default button bindings, open the system-wide window manager configuration file, */usr/lib/X11/system.pmwmrc* if you are using **pmwm** mode or *system.mwmrc* if you are using **mwm** mode. The system-wide file contains the default functionality for the window manager button bindings.

If you want to customize the button bindings in your local Graphical Environment, edit your personal window manager configuration file, **$HOME/***.pmwmrc* if you are using **pmwm** mode or **$HOME/***.mwmrc* if you are using **mwm** mode. The personal configuration file is not provided with the system by default. If it does not already exist, you must create your own by copying the appropriate system-wide window manager configuration file to *.pmwmrc* or *.mwmrc* in your **$HOME** directory.

NOTE Once the *.pmwmrc* or *.mwmrc* file exists in **$HOME**, it completely overrides the system-wide window manager configuration file. Therefore, make sure you copy the entire system file to your home directory, to avoid losing critical functionality.

See also:

- Chapter 12, "Customizing the window manager" (page 219) for more information on the system-wide and local window manager configuration files

Step 2: Locating the DefaultButtonBindings section

When you locate the "DefaultButtonBindings" section, you see entries with the following syntax:

```
Buttons  DefaultButtonBindings
{
    button context  function
    button context  function
                .
                .
}
```

Buttons	the type of the binding set being defined
"DefaultButtonBindings"	the default name assigned to a set of bindings
button	the specification that defines the mouse button action or the mouse button action plus a key sequence. See Table 14-2, "Button event definitions" (page 259) for a list of the values that can be used for ***button***. See Table 14-3, "Modifiers" (page 260) for a list of modifiers that can be used with the button actions.
context	the specification that defines the context in which the button specification becomes active. See Table 14-4, "Button binding contexts" (page 262) for a list and description of the values that can be used for ***context***.

function — the specification that defines one of the many window manager functions or actions. See "About window manager functions" (page 256) for a basic description of functions. See "Using window manager functions" (page 223) for a list and a detailed description of functions.

Create a template for your new binding set by copying the "DefaultButtonBindings" section, placing the copy below the existing section. Assign the new set a different name.

Step 3: Defining button, function, and context specifications

Each button binding definition consists of button event, function, and context specifications. Use the template you created in Step 2 (page 264) to create these new specifications.

- Define the new button event specification. The button event specification has the following syntax:

  button = [***modifier_list***]***button_event_name***

  where:

  modifier_list = ***modifier_name*** {***modifier_name***}

  See Table 14-2, "Button event definitions" (page 259) and Table 14-3, "Modifiers" (page 260) for lists of the values that can be used for ***button_event_name*** and ***modifier_name***.

- Define the function specification, after the button event specification. The syntax for naming a function is:

  function = ***function_name*** [***function_args***]

  where ***function_name*** is one of the valid window manager functions, and ***function_args*** is a valid argument to the function. If ***function_args*** contains more than one word, the argument must be contained in quotes.

  See "Using window manager functions" (page 223) for a complete list and a detailed explanation of all the default window manager functions.

- Define the context specification after the function specification. The syntax for the context specification is:

  context = **app** | **border** | **frame** | **icon** | **root** | **title** | **window**

  The context specification defines the context in which the button specification becomes active; it indicates where the pointer must be for the button specification to be effective. The button specification can be active in more than one context. See Table 14-4, "Button binding contexts" (page 262) for a list and description of the values that can be used for ***context***.

Step 4: Specifying the buttonBindings resource

Define the new button binding set through the **buttonBindings** resource, using the appropriate resource file.

If you are making local changes, you must define the new binding set in the **$HOME**/*.Xdefaults-**hostname*** file. (If this file does yet not exist, create a file in your **$HOME** directory named *.Xdefaults-**hostname***, where ***hostname*** is the name of the host, or machine, where the window manager is running.)

If you want the new button bindings to be used by all users on your system, specify the resource in */usr/lib/X11/app-default/Pmwm* (if you are using **pmwm** mode) or */usr/lib/X11/app-default/Mwm* (if you are using **mwm** mode).

The syntax of the resource specification is the same for all of the window manager configuration files:

Pmwm*buttonBindings: ***new_button_bindings_set_name***

or

Mwm*buttonBindings: ***new_button_bindings_set_name***

See also:

- Chapter 5, "Understanding resources" (page 79) for more information on resources and the *.Xdefaults-**hostname*** file

Step 5: Restarting the window manager

After you create your new button binding set, you must restart the window manager so your changes can take effect. Restart the window manager by selecting the **Restart Window Manager** option from the **Root** menu. The **Root** menu is accessed by pressing and holding mouse button 1 on the Desktop background or, if the Desktop is not running, in the Root window.

Verify that the new button binding set was created correctly by testing your new button sequences in the appropriate contexts. The button bindings are effective immediately after the window manager is restarted.

Example of creating a new button set

This section provides a comprehensive example that ties together some of the concepts and procedures discussed in this chapter.

The following example assumes that you are using the default **pmwm** mode of the window manager, that you want to create a new button binding set for your local environment, and you want to name the set "MyButtonBindings". MyButtonBindings consists of five new button binding definitions that produce the following results:

- Pressing mouse button 1 when the pointer is on the **Root** menu posts the menu named "MyRootMenu". (See line six of the following sample file.)
- Pressing mouse button 1 when the pointer is on a window frame raises the window. (See line seven of the following sample file.)
- Pressing mouse button 2 or 3 when the pointer is on either the window frame or on a window manager icon posts the menu specified by the **Pmmwm*windowMenu** resource. (See lines eight and nine of the following sample file.)
- Pressing the Meta key and mouse button 1 simultaneously, with the pointer on either an icon or a window, raises the icon or the window to the top of the window stack if it is partially obscured by another icon or window; otherwise, it lowers the icon or window to the bottom of the window stack. (See line ten of the following sample file.)
- Pressing the Meta key and mouse button 2 simultaneously, with the pointer on a window, sets the colormap focus to the window. (See line eleven of the following sample file.)

To create this new set:

1. Open the *.pmwmrc* file in your **$HOME** directory for editing. (If *.pmwmrc* does not already exist, create it by copying */usr/lib/X11/system.pmwmrc* to *.mwmrc* in your **$HOME** directory).
2. Locate the "DefaultButtonBindings" section in the *.pmwmrc* file and place a copy of the section underneath the default definitions. You can use this copy as a template for your new button binding set.
3. In the copy of the default definitions, rename "DefaultButtonBindings" to "MyButtonBindings."

4. Edit the template so that it looks like this:

```
1   #
2   # button binding descriptions
3   #
4   Buttons MyButtonBindings
5   {
6       <Btn1Down>        root          f.menu MyRootMenu
7       <Btn1Down>        frame         f.raise
8       <Btn2Down>        frame|icon    f.post_wmenu
9       <Btn3Down>        frame|icon    f.post_wmenu
10      Meta<Btn1Down>    icon|window   f.raise_lower
11      Meta<Btn2Down>    window        f.focus_color
}
```

   Note that for the **f.focus_color** function to have an effect, the **pmwm colormapFocusPolicy** resource must be set to "explicit". See Chapter 12, "Customizing the window manager" (page 219) and Appendix A, "OSF/Motif window manager resources" (page 377) for more information.

5. Define the **buttonBindings** resource to announce the new button binding set to the window manager. Specify the resource in the **$HOME**/.*Xdefaults-**hostname*** file, where ***hostname*** is the name of the host, or machine, where the window manager is running. The resource specification should look like this:

   Pmwm*buttonBindings: MyButtonBindings

6. Restart the window manager so that the new button binding information is implemented.

Chapter 15
Configuring window manager key bindings

Key bindings provide a way to define the action that is performed when you press a key or a sequence of keys in various window manager contexts. You can use many combinations of keystrokes to invoke many different actions. The actions performed depend on where the keyboard input focus is on the screen.

This chapter describes:

- background information about window manager accelerator and mnemonic key bindings (page 270)
- configuring new key bindings (page 273)
- creating new key binding sets (page 278)

There is also an example (page 282) at the end of this chapter that helps tie together many of the concepts and procedures discussed in this chapter.

This chapter does not discuss translation tables and hardcoding key bindings into applications. For more information about translation tables refer to O'Reilly and Associates' *X Toolkit Intrinsics Programming Manual Volume Four.*

See also:

- Chapter 12, "Customizing the window manager" (page 219)
- Chapter 5, "Understanding resources" (page 79)

Default key bindings

The key bindings that are provided with your system are referred to as default key bindings. Most default key bindings define the basic functionality of the window manager.

All key bindings are located in the */usr/lib/X11/system.pmwmrc* file (for **pmwm** mode) or the */usr/lib/X11/system.mwmrc* file (for **mwm** mode) in the **Keys** section type. The default bindings are defined by a set named "DefaultKeyBindings." System administrators can customize the functionality of some button bindings and make system-wide changes by editing this file. Users can customize the functionality of key bindings in their local environment by copying the appropriate system-wide window manager configuration file to either *.pmwmrc* (for **pmwm** mode) or *.mwmrc* (for **mwm** mode) in their **$HOME** directory. Customizing default bindings is discussed in more detail in "Configuring key bindings" (page 273).

The entries in the "DefaultKeyBindings" section of the system-wide window manager configuration file look similar to these:

```
Keys  DefaultKeyBindings
{
            .                  .                  .
    Shift<Key>Escape  icon|window          f.post_wmenu
    Alt<Key>space     icon|window          f.post_wmenu
            .                  .                  .
            .                  .                  .

}
```

Each line represents a key binding. A key binding consists of a key press action (such as Shift⟨Key⟩Escape), the window manager context in which the action is valid (such as **icon** or **window**), and the function the key press provides (such as **f.post_wmenu**). Key press actions, contexts, and functions are discussed in more detail later in this chapter.

Table 15-1, "Default key bindings" lists the default key bindings.[1]

Table 15-1 Default key bindings

Key action	Context	Function
Shift<Key>Escape	icon | window	f.post_wmenu
Alt<Key>space	icon | window	f.post_wmenu
Alt<Key>Tab	root | icon | window	f.next_key
Alt Shift<Key>Tab	root | icon | window	f.prev_key
Alt<Key>Escape	root | icon | window	f.next_key
Alt Shift<Key>Escape	root | icon | window	f.prev_key
Alt Ctrl Shift<Key>exclam	root | icon | window	f.set_behavior
Alt<Key>F6	window	f.next_key transient
Alt<Key>Left	root | icon | window	f.pan_activescreen left 100
Alt<Key>Right	root | icon | window	f.pan_activescreen right 100
Alt<Key>Up	root | icon | window	f.pan_activescreen up 100
Alt<Key>Down	root | icon | window	f.pan_activescreen down 100
Alt<Key>Home	root | icon | window	f.set_activescreen home
Alt<Key>End	root | icon | window	f.set_activescreen "-0-0"
Alt<Key>minus	root | icon | window	f.toggle_autopan
Alt<Key>F2	window	f.identify
Alt<Key>F2	root	f.identify

See also:

- "About mnemonics and accelerators" (page 272)
- "About window manager functions" (page 272)
- "Configuring key bindings" (page 273)

1. A context of **icon** indicates window manager icons, not icons on the Desktop.

About mnemonics and accelerators

Mnemonics and accelerators are types of key bindings. Their most common use is to supply a keyboard interface to an application that is usually pointer-driven. Both accelerators and mnemonics allow you to invoke a variety of menu items without using the mouse. They are defined in association with window manager functions.

See also:

- "About window manager functions" (this page)
- "Adding or modifying window manager menus" (page 237) for information on how to specify mnemonics and accelerators

About window manager functions

Window manager functions are a component of every section of the window manager configuration files. The system-wide and local configuration files use window manager functions to define the behavior of keys, mouse buttons, and menu panes.

You configure key bindings by associating each key press sequence with at least one window manager function. For example, if you bind Ctrl⟨Key⟩Space to a client menu with the **f.menu** function, the client menu posts each time you press the ⟨Ctrl⟩⟨Space⟩ keys simultaneously. This key binding definition would look like this:

```
Keys  bindings_set_name
{
                .
    Ctrl<Key>Space  window     f.menu client_menu
                .
                .
}
```

See also:

- "Using window manager functions" (page 223) for a complete list and a detailed explanation of all of the default window manager functions

Configuring key bindings

To modify an existing key binding or to create a new key binding, perform the following steps. For more information on each of these steps, see the sections immediately following this procedure.

1. Open the desired window manager configuration file for editing.
   - **pmwm** mode:

     /usr/lib/X11/system.pmwmrc for system-wide changes

     $HOME/*.pmwmrc* for individual changes

   - **mwm** mode:

     /usr/lib/X11/system.mwmrc for system-wide changes

     $HOME/*.mwmrc* for individual changes

2. Locate the key binding section in the window manager configuration file. The key binding section uses the following syntax:

```
Keys  bindings_set_name
{
    key context   function
    key context   function
    key context   function
                .
                .
    key context   function
}
```

3. Configure the key binding specification, if desired.
4. Configure the function specification, if desired.
5. Configure the context specification, if desired.
6. Restart the window manager and test your new key binding.

See also:

- "Example of configuring key bindings" (page 282) for sample key binding definitions

Step 1: Editing a window manager configuration file

If you want to make system-wide changes to the default key bindings, open the system-wide window manager configuration file, */usr/lib/X11/system.pmwmrc* if you are using **pmwm** mode or *system.mwmrc* if you are using **mwm** mode. The system-wide file contains the default functionality for the window manager key bindings.

If you want to customize the key bindings in your local Graphical Environment, edit your personal window manager configuration file, **$HOME**/*.pmwmrc* if you are using **pmwm** mode or **$HOME**/*.mwmrc* if you are using **mwm** mode. The personal configuration file is not provided with the system by default. If it does not already exist, you must create your own by copying the appropriate system-wide window manager configuration file to *.pmwmrc* or *.mwmrc* in your **$HOME** directory.

NOTE Once the *.pmwmrc* or *.mwmrc* file exists in **$HOME**, it completely overrides the system-wide window manager configuration file. Therefore, make sure you copy the entire system file to your home directory, to avoid losing critical functionality.

See also:

- Chapter 12, "Customizing the window manager" (page 219) for more information on the system-wide and local window manager configuration files

Step 2: Locating the key binding section

When you locate the appropriate key binding section, you see entries with the following syntax:

```
Keys  bindings_set_name
{
    key context   function
    key context   function
              .
              .
}
```

Keys	the type of the binding set being defined
bindings_set_name	the name assigned to a set of bindings. The default name is "DefaultKeyBindings".
keys	the specification that defines the key press action

context — the specification that defines the window manager context in which the key specification becomes active. For example, a context of a **window** indicates that the keyboard input focus must be on a client window or a window management frame for the key specification to be effective.

function — the specification that defines one of the many window manager functions or actions. See "Using window manager functions" (page 223) for a list of these functions.

Step 3: Configuring the key binding specification

The window manager configuration files use window manager functions to define the behavior and control the functionality of key events. A key event describes an action that you take, such as pressing the "r" key to execute a function (for example: redrawing a window). Key events are single key presses; key releases are ignored. The key event specification has the following syntax:

key = [*modifier_list*]⟨Key⟩*key_name*

where:

modifier_list = *modifier_name* {*modifier_name*}

Designate any single key for the value in ***key_name***. If you are creating a mnemonic key binding, do not designate a value for ***modifier_name***.

If you are creating an accelerator key binding, make sure your key-action combination is unique by selecting one or more of the values in Table 15-2, "Modifiers" for the value in ***modifier_name***.

NOTE The ⟨Alt⟩ key is frequently labeled **Extend** or **Meta**. **Alt** and **Meta** can be used interchangeably for an event specification.

Table 15-2 Modifiers

modifier_name	Description
Ctrl	Control Key
Shift	Shift Key
Alt	Alt/Meta Key
Meta	Meta/Alt Key
Lock	Lock Key
Mod1	Modifier1
Mod2	Modifier2
Mod3	Modifier3
Mod4	Modifier4
Mod5	Modifier5

If you are editing an existing key binding, replace the old key specification with the new one. If you modify any of the default key bindings, make sure you do not remove functionality that you need.

If you are adding a new key binding, put the new key specification on a new line.

Step 4: Configuring the function specification

The syntax for naming a function is the same, no matter what type the function describes. The syntax for naming a function is:

function = ***function_name [function_args]***

function_name is one of the valid window manager functions, and ***function_args*** is a valid argument to the function. If ***function_args*** contains more than one word, the argument must be contained in quotes.

If you are editing an existing key binding, replace the old function specification with the new one.

If you are adding a new key binding, put the new function specification on a new line, after the related key specification.

See also:

- See "Using window manager functions" (page 223) for a complete list and a detailed explanation of all of the valid window manager functions

Step 5: Configuring the context specification

The syntax for the context specification is:

context = **app | border | frame | icon | root | title | window**

The context specification defines the window manager context in which the key specification becomes active; it indicates where the keyboard input focus

must be for the key specification to be effective. For example, a context of a **window** indicates that the keyboard input focus must be on a client window or a window management frame for the key binding to be effective. The button specification can be active in more than one context.

Table 15-3, "Key binding contexts" lists and describes the values that can be used for ***context***.

Table 15-3 Key binding contexts

Context	Description
app	The key binding is effective when the keyboard input focus is on the application window, not including the window management frame.
border	The key binding is effective when the keyboard input focus is on the border of the window management frame, not including the title bar.
frame	The key binding is effective when the keyboard input focus is on the window management frame around a client window, border, and title bar.
icon	The key binding is effective when the keyboard input focus is on an icon. Note that icon refers to window manager icons only, not Desktop icons.
root	The key binding is effective when: 1. the keyboard input focus is on the Root menu, and 2. neither a client window nor an icon is to be acted upon by the function.
title	The key binding is effective when the keyboard input focus is on the title area of the window frame.
window	The key binding is effective when the keyboard input focus is on a client window, title bar, or a window management frame.

If you are editing an existing key binding, replace the old context specification with the new one.

If you are adding a new key binding, put the context specification on a new line, after the related key and function specifications. See "Function constraints" (page 230) and Table 12-2, "Where functions can be used" (page 232) for a list of the contexts that are available to each of the window manager functions.

Step 6: Restarting the window manager

After you configure your new key bindings, you must restart the window manager before your changes can take effect. Restart the window manager by selecting the **Restart Window Manager** option from the **Root** menu. The **Root** menu is accessed by pressing and holding mouse button 1 on the Desktop background or, if the Desktop is not running, in the Root window.

Verify that the new key bindings are configured correctly by testing your new key sequences in the appropriate contexts. The key bindings are effective immediately after the window manager is restarted.

Creating a new key binding set

To create a new key binding set, perform the following steps. For more information on each of these steps, see the sections immediately following this procedure.

1. Open the desired window manager configuration file for editing.
   - **pmwm** mode:

     /usr/lib/X11/system.pmwmrc for system-wide changes

     $HOME/*.pmwmrc* for individual changes

   - **mwm** mode:

     /usr/lib/X11/system.mwmrc for system-wide changes

     $HOME/*.mwmrc* for individual changes
2. Locate the "DefaultKeyBindings" section in the window manager configuration file, copy it, then use it as a template for your new set.
3. Configure the new key, function, and context specifications.
4. Configure the new key binding set for use by the window manager with the **keyBindings** resource.

   Add this resource to */usr/lib/X11/app-defaults/Pmwm* (for **pmwm** mode), */usr/lib/X11/app-defaults/Mwm* (for **mwm** mode), or to the **$HOME/***.Xdefaults-**hostname*** file, using the following syntax:

   Pmwm*keyBindings: *bindings_set_name*

   or

   Mwm*keyBindings: *bindings_set_name*
5. Restart the window manager and test your new key binding set.

See also:

- "Example of configuring key bindings" (page 282) for sample key binding definitions

Step 1: Editing a window manager configuration file

If you want to make system-wide changes to the default key bindings, open the system-wide window manager configuration file, */usr/lib/X11/system.pmwmrc* if you are using **pmwm** mode or *system.mwmrc* if you are using **mwm** mode. The system-wide file contains the default functionality for the window manager key bindings.

If you want to customize the key bindings in your local Graphical Environment, edit your personal window manager configuration file, **$HOME**/*.pmwmrc* if you are using **pmwm** mode or **$HOME**/*.mwmrc* if you are using **mwm** mode. The personal configuration file is not provided with the system by default. If it does not already exist, you must create your own by copying the appropriate system-wide window manager configuration file to *.pmwmrc* or *.mwmrc* in your **$HOME** directory.

NOTE Once the *.pmwmrc* or *.mwmrc* file exists in **$HOME**, it completely overrides the system-wide window manager configuration file. Therefore, make sure you copy the entire system file to your home directory, to avoid losing critical functionality.

See also:

- Chapter 12, "Customizing the window manager" (page 219) for more information on the system-wide and local window manager configuration files

Step 2: Locating the DefaultKeyBindings section

When you locate the "DefaultKeyBindings" section, you see entries with the following syntax:

```
Keys  DefaultKeyBindings
{
    keys context   function
    keys context   function
              .
              .
}
```

Keys	the type of the binding set being defined
"DefaultKeyBindings"	the default name assigned to a set of bindings

keys	the specification that defines the key press action. See Table 15-2, "Modifiers" (page 276) for a list of modifiers that can be used with the key press actions.
context	the specification that defines the context in which the key specification becomes active. See Table 15-3, "Key binding contexts" (page 277) for a list and description of the values that can be used for ***context***.
function	the specification that defines one of the many window manager functions or actions. See "About window manager functions" (page 272) for a basic description of functions. See "Using window manager functions" (page 223) for a list and a detailed description of window manager functions.

Create a template for your new binding set by copying the "DefaultKeyBindings" section and placing the copy below the existing section. Assign the new set a different name.

Step 3: Defining key, function, and context specifications

Each key binding definition consists of key event, function, and context specifications. Use the template you created in Step 2 (page 279) to create these new specifications.

- Define the new key event specification on the first line. The key event specification has the following syntax:

  key = [***modifier_list***]⟨**Key**⟩***key_name***

  where:

  modifier_list = ***modifier_name*** {***modifier_name***}

  Key events are single key presses; key releases are ignored. Designate any single key for the value in ***key_name***. If you are creating a mnemonic key binding, do not designate a value for ***modifier_name***. If you are creating an accelerator key binding, make sure your key-action combination is unique by selecting one or more of the values in Table 15-2, "Modifiers" (page 276) for the value in ***modifier_name***.

- Define the function specification, after the key event specification. The syntax for naming a function is:

  function = ***function_name*** [***function_args***]

  function_name is one of the valid window manager functions and ***function_args*** is a valid argument to the function. If ***function_args*** contains more than one word, the argument must be contained in quotes.

  See "Using window manager functions" (page 223) for a complete list and a detailed explanation of all of the valid window manager functions.

- Define the context specification after the function specification. The syntax for the context specification is:

  context = **app | border | frame | icon | root | title | window**

  The context specification defines the context in which the key specification becomes active; it indicates where the keyboard input focus must be for the key specification to be effective. The button specification can be active in more than one context. See Table 15-3, "Key binding contexts" (page 277) for a list and description of the values that can be used for ***context***.

Step 4: Specifying the keyBindings resource

Define the name of the new key binding set through the **keyBindings** resource, using the appropriate resource file.

If you are making local changes, you must define the new binding set in the **$HOME**/*.Xdefaults-**hostname*** file. (If this file does yet not exist, create a file in your **$HOME** directory named *.Xdefaults-**hostname***, where ***hostname*** is the name of the host, or machine, where the window manager is running.)

If you want the new key binding set to be used by all users on your system, specify the resource in */usr/lib/X11/app-default/Pmwm* (if you are using **pmwm** mode) or */usr/lib/X11/app-default/Mwm* (if you are using **mwm** mode).

The syntax of the resource specification is the same for all of the window manager configuration files:

Pmwm*keyBindings: ***new_key_bindings_set_name***

or

Mwm*keyBindings: ***new_key_bindings_set_name***

See also:

- Chapter 5, "Understanding resources" (page 79) for more information on resources and the *.Xdefaults-**hostname*** file

Step 5: Restarting the window manager

After you create your new key binding set, you must restart the window manager so your changes can take effect. Restart the window manager by selecting the **Restart Window Manager** option from the **Root** menu. The **Root** menu is accessed by pressing and holding mouse button 1 on the Desktop background or, if the Desktop is not running, in the Root window.

Verify that the new key binding set was created correctly by testing your new key sequences in the appropriate contexts. The key bindings are effective immediately after the window manager is restarted.

Example of configuring key bindings

This section provides a comprehensive example that ties together some of the concepts and procedures discussed in this chapter.

The following example assumes that you are using the default **pmwm** mode of the window manager, that you want to create a new key binding set for your local environment, and you want to name the set "MyKeyBindings". MyKeyBindings consists of four key binding definitions that produce the following results:

- Pressing ⟨Shift⟩⟨Esc⟩ with the keyboard input focus on either an icon or window posts the menu named for the **Pmwm*windowMenu** resource specification. (See line 6 of the following sample file.)
- Pressing ⟨Ctrl⟩⟨Space⟩ with the keyboard input focus on either an icon or window, or on the **Root** menu, posts the menu named "MyRootMenu". (See line 7 of the following sample file.)
- Pressing ⟨Alt⟩⟨Esc⟩ with the keyboard input focus on either an icon or window, or on the **Root** menu, sets the keyboard input focus to the next icon or window in the set of icons or windows managed by the window manager. (See line 8 of the following sample file.)
- Pressing ⟨Alt⟩ ⟨Shift⟩⟨Esc⟩ with the keyboard input focus on either an icon or window, or on the **Root** menu, sets the keyboard input focus to the previous icon or window in the set of icons or windows managed by the window manager. (See line 9 of the following sample file.)

To create this new set:

1. Open the *.pmwmrc* file in your **$HOME** directory for editing. (If *.pmwmrc* does not already exist, create it by copying */usr/lib/X11/system.pmwmrc* to *.pmwmrc* in your **$HOME** directory.)
2. Locate the "DefaultKeyBindings" section in the *.pmwmrc* file and place a copy of the section underneath the default definitions. You can use this copy as a template for your new key binding set.

3. Rename "DefaultKeyBindings" to "MyKeyBindings".
4. Edit the template so that it looks like this:

```
1   #
2   # key binding descriptions
3   #
4   Keys  MyKeyBindings
5   {
6       Shift<Key>Escape      icon|window          f.post_wmenu
7       Ctrl<Key>space        icon|window|root     f.menu MyRootMenu
8       Alt<Key>Escape        icon|window|root     f.next_key
9       Alt Shift<Key>Escape  icon|window|root     f.prev_key
10  }
```

5. Define the **keyBindings** resource to announce the new key binding set to the window manager. Specify the resource in the **$HOME**/*.Xdefaults-hostname* file, where ***hostname*** is the name of the host, or machine, where the window manager is running. The resource specification should look like this:

   Pmwm*keyBindings: MyKeyBindings

6. Restart the window manager so that the new key binding information is implemented.

Chapter 16

Customizing the Desktop with rules

The behavior and appearance of the Desktop are not fixed, and are determined by files which you can alter to provide a different behavior or appearance. These files are called the Desktop "rule files". Each rule file consists of a sequence of "rule clauses".

For maximum flexibility, the physical appearance and design of the Desktop can also be altered by changing the characteristics specified in the Desktop resource files. See Appendix B, "Desktop resources" (page 403) for more information.

Specifically, this chapter describes:

Rule clauses

There are six different types of rule clauses, each dealing with a different aspect of the Desktop's configuration. These are:

- **desktop_layout**
- **icon_rules**
- **initial_actions**
- **locked_on_desktop**
- **menu**
- **final_actions**

For example, **icon_rules** clauses define the behavior of an icon in the Desktop, or the action when the user manipulates it in a particular way.

See also:

- Chapter 21, "Configuring icons" (page 327)
- Chapter 22, "Configuring Desktop windows" (page 333)
- Chapter 23, "Configuring directory windows" (page 337)
- Chapter 24, "Configuring Desktop menus" (page 341)
- Chapter 20, "Creating objects for the Desktop" (page 315) for an extensible way to configure icon behavior and appearance
- **desktop_layout**, **icon_rules**, **initial_actions**, **locked_on_desktop**, **menu**, and **final_actions** in the **xdt3**(XC) manual page

Defining the scope of rules

It is possible to provide rules that will apply to all things of one type, such as all the directory windows or all users of the system, by defining an appropriate rule clause in a system-wide module. More usually, you will want to limit the effect of your rules to certain icons, or certain users on the system. This is referred to as defining the "scope" of the rules.

Usually, the location of the file containing a rule determines its scope. For example, you can provide special behavior for one particular user by providing rule clauses in a suitable file in that user's home directory.

There are two essentially different ways of specifying the scope of a rule: "implicitly" and "explicitly". You can specify the scope implicitly by choosing the location of the file containing the rule. However, for **icon_rules** clauses you can also specify the scope explicitly, by specifying a pattern that matches the files or directories to which you want the rule to apply.

In some cases these methods of specifying the scope are interchangeable. For example, you could write a rule to apply to all the files in one directory using either of the following two methods:

- provide an **icon_rules** clause matching all files, and put it in a local rule file in the directory to be affected
- provide an **icon_rules** clause matching files with the appropriate pathname, and put it in a system-wide module or a user rule file

See also:

- "Specifying scope implicitly" (this page)
- "Specifying the scope explicitly" (page 289)

Specifying scope implicitly

You can specify the scope implicitly by choosing where you locate the rule:

all users on the system	use a system-wide module (page 301)
some users	use a module specified for a particular UNIX group ID, or a custom user type (page 305)
one user	use a user rule file (page 288) in that user's home directory
the icons in one directory	use a local rule file (page 289) in that directory
the icons on a desktop	use the desktop rule file (page 289) for that desktop
dynamically	use dynamically loaded rules (page 289)

The contents of user rule files, modules, user type rules, and the system rule file are only examined when the Desktop starts or after the Deskshell command **reset**.

See also:

- "Changing the behavior for all users" (this page)
- "Changing the behavior for different types of user" (this page)
- "Changing the behavior for a single user" (this page)
- "Changing the behavior of a directory" (page 289)
- "Changing the behavior of a desktop" (page 289)
- "Changing behavior dynamically" (page 289)

Changing the behavior for all users

To provide custom rules for all users, you should use a module. You should **not** edit the system rule file.

See Chapter 17, "Using Desktop modules" (page 301) for more information.

Changing the behavior for different types of user

To provide rules for a particular type of user, create a user type.

See Chapter 18, "Defining Desktop user types" (page 305) for more information.

Changing the behavior for a single user

To provide rules which will apply to a specific user on the system, you should create a user rule file in that user's home directory. The name used for user rule files is, by default, named *.xdtuserinfo*.

User rule files allow you to give each user's desktops a different appearance and behavior. For example, for advanced users you can define short cuts for all their frequently-used operations, whereas for less experienced users you can provide a simpler system in which they are less likely to make mistakes.

If you have a large number of users of a particular type, you may like to consider creating a new user type.

See also:

- Chapter 18, "Defining Desktop user types" (page 305)

Changing the behavior of a directory

To provide different behavior for files in one directory, you should include rules in a local rule file in that directory. The name of the local rule file is *.xdtdir/ll_TT*, where, by default, *ll_TT* is set to *en_US*.

With local rule files you can define special behavior for items within specific directories. For example, an archiving directory could be created which would compress any file dragged into its directory window.

Changing the behavior of a desktop

To define the appearance and behavior of desktops, you should provide rules in the desktop rule file. Generally, these rules specify which files and directories are on the desktop, and what their positions are. The desktop rule file has the same name as the desktop, with the extension *.dt*.

The contents of desktop rule files are only examined when the file is loaded; subsequent changes are ignored until the next time that desktop is opened. When a desktop is closed, the rule file is automatically updated to reflect any changes in icon positions.

Changing behavior dynamically

To change the behavior of the Desktop dynamically, you can load a rule using the **dynamic_rule** Deskshell command. Dynamic rules cannot be changed once loaded, but can be unloaded when no longer required.

See also:

- **dynamic_rule** in the **deskcommands**(XC) manual page

Specifying the scope explicitly

For **icon_rules** clauses, you specify the scope of the rule explicitly by providing a specification that matches the files or directories in which you are interested.

The format of the **icon_rules** clause is:

```
icon_rules
{
   pattern [/class]
   {
      clauses applying to specified files
   }
}
```

The specification consists of two parts:

- a ***pattern***, which specifies the names of files or directories to which you want the rules to apply, using wildcards to select groups of files
- an optional ***class***, which allows you to filter out certain categories of file on the basis of the ownership permissions, or the type of file

See also:

- "Patterns" (this page)
- "Classes" (page 291)

Patterns

Patterns look like filenames or pathnames, but can contain certain special characters or wildcards. There must be at least one space or newline before specifying a ***class***.

The **basename** of a pattern can include the following wildcard characters:

?	any single character. For example, **a?c** includes the files *aac*, *abc* and so forth, but not the file *abbc*.
*	any sequence of characters, including none. For example, **a*c** includes the files *ac*, *abc*, *acbc* and so forth.
[***chars***]	any one of the specified set of characters. For example, **[abc]d** includes the files *ad*, *bd* and *cd* but no others.
[!***chars***]	none of the specified characters. For example, **[!a]bc** includes *bbc*, *cbc* and so forth, but not *abc*.

These patterns can be combined. For example, **[!A]*** means any file beginning with a character other than *A*.

See also:

- "Relative patterns" (this page)
- "Absolute patterns" (this page)
- "Rule file precedence" (page 295)
- **basename** in the **deskcommands**(XC) manual page

Relative patterns

If the pattern does not begin with a " / ", it is a "relative pattern", which can match files anywhere in the system. The pattern cannot include " / ".

Rules following relative patterns in a local rule file apply to files in the directory that match the pattern. Rules following relative patterns in all other rule files apply to all files whose basenames match the pattern.

Absolute patterns

If the pattern begins with a " / ", it is an "absolute pattern", which only matches files in a specific directory.

Absolute patterns cannot occur in local rule files. Wildcards can only be included after the last " / " in an absolute pattern.

Rules following absolute patterns apply to files in the directory given by the pattern up to the last " / ", and whose basename matches the part after the last " / ". These rules take precedence over those following relative patterns in any rule file.

Classes

Classes are used to represent the properties of files in a concise form. These properties fall into the following six sets:

- file type
- execute permissions
- read/write permissions
- ownership
- symbolic
- variation class

A file has one property from each set. The properties are each represented by a character, so that the class of a file consists of exactly six characters. The Desktop always specifies classes in upper case, though it accepts classes in either case. For example, an executable file might have the full class definition:

```
FXWM-0
```

The classes restrict the group of files affected by the subsequent clauses, and so omitting a specifier from one set of options will match all the alternatives.

The following examples show some of the most useful class specifications:

none	everything
D	directories
F	files
FE	executable files
FX	files executable by the user
FN	data files
FNW	data files that the user can alter
FNR	data files that the user can read

See also:

- **fileclass** in the **deskcommands**(XC) manual page for details on the different class characters, and how to read the class of a file

Effect of rules in different rule files

The relationship between the different types of rule file, and the rule clauses they can contain, is shown below.

- **Local rule files:**

icon_rules	affect icons in the directory containing the rule file
locked_on_desktop	ignored
desktop_layout	ignored
initial_actions	occur when the directory is opened
final_actions	occur when the directory is closed
menu	available in the directory containing the rule file

- **Desktop rule files:**

icon_rules	affect icons on the desktop
locked_on_desktop	applies to the desktop
desktop_layout	applies to the desktop
initial_actions	occur when the desktop is opened
final_actions	occur when the desktop is closed
menu	available on the desktop

- **User rule files:**

icon_rules	affect all of the user's icons
locked_on_desktop	applies to the main Desktop.
desktop_layout	applies to desktops not already holding a **desktop_layout** clause
initial_actions	occur when the Desktop starts (after the system rule file's **initial_actions**)
final_actions	occur when the Desktop exits (before the system rule file's **final_actions**)
menu	available to all directories and desktops

- **System rule file, user type rules, and modules:**

icon_rules	affect all icons
locked_on_desktop	applies to the main Desktop
desktop_layout	applies to desktops not already holding a **desktop_layout** clause
initial_actions	occur when the Desktop starts (before the user rule file's **initial_actions**)
final_actions	occur when the Desktop exits (after the user rule file's **final_actions**)
menu	available to all directories and desktops

- **Dynamic rules:**

icon_rules	affect all icons while loaded
locked_on_desktop	apply to the main Desktop
desktop_layout	applies to desktops not already holding a **desktop_layout** clause
initial_actions	occur when the rule is installed, if the **-x** option is used when loading the rule
final_actions	occur when the rule is removed, if the **-x** option is used when unloading the rule
menu	available to all directories and desktops

- **Built-in rules:**

icon_rules	affect all icons
locked_on_desktop	none
desktop_layout	none
initial_actions	none
final_actions	none
menu	available to all directories and desktops

See also:

- **icon_rules, locked_on_desktop, desktop_layout, initial_actions, final_actions,** and **menu** in the **xdt3**(XC) manual page

Rule file precedence

The rule files are searched in the order listed below:

1. Absolute patterns in:
   - desktop file, if the file or directory is on a desktop
   - dynamic rules, in the order specified when they are loaded
   - user rule file
   - system rule file, user type rules, and modules
   - built-in rules
2. Relative patterns in:
   - local rule file
   - desktop file, if the file or directory is on a desktop
   - dynamic rules, in the order specified when they are loaded
   - user rule file
   - system rule file, user type rules, and modules
   - built-in rules

See also:

- "How Deskshell commands are executed" (page 361)
- **initial_actions** and **final_actions** in the **xdt3**(XC) manual page

Structure of rule files

Rule files are text files, and can be created and edited using any suitable text editor, such as **vi** or **scoedit**.

Rule files consist of sequences of "clauses". Each clause can have one of the following two forms:

keyword=value ;

or

keyword { ***body*** } [;]

Each rule clause in a rule file starts with a ***keyword*** specifying what type of rule it is. The keyword may have an abbreviation. Each keyword is typically followed by either a value, or a block enclosed in matching curly brackets containing further clauses.

In the second form the semicolon is optional. The ***body*** is normally a sequence of commands.

Rule files are block structured like the programming languages C or Pascal, and in general the layout is not important. For example, the following two rules are equivalent:

```
icon_rules {* /D{picture=dir.px;}* /F{picture=file.px;}}
```

and:

```
icon_rules
{
   * /D
   {
      picture=dir.px ;
   }
   * /F
   {
      picture=file.px ;
   }
}
```

The recommended layout, shown in the second example above, helps clarify the structure of the rule, and makes it easier to match pairs of brackets. This style of layout will be used for all the examples in this guide.

Within a rule file you can include:

- other rule files with **%+*filename*+**
- the values of UNIX environment variables with **%$*variable*$**
- comments with **%//**

To illustrate the characteristics of a typical rule, look at the following simple **icon_rules** clause, which defines the characteristics of an **Edit** icon:

```
icon_rules
{
   Edit /F
   {
      picture=edit.px;
      title=Editor;
      trigger_action: drop
      {
         edit -merge $dynamic_args
      }
   }
}
```


The rule is introduced by the keyword **icon_rules**. This specifies it as a command to determine the behavior of an icon or group of icons.

The icons to which the command applies are defined by the construct `Edit /F`, where `/F` denotes files. In this case the rule applies to any file with the filename *Edit*.

The **picture** command specifies the file containing the picture for the icon, and the **title** command defines its title.

The **trigger_action** command defines what happens when the user performs an action on, or "triggers", an icon in a particular way. In this case the trigger "action" is **drop**, which occurs when the user drops one or more icons onto this icon using mouse button 1. This is followed in curly brackets by the action to be carried out under the specified circumstances.

Further examples of rules are given in the **xdt3**(XC) manual page. The default system rules for your system can be found in */usr/lib/X11/IXI/XDesktop/rules/system/xdtsysinfo*. Please do not make any changes to this file, as they will not be supported if you later choose to upgrade. See Chapter 17, "Using Desktop modules" (page 301) and Chapter 18, "Defining Desktop user types" (page 305) for information on how to configure system-wide Desktop behavior without modifying the system rule file.

See also:

- **%+*filename*+**, **%$*variable*$**, and **%//** in the **deskshell**(XC) manual page
- **drop**, **picture**, **title**, and **trigger_action** in the **xdt3**(XC) manual page

Processing filenames in rules

This section discusses how filenames are represented in rule file commands, and describes the commands for manipulating filenames.

See also:

- "Referring to file and directory names" (this page)
- "Canonical form" (page 299)
- "Filename processing commands" (page 299)
- "Specifying actions" (page 299)

Referring to file and directory names

When a file or directory is referred to in the Desktop, its name may be used in four ways:

absolute pathname	the full name of the file or directory, which always begins with a slash
basename	the name of the file/directory within its directory. It is the part of the absolute pathname following the last slash. In addition, the file or directory name's "extension" is the part of the basename from the last dot.
dirname	the name of the directory holding the file or directory. It is the part of the absolute pathname preceding the last slash.
relative pathname	the path to a file or directory, starting from your home directory

For example, the various names of the file */user/fred/work/letter.ed* are:

absolute pathname	*/user/fred/work/letter.ed*
basename	*letter.ed*
extension	*.ed*
dirname	*/user/fred/work*
relative pathname	*work/letter*

NOTE There is one special case: the dirname of " / " is /. (slash-dot), and its basename is / (slash).

Canonical form

In representing filenames, "." represents the current directory and ".." represents the parent directory. The "canonical form" eliminates these symbols from the pathname to represent the pathname without any redundancy.

Filename processing commands

These commands perform operations on each of a list of filenames. If they are not used in a list substitution construct of the form **'(...)**, then the results are sent to standard output, separated by the value of the first string in the variable **ofs** if there is more than one argument.

absreadlink	absolute pathname of the value of a symbolic link
basename	basename of its argument
canonical	argument converted to canonical (non-redundant) form
dirname	canonical form of the directory of its argument
extension	extension of its argument
fileclass	class of its argument
followlink	absolute pathname of the final destination of a symbolic link
readlink	contents of a symbolic link
relativepath	pathname relative to the user's home directory
unextended	canonical form of the argument with its extension removed

See also:

- **'(...)** and **ofs** in the **deskshell**(XC) manual page

Specifying actions

In the example in "Structure of rule files" (page 295), the action is a single command to run the **Edit** program with suitable arguments, depending on the files that were dragged onto its icon.

A list of the names of the files dragged onto the icon is provided in the variable **dynamic_args**. So, for example, if the icons **chapter1** and **chapter2** were dragged onto the **Edit** icon the action would be to run the command:

edit -merge chapter1 chapter2

See also:

- Chapter 19, "Defining Desktop triggers" (page 309) for more information about defining actions
- **dynamic_args** in the **deskshell**(XC) manual page

Chapter 17

Using Desktop modules

To provide rules that apply to all users on the system, use a system-wide "module". Modules are sets of rules that have the same effect as if they were in the system rule file. You should not edit the system rule file itself.

Using modules provides the following advantages:

- you do not need to understand the system rule file
- your own sections are separated from others, making them easier to support
- it is easier to replace or update a module than to maintain an edited system rule file
- modules can be specified for individual users or UNIX groups, using resources

The **MODULEDIR** environment variable specifies which directories are searched for modules. By default, this includes */usr/lib/X11/IXI/XDesktop/rules/modules* (for system-wide modules) and **$HOME**/*.xdt_dir/modules* (for your own custom modules).

To specify modules for a user or UNIX group, the system administrator should set the following system-wide resources, typically in the default preferences file for the appropriate user type:

- **XDesktop3.Rules.defaultModules:** ***module1 module2 ...***
  The ***module1, module2*** modules are loaded for all users.
- **XDesktop3.Rules.*group_ID*.groupModules:** ***module1 ...***
  The modules are only loaded for those users in the UNIX group ***group_ID*** (for example, 100). The group must be specified in numeric form, and not by any symbolic group name.
- **XDesktop3.Rules.*username*.userModules:** ***module1 ...***
  The modules are loaded only for the user ***username***.

In addition to the modules described here, there are two special types of module: "auto" and "loop".

See also:

- "Auto modules" (this page)
- "Loop modules" (this page)
- "Text displayed by modules" (page 303)
- Chapter 18, "Defining Desktop user types" (page 305)

Auto modules

These modules are loaded automatically, per the values of the resources specified in Chapter 17, "Using Desktop modules" (page 301). Auto modules are distinguished by the suffix *.auto*.

Loop modules

These modules are performed periodically within a background loop. This loop is run at start-up time and then every ***n*** seconds, where ***n*** is defined by the **XDesktop3.Rules.loopDelay** resource.

Loop modules are distinguished by the prefix *Loop_*.

You should keep loop modules short. For example, the Desktop uses a loop module for a directory contents checker. If you want a loop module to run every ten times the main loop is run once, for example, you can use a counter within the module.

The system administrator can specify loop modules on a per user or per group basis, using the following system-wide resources:

- **XDesktop3.Rules.defaultLoopModules**
- **XDesktop3.Rules.*group_ID*.groupLoopModules**
- **XDesktop3.Rules.*user_name*.userLoopModules**

Text displayed by modules

All text strings displayed by a module should be stored in the file *module/ll_TT*, where ***ll*** is a two-character code for the language (as defined by the ISO 639 standard) and ***TT*** is a two-character code for the territory (as defined by the ISO 3166 standard). By default, the text strings are located in *module/en_US*.

The file should contain a number of Deskshell variable assignments. The rules of the module should refer to the variable names to determine what text to display.

Chapter 18

Defining Desktop user types

You may need to support users that have differing UNIX experience levels, or otherwise configure the Desktop for a variety of user types. The Desktop provides a "user type" mechanism to allow you to do this.

All user type configuration files are kept within the main Desktop rules directory, */usr/lib/X11/IXI/XDesktop/rules*. Each user type has its own subdirectory with name ***UserType****.user*, for example *SCO.user*.

A user type directory must contain a file called *Rule.dr*, which holds the default rules for this user type. The *Rule.dr* file can read and manipulate any files it needs to. For example, within the *SCO.user* user type directory, you may find the following:

Rule.dr the main rule file for this user type, which is treated as if it were part of the system rule file

menus.dr a rule file used to set up all the menus for this user type

Main.dt the file used for the default main Desktop for each user of this type

objects a subdirectory containing the objects used by default for each user of this type

ll_TT*.prf* The default preferences file for this user type, in the language ***ll_TT***, where ***ll*** is a two-character code for the language (as defined by the ISO 639 standard) and ***TT*** is a two-character code for the territory (as defined by the ISO 3166 standard). By default, the *SCO.user* user type provides the *en_US.prf* file.

ll_TT The language file containing all the text strings used for this user type, in the form of Deskshell variable assignments. The rules of the user type should refer to the variable names to determine what text to display.

See also:

- "Creating a new user type" (this page)
- "Determining a user type" (this page)

Creating a new user type

To create a new user type, copy an existing user type and modify the copy. It is important to remember to replace all instances of the old user type name (e.g. *SCO*) in all the files with the new name, to ensure that the correct files are used.

To specify the modules to use for this user type, use lines in the appropriate preferences files (i.e., *en_US.prf* in the *SCO.user* directory) similar to:

```
irl 'XDesktop3.Rules.defaultModules: modules...'
irl 'XDesktop3.Rules.group_ID.groupModules: modules...'
irl 'XDesktop3.Rules.user_name.userModules: modules...'
```

You can set any of the resources used for modules. These resource lines should contain the full list of modules to be used for this user type; you may need to examine the default Desktop resource file to discover the modules used for the existing user type.

See also:

- "Changing the behavior for all users" (page 288)
- **irl** in the **deskcommands**(XC) manual page

Determining a user type

Three resources are checked to determine the user type assigned to each user:

1. **XDesktop3.Rules.*user_name*.userType**
2. **XDesktop3.Rules.*group_ID*.groupUserType**
3. **XDesktop3.Rules.defaultUserType**

If none of these are set, then the user type "SCO" is used.

To assign a user a particular user type:

1. Edit the user's *.Xdefaults-**hostname*** file, where ***hostname*** is the name of the machine on which the client is running, to include the resource:

   XDesktop3.Rules*userType: ***UserType***

2. Include the following line in the preferences files for that user type:

   ```
   irl 'XDesktop3.Rules*userType: UserType'
   ```

3. Remove the user's **$HOME/***.xdt_dir* directory, if it exists.

The **userType** resource is set in the user's *.Xdefaults-**hostname*** file to make the initial switch to the new user type. After that, the setting is done as part of the preferences file in the user type directory, and it does not matter if the user removes the line from their personal resource file.

Chapter 19

Defining Desktop triggers

A trigger is a general-purpose method for performing an action specific to a particular icon or window. For instance, if you want an icon to do something when you double-click on it, you should define an action for the **activate** trigger for that icon.

Some triggers correspond directly to an action by the user, such as clicking once on an icon to select it (the **select** trigger). These triggers are defined in the Desktop "trigger table", which is specified by the Desktop trigger mappings resource. Triggers not defined in the trigger table are commonly used for higher-level functions such as requesting help on an icon, or printing.

This chapter describes:

- what triggers are (page 310)
- the different types of triggers (page 310)
- variables that can be set by triggers (page 313)

See also:

- Chapter 26, "Mapping mouse triggers for the Desktop" (page 371)
- Appendix B, "Desktop resources" (page 403)

About triggers

When a user interacts with a Desktop icon or window background, the interaction is looked up in the trigger table to derive the trigger name corresponding to that action.

For example, if a user double-clicks on an icon with mouse button 1, this is converted to the trigger name **activate** in the trigger table. The Desktop then executes the command:

```
actions_of activate
```

Arguments are set to the appropriate values for the triggered icon and the icon's location.

The **actions_of** command searches the **icon_rules** clauses that match the triggered icon for the following clauses:

- **trigger_action: activate**
- **trigger_action: s***
- **trigger_action: ***

The above clauses specify an exact match, a match against any static trigger, and a match against any trigger at all, respectively.

The first match found causes a new thread to be created to execute the script specified in the **trigger_action** clause.

See also:

- "Threads" (page 362)
- **actions_of** in the **deskcommands**(XC) manual page
- **"*"**, **s***, **activate**, **icon_rules**, and **trigger_action** in the **xdt3**(XC) manual page

Types of trigger

The three different types of trigger found in the trigger table are defined as follows:

static trigger	the icon or window background is clicked or double-clicked without moving the mouse
dynamic trigger	the icon is dragged and dropped onto another icon or window background
hold trigger	the mouse button is held down on an icon or window background

See also:

- "Static triggers" (this page)
- "Dynamic triggers" (this page)
- "Hold triggers" (page 312)
- "Icons and windows" (page 312)

Static triggers

Clicking once is called "single-clicking". Clicking twice in quick succession, without moving the mouse pointer, is called "double-clicking" or "activating". Single and double clicks are referred to as "static" triggers because the mouse does not move during the action.

To make the rules as portable as possible, the static triggers are pre-defined with names as follows:

select	single-click on an icon picture
alt_select	single-click on an icon picture with mouse button 2
rename	single-click on an icon title
alt_rename	single-click on an icon title with mouse button 2
activate	double-click on an icon
alt_activate	double-click on an icon with mouse button 2
deselect	single-click on a window background
report	double-click on a window background
alt_report	double-click on a window background with mouse button 2
s*	matches any static trigger

Dynamic triggers

The action of dragging one or more icons and dropping them onto another icon is called a "dynamic" trigger, to contrast it with a static trigger. This is also sometimes referred to as a "drag" trigger.

The dynamic triggers are pre-defined as follows:

drop	drop one or more icons on an icon or window background
alt_drop	drop one or more icons on an icon or window background with mouse button 2
d*	matches any drag trigger

Hold triggers

"Hold" triggers activate an icon or directory window when the user presses one of the mouse buttons and holds it down without moving the pointer.

The hold triggers are pre-defined as follows:

menu	hold mouse button 3 on an icon picture or title
popup_menu	hold mouse button 3 on a window background
h*	matches any hold trigger

Icons and windows

You will notice that some triggers affect an icon, and some affect the background of a window. A slightly different command is used to define the actions to perform in each case.

Icon triggers require a corresponding **trigger_action** clause in an **icon_rules** clause matching that icon. Background triggers require a **drop_in_action** clause in an **icon_rules** clause for that window.

Triggers requiring a **drop_in_action** clause are:

- **drop** and **alt_drop** (*when referring to windows, not icons*)
- **report** and **alt_report**
- **deselect**
- **popup_menu**

All other triggers use a **trigger_action** clause.

See also:

- **drop_in_action**, **icon_rules**, and **trigger_action** in the **xdt3**(XC) manual page

Variables

The Desktop sets some variables when a trigger occurs. You can use these in rule files to find out what was triggered and how it was triggered.

You can also trigger things using commands, just as if the user had performed the appropriate action. The same variables are set in this case.

The following sections describe the circumstances under which these variables are set:

See also:

- **trigger**, **d_desktop**, **s_desktop**, **d_position**, **s_position**, **dynamic_args**, and **static_arg** in the **deskshell**(XC) manual page

Click or hold

When the user clicks or holds on an icon or window background, the following variables are set to the values shown below.

trigger	trigger name
static_arg	icon or window name
s_position	position of the click or hold
dynamic_args	selected icons
d_position	the string **V**, effectively meaning it does not matter
d_desktop	an empty list
s_desktop	click or hold in a: • desktop window — the name of the desktop • directory window — an empty list

Drag

When the user drops one or more icons onto an icon or a window background, the following variables are set to the values shown below.

trigger trigger name

static_arg icon or window name

s_position position of the click or hold

dynamic_args if the drag started on a:

- selected icon — the name of that icon followed by all the other selected icons
- non-selected icon — the name of that icon
- window background — an empty list

d_position position of the start of the drag

d_desktop if the drag started in a:

- desktop window — the name of the desktop
- directory window — an empty list

s_desktop if the drag ended in a:

- desktop window — the name of the desktop
- directory window — an empty list

Menu selection

When the user chooses a command from a menu, the following variables are set to the values shown below.

static_arg if the menu was popped up from:

- an icon — the name of the icon
- window background — the name of the window

dynamic_args selected icons

d_desktop if the menu was popped up in a:

- desktop window — the name of the desktop
- directory window — an empty list

Chapter 20
Creating objects for the Desktop

Objects are useful for linking mouse actions or "triggers" to system or Desktop actions and tasks. Use objects when you want to implement applications on the Desktop. A single object, represented by a user-selected icon, can have different scripts to interact with a binary in different ways.

Because objects are used to define behavior for single icons, **icon_rules** clauses are needed to define Desktop behavior that affects more than a single icon. Whenever possible, however, objects should be used instead of **icon_rules** clauses. For example, objects should be used to implement all stand-alone applications on the Desktop, while **icon_rules** clauses in a local rule file should be used to define behavior uniform to all icons in a particular directory. For more information on **icon_rules** clauses, see "Rule clauses" (page 286).

Some advantages of objects over **icon_rules** clauses is that objects are self-contained, well-suited for portability, easily exchanged from user to user, simply structured and easy to debug.

There are two ways to create objects:

- using the **Object Builder** client, which is the preferred tool for creating objects since it greatly simplifies the job, or
- by manually editing and creating the appropriate configuration files and directories

Creating an object using the Object Builder

The **Object Builder** lets you define the title, picture (icon), double-click actions, and drag-and-drop actions associated with a Desktop object.

There are two ways you can use the **Object Builder** to define an object. You can:

- use an existing object as a template for defining a new object (page 318) by opening the object, modifying its definitions, and then saving the object under a new name, or
- open a new, undefined object (page 322) and install a picture and action definitions separately

To use the **Object Builder**, open the *Controls* window and double-click on the **Object Builder** icon.

When you start the **Object Builder**, you see the following:

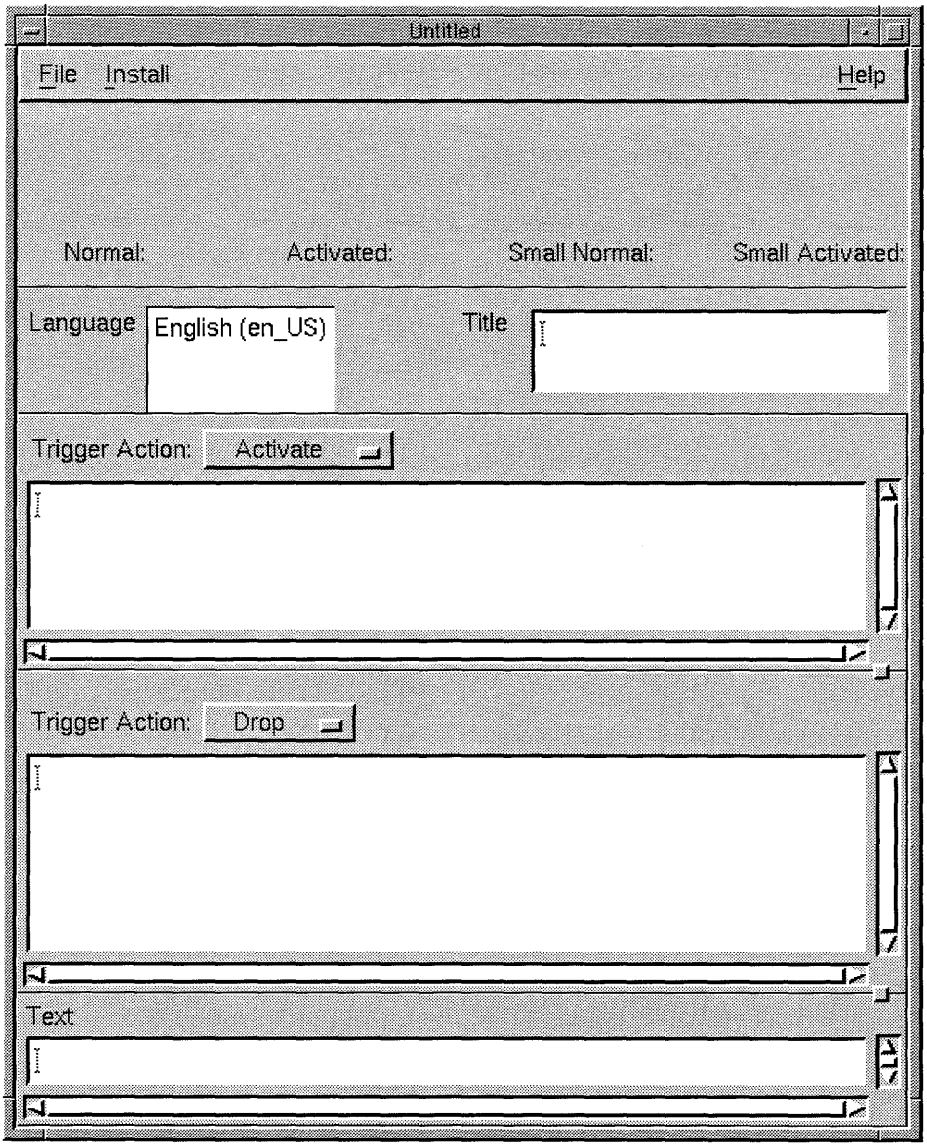

Figure 20-1 Object Builder window

See also:

- "Changing an action definition" (this page)
- "Opening an existing object" (this page)
- "Installing action definitions" (page 319)
- "Installing a picture" (page 320)
- "Installing an executable" (page 321)
- "Saving an object" (page 321)
- "Opening a new object" (page 322)
- **objbld**(XC) manual page for information on running the **Object Builder** from the UNIX command line.

Changing an action definition

You can use existing action scripts in a file to define the actions associated with the object. To select an action script file:

- drop its icon on the **Object Builder** window, or
- select either **Activate Actions** or **Drop Actions** from the **Install** menu. You must then select the appropriate script from the cascading menu. You have the choice of selecting a script for double-clicking with either mouse button 1 (**Activate**) or mouse button 2 (**Alt_Activate**) and a script for dragging and dropping with either mouse button 1 (**Drop**) or mouse button 2 (**Alt_Drop**). The action script is installed in the appropriate field in the **Object Builder** window.

Once the script is installed, you can edit it by clicking on that part of the window.

See also:

- "Installing action definitions" (page 319)
- **deskcommands**(XC) manual page for more information on action scripts

Opening an existing object

You can open an existing Desktop object and modify its picture or action definitions, or you can use an existing object as a template for defining a new object.

To open an existing object:

- drop its icon on the **Object Builder** window, or
- select **Open Object** from the **File** menu and then select the object from the file selection box. The object's icon and its action scripts are installed in the **Object Builder** window.

When you drop an object's icon on the **Object Builder** window, you are prompted to specify which component(s) to load. You can choose to load the picture only, the actions only, or both.

NOTE Save the definitions for a new object by selecting **Save As** from the **File** menu. Use **Save** to save changes to the original object.

Installing action definitions

Scripts that define actions for the object are displayed in the first two full-length fields on the **Object Builder** window. You can define two types of action sequences for the object:

- those taken when a user double clicks on the object. You can define actions taken when the user double clicks with mouse button 1 (**Activate**) separately from those taken when the user double clicks with mouse button 2 (**Alt_Activate**).
- those taken when a user drags and drops the object. You can define actions taken when the object is dragged with mouse button 1 (**Drop**) or mouse button 2 (**Alt_Drop**).

You can define the actions to associate with the object in one of four ways:

- install the actions from an existing action script by dropping the file containing the action script on the **Object Builder** window
- install the actions defined in an existing object by dropping the object on the **Object Builder** window
- install default actions by dropping an executable on the **Object Builder** window or by selecting **Executable** from the **Install** menu
- enter your own actions directly in the "Trigger Action" fields

When you drop a non-executable, non-directory file icon on the **Object Builder** window, the **Object Builder** assumes it contains a script of action definitions.

If you drop an action script, existing object, or an executable on the **Object Builder** window, you must first specify the action sequence that you are defining: **Activate, Alt_Activate, Drop,** or **Alt_Drop.** Make your choice using the toggle buttons above the "Trigger Action" fields.

When you drop an object's icon on the **Object Builder** window, you are prompted to specify which component(s) to load. You can choose to load the picture only, the actions only, or both.

See also:

- "Changing an action definition" (page 318)
- "Opening an existing object" (page 318)
- "Installing an executable" (page 321)
- Chapter 25, "Writing Deskshell commands" (page 349) for information on writing action scripts

Installing a picture

You can define a picture (an icon) to associate with the object in one of two ways:

- install a picture defined in an existing pixmap or bitmap file
- drop an object or a directory icon on the **Object Builder** window to install that icon

To install any existing picture file with the suffixes *.xpm, .xbm,* or *.px,* select **Picture** from the **Install** menu. Some valid bitmap files do not include these suffixes. If you want to install such a file, make a copy of it, appending one of those suffixes, and then install it.

After you select the picture from the "Install Pictures" dialog box, you are asked to assign the file to one of four icon categories: **Normal, Activated, Small normal,** or **Small activated.** You can only assign the selected file to one of these categories at a time. Click on **OK** to complete the selection.

The installed picture is displayed in the upper part of the **Object Builder** window, above the appropriate category label.

NOTE If you change an existing picture, you do not see the change in the Desktop icon until you restart the Desktop. This is because the picture is cached in memory (to improve access speed), and the cached picture is not updated until the Desktop is restarted.

See also:

- "Step 2: Selecting an icon" (page 323) for more information on the four icon picture types

Installing an executable

You can associate an executable file with an object by dropping the executable's icon on the **Object Builder** window or by selecting **Executable** from the **Install** menu and selecting the executable file from the file selection box.

When you drop an executable icon on the **Object Builder** window, you are prompted to specify whether or not the executable is a graphical application. A graphical application is one that is designed to run in a graphical environment, such as the Desktop. A non-graphical application can be run outside the graphical environment, or from a terminal emulation window (such as **scoterm**).

- If you select **Yes**, the **Object Builder** generates default scripts for both double-click and drag-and-drop actions.
- If you select **No**, you are prompted to specify whether or not you want the user to press any key to continue after the executable has run. If the executable displays any information that needs to be left in an open window for the user to read, you should require a keystroke to continue. The application will run in a shell window.

Saving an object

To save changes to an existing object, select **Save** from the **File** menu.

To save your changes under a different object name, select **Save As**. This is the method to use when you have opened an existing object and used it as a template for building a new object. A file selection dialog box prompts you for an object name. Be sure you save the object in a directory in which you have read and write permissions. A good choice is **$HOME/***.xdt_dir/objects*.

To quit the **Object Builder**, select **Exit** from the **File** menu.

If you exit the **Object Builder** using **Cancel** after making changes, the **Object Builder** checks to make sure you really want to discard your changes.

Opening a new object

You can start with a new **Object Builder** window by selecting **New Object** from the **File** menu. This opens an unnamed object with no picture or action definitions.

Creating an object manually

To create an object on the Desktop, perform the following steps. For more information on each of these steps, see the sections immediately following this procedure.

1. Create an object directory with the desired name, followed by an *.obj* extension.
2. Select pixmaps to represent the object and copy them to the object directory, using one of the following filenames:

   picture.px
   open.px
   s_picture.px
   s_open.px

3. Decide which mouse actions or triggers are to be applied to the object.
4. For each trigger, write an object script that will perform some Desktop or system action.
5. Place each object script in the file specified by its corresponding trigger name (for example, *activate* for the **activate** trigger).

Step 1: Creating the object directory

Every object has a corresponding object directory. This directory contains all of the information that defines the object's behavior.

The object directory should be named by appending the extension *.obj* to the desired name for the object. For example, an object called "Compress" has an object directory named *Compress.obj*.

The object's title is specified in the name of the corresponding object directory. Or you can define an object's title by storing it in a file called *title*, which you should locate in a language subdirectory, ***ll_TT***, within the object directory. ***ll*** is a two-character code for the language (as defined by the ISO 639 standard) and ***TT*** is a two-character code for territory (as defined by the ISO 3166 standard). By default, *en_US* is used.

An object only appears on the Desktop or a desktop window if it is explicitly dragged onto the Desktop or a desktop window using the Deskshell command **get_out** or if the object's parent directory is open. For example, *applications/My.obj* is visible only if either the directory *applications* is open or *My.obj* has been dragged out onto the Desktop or a desktop window.

If you wish to include an object Desktop-wide, you should create it or, if it already exists, use an *Applications* or *Tools* directory and place the object directory there. The main Desktop's desktop rule file can then be configured so that this *Tools* directory appears on it. The current default directory for the applications directory is */usr/lib/X11/XDesktop3/apps* and the tools directory is */usr/lib/X11/XDesktop3/tools*.

See also:

- **deskcommands**(XC) manual page for more information on Deskshell commands

Step 2: Selecting an icon

When defining an object, first consider what its icon or pictorial representation will look like. The icon for the object can be one of many pre-defined picture files or a custom-made icon.

By default, objects can contain four pixmap files to use for an icon. These are:

picture.px	a large [64x64] image for the icon when not active. This is the only file that **must** be present.
s_picture.px	a small [32x32] image for the icon when not active
open.px	a large [64x64] image for the icon when active
s_open.px	a small [32x32] image for the icon when active

All of the pre-defined Desktop picture files are located in the directory */usr/lib/X11/IXI/XDesktop/bitmaps* and in its following sub-directories:

local_large	your own large [64x64] color icons
xdt_large	the Desktop's default large [64x64] color icons
xdt_small	the Desktop's default small [32x32] color icons
xdt_c_large	large [32x32] cursors
xdt_c_small	small [16x16] cursors

The picture files should be placed within an object directory's language subdirectory if they are language-specific, otherwise, within the object directory itself.

Picture files can be in either pixmap or bitmap format. Pixmap format is a color bitmap format defined using standard ASCII characters. The resulting files can be created and edited using a standard text editor. However it is recommended that pixmaps are edited using the **scopaint** client.

Bitmap is a monochrome format and should only be used if backward compatibility with SCO Open Desktop, Release 1.1 is required.

To create a custom-made icon for an object, the best approach is to start with an existing picture file and then edit it as follows:

1. Copy an existing picture file from one of the preceding directories or the blank picture file *blank.px* into the object directory and rename it appropriately, as described above.
2. Change to the object directory. At the command line, type **scopaint** ***filename*** to execute the **scopaint** client. Edit the picture as desired.
3. When you are satisfied with the picture, exit from **scopaint**, selecting **Save** or **Save As** from the **File** menu.

Step 3: Selecting trigger actions

The action of pointing at an icon with the mouse pointer and performing an action such as double-clicking is referred to as "triggering" the icon. Triggers are used to execute different Desktop and/or system actions and tasks. When an object is triggered, it performs certain actions on the Desktop or the system, depending the trigger type.

A trigger is either a click, a sequence of clicks only, or a sequence of clicks followed by a hold or drag. You can trigger icons on the Desktop in three ways: static triggers, hold triggers and drag triggers.

Clicking twice in succession without moving the mouse pointer is called either double-clicking, activating the icon, or a "static" trigger. Dragging one or more icons and dropping them on to another icon is called a "drag and drop" or a "dynamic" trigger. Pressing a mouse button and holding it down while the pointer is over an object is referred to as a "hold" trigger.

The notation for the different triggers is based on the trigger type and the mouse button used. The following are the pre-defined Desktop triggers:

- **Static triggers:**

select	single-click on an icon picture
alt_select	single-click on an icon picture with mouse button 2
rename	single-click on an icon title

alt_rename	single-click on an icon title with mouse button 2
activate	double-click on an icon
alt_activate	double-click on an icon with mouse button 2
deselect	single-click on a window background
report	double-click on a window background
alt_report	double-click on window background with mouse button 2
s*	matches any static trigger

- **Dynamic triggers:**

drop	drop one or more icons on an icon or window background
alt_drop	drop one or more icons on an icon or window background with mouse button 2
d*	matches any drag trigger

- **Hold triggers:**

menu	hold mouse button 3 on an icon picture or title
popup_menu	hold mouse button 3 on a window background
h*	matches any hold trigger

To take advantage of triggers, first determine how many different actions you want the object to perform, then assign each action to a unique trigger type.

See also:

- Chapter 26, "Mapping mouse triggers for the Desktop" (page 371) for information on mapping your own triggers or changing the existing ones

Step 4: Writing trigger scripts

After the triggers to be applied to the objects are determined, the Desktop or operating system actions and tasks that are to occur after each trigger action must be defined.

The Deskshell command language is used to to define the actions that occur when the icon is triggered. Deskshell scripts written explicitly for objects are referred to as object scripts.

Deskshell is a command language, complete with a flexible range of control structures and a wide range of commands. You can specify regular UNIX operating system commands within Deskshell scripts. However, it is recommended that you learn how to use the Deskshell commands and scripts.

Because Deskshell is designed specifically for use with the Desktop and because Deskshell commands execute significantly faster than regular UNIX shell commands, Deskshell commands are recommended for coding your script.

See also:

- Chapter 25, "Writing Deskshell commands" (page 349)

Step 5: Naming trigger scripts

The final step involves naming the object scripts and placing them in the proper location.

After an object script is written in Deskshell, it must be placed in a file. Filenames correspond directly to the trigger names (i.e., *activate* for an **activate** trigger and *drop* for a **drop** trigger).

NOTE You should surround the commands in trigger files with **begin** and **finish**, and not specify them as **trigger_action** clauses.

Trigger files should be placed directly in the object directory, **not** within any language subdirectory.

Any text that you intend to display to the user should be represented by variable names, and the definitions for these variables placed in the *text* file within the object directory's language subdirectory. For example:

```
msg1='You have activated this object'
msg2='You have dropped an icon on this object'
```

See also:

- **xdt3**(XC) manual page
- **deskshell**(XC) manual page
- **deskcommands**(XC) manual page

Chapter 21

Configuring icons

The desktop represents files and directories in the UNIX filing system by pictorial "icons", which provide the user with a convenient way of manipulating files and give additional information about the types of files and their access permissions.

In this chapter you will learn how to:

- define the appearance of icons (this page)
- configure the behavior of icons (page 330)

See also:

- Chapter 20, "Creating objects for the Desktop" (page 315) for information on a simple and flexible way to configure icons, using objects

Defining the appearance of icons

To alter the appearance of an icon or number of icons, use an **icon_rules** clause matching the appropriate icons.

See also:

- "Defining rules for icons" (page 328)
- "Defining a picture for icons" (page 329)
- "Defining a title for icons" (page 330)

Defining rules for icons

The **icon_rules** clauses use the following form:

```
icon_rules
{
   filespec
   {
      picture=filename;
      title=name;
   }
}
```

The **icon_rules** keyword introduces the rules, and ***filespec*** specifies the icons to which the rules should apply. ***filespec*** also specifies the filenames to be matched, and the classes of files to be matched.

For each group of files specified by ***filespec***, you can provide a **picture** clause, a **title** clause, or both, specifying what the title or icon should be for those files.

Note that a single **icon_rules** clause can include several ***filespec*** sections, to provide pictures and/or titles for different groups of icons:

```
icon_rules
{
   filespec1
   {
      picture=filename1;
      title=name1;
   }
   filespec2
   {
      picture=filename2;
      title=name2;
   }
   ...
}
```

If an icon matches more than one ***filespec***, the title is determined by the first matching clause containing a **title**, and the picture is determined by the first matching clause containing a **picture**.

See also:

- "Defining a picture for icons" (this page)
- "Defining a title for icons" (page 330)
- "Defining the scope of rules" (page 286)
- "Classes" (page 291)
- **icon_rules** in the **xdt3**(XC) manual page

Defining a picture for icons

The **picture** clause specifies the name of the bitmap or pixmap file to be used for the icons of all files in the specified group. It has the format:

```
picture=filename;
```

If ***filename*** begins with " / ", it is taken as an absolute pathname. Otherwise, the Desktop searches the sequence of picture file directories, specified by the **pictureDirectory** resource. By default, the Desktop searches for pictures in the following order:

- */usr/lib/X11/IXI/XDesktop/bitmaps/xdt_c_large*
- **$HOME**/*.xdt_dir/bitmaps/xdt_large*
- */usr/lib/X11/IXI/XDesktop/bitmaps/xdt_large*

Picture files are provided in the following subdirectories of */usr/lib/X11/IXI/XDesktop/bitmaps*:

local_large	your own large [64x64] color icons
xdt_large	the Desktop's default large [64x64] color icons
xdt_small	the Desktop's default small [32x32] color icons
xdt_c_large	large [32x32] cursors
xdt_c_small	small [16x16] cursors

See also:

- **picture** in the **xdt3**(XC) manual page

Defining a title for icons

The **title** clause gives the title for the specified group of files. It has the format:

```
title=name;
```

The title is taken as the text from the "=" to the ";" characters, including any spaces.

The title can include the following special sequences, to substitute the corresponding string into the title:

%B0 basename of the file

%C0 class of the file, given as six characters in standard order

%D0 absolute pathname of the directory (page 298) holding the file

%E0 as %B0 but extensionless — with the last dot and any characters following removed

%P0 absolute pathname of the file

%R0 relative pathname of the file, which within a directory window will be the same as %B0

See also:

- "Referring to file and directory names" (page 298)
- "Classes" (page 291)
- **title** in the **xdt3**(XC) manual page

Defining the behavior of icons

The trigger table defines which triggers correspond to which actions by the user. Some other triggers are also used for higher-level functions such as providing help on an icon.

To define the behavior of an icon when it is triggered in a particular way, include a **trigger_action** clause for each trigger that should be understood by that icon:

```
trigger_action: trigger
```

See also:

- "Writing trigger rules" (page 331)
- Chapter 19, "Defining Desktop triggers" (page 309)

Writing trigger rules

The **trigger_action** clauses are contained within **icon_rules** clauses, just like the **title** and **picture** clauses described in "Defining a picture for icons" (page 329) and "Defining a title for icons" (page 330). They take the form:

```
trigger_action: trigger { script }
```

script specifies the Deskshell script that will be run when *trigger* is applied to an icon matching *filespec* in the **icon_rules** clause. *trigger* can specify the name of one of the triggers defined in the trigger table, or:

- **s*** to indicate any static trigger
- **d*** to indicate any drag trigger
- **h*** to indicate any hold trigger
- * to indicate any trigger

Alternatively, *trigger* can specify the name of any trigger that you want to define for that icon.

See also:

- **trigger_action**, **icon_rules**, **s***, **d***, **h***, and * in the **xdt3**(XC) manual page

Chapter 22

Configuring Desktop windows

In the default configuration of the Desktop, users can place icons on the main Desktop or a desktop window by dragging the file icons into the appropriate desktop.

This chapter describes how you can set up your own Desktop rules to tailor:

- the behavior of desktops (this page)
- the appearance of desktops (page 334)

Defining the behavior of desktop windows

To define the behavior that results from the following action:

Single-click on the desktop window background	include a **drop_in_action: deselect** clause in an **icon_rules** clause applying to the desktop
Drag one or more icons into the desktop window	include a **drop_in_action: drop** clause in an **icon_rules** clause applying to the desktop
Double-click on the background of the desktop window	include a **drop_in_action: report** clause in an **icon_rules** clause applying to the desktop
Open or close the desktop window	include an **initial_actions** or **final_actions** clause in the desktop rule file.

See also:

- "Example" (this page)
- "Changing the behavior of a desktop" (page 289)
- "Changing desktop, directory, dialog box, and icon behavior" (page 37)
- **deselect, drop, final_actions, icon_rules, initial_actions,** and **report** in the **xdt3**(XC) manual page
- **drop_in_action** in the **deskcommands**(XC) manual page

Defining the appearance of desktop windows

To define:

The position of icons on the desktop	include a **desktop_layout** clause in the desktop rule file
The icons that are locked onto the desktop window	include a **locked_on_desktop** clause in the desktop rule file

See also:

- "Example" (this page)
- "Changing the behavior of a desktop" (page 289)
- **desktop_layout** and **locked_on_desktop** in the **xdt3**(XC) manual page

Example

This example shows a rule which defines a desktop window with the following characteristics:

- single-clicking on the desktop window background displays an information dialog box
- double-clicking on the desktop window background closes the desktop window
- dropping one or more icons onto the desktop window puts them on the desktop
- opening the desktop window displays an information dialog box
- the main Desktop icon is locked onto the desktop

The rules defining these characteristics are provided in the following desktop rule file, which should be called **demo.dt**:

```
%/dt/
desktop_layout
{
   %$HOME$/.ixi/xdt/Main.dt @GO,0;
}
locked_on_desktop
{
   %$HOME$/.ixi/xdt/Main.dt;
}
initial_actions
{
   for_info 'Welcome to the demo desktop'
}
icon_rules
{
   demo.dt /F
   {
      drop_in_action: deselect
      {
         for_info You have clicked on the background of the demo desktop.
      }
      drop_in_action: report
      {
         close_desktop $static_arg
      }
      drop_in_action: drop
      {
         get_out -l $dynamic_args -d $static_arg
      }
   }
}
```


See also:

- "Defining the scope of rules" (page 286) for information on the rule file in which to insert the rules clauses to achieve the desired effect

Chapter 23

Configuring directory windows

In the default configuration of the Desktop, users can move and copy files between directories simply by dragging the file icons into the appropriate directory window.

This chapter:

- describes how you can set up your own local rules to tailor the behavior of directories (this page) on your system, and
- provides an example of configuring directory behavior (page 338)

Defining the behavior of directory windows

To define the behavior that results from the following action:

Single-click on the directory window background	include a **drop_in_action: deselect** clause in an **icon_rules** clause applying to the directory
Drag one or more icons into the directory window	include a **drop_in_action: drop** clause in an **icon_rules** clause applying to the directory
Double-click the background of the directory window	include a **drop_in_action: report** clause in an **icon_rules** clause applying to the directory
Open or close the directory window	include an **initial_actions** or **final_actions** clause in the local rule file

See also:

- "Example" (this page)
- "Changing the behavior of a directory" (page 289)
- **drop_in_action** in the **deskcommands**(XC) manual page
- **deselect, drop, final_actions, icon_rules, initial_actions,** and **report** in the **xdt3**(XC) manual page

Example

This example shows a rule that defines a directory with the following characteristics:

- single-clicking on the directory window background displays an information dialog box
- double-clicking on the directory window background closes the window and opens the parent directory window
- dropping one or more icons onto the directory window copies them into the directory
- closing the directory window displays a dialog box

The rules defining these characteristics are provided below:

```
icon_rules
{
   TestDir /D
   {
      drop_in_action: deselect
      {
         for_info You have clicked on the TestDir directory background.
      }
      drop_in_action: report
      {
         display_directory $static_arg '(dirname $static_arg)
      }
      drop_in_action: drop
      {
         copy_into $static_arg $dynamic_args
      }
   }
}
```

You would also place the following rules in *TestDir/.xdtdir/ll_TT* (where, by default, *ll_TT* is *en_US*):

```
final_actions
{
   for_info Closing the TestDir directory.
}
```

See also:

- "Defining the scope of rules" (page 286) for information on the rule file in which to insert the rule clauses to achieve the desired effect

Chapter 24

Configuring Desktop menus

The Desktop supports both pull-down menus, which drop down from a menu name in a menu bar, and pop-up menus, which are usually displayed beneath the mouse pointer.

For each type of menu, the menu commands can include mnemonics and accelerator keys, and also display further cascade menus.

Each type of menu is defined using an identical syntax. However, the names of the desktop, directory and treeview menu bars are defined in the Desktop resource file.

This chapter describes how to:

- provide additional commands for users on the standard menus (page 342)
- define mnemonics and accelerator keys (page 344)
- provide additional pull-down (page 345) or pop-up menus (page 346)
- disable menu commands (page 347)
- remove menus (page 348)

See also:

- Appendix B, "Desktop resources" (page 403)
- Chapter 13, "Customizing window manager menus" (page 235)

Defining menus

To define the following behavior:

Command available on a menu	include a **menu** clause defining the menu
Action when the user chooses a menu command	include a **menu_item** clause defining the command
Cascade menu when the user chooses a menu command	include a **pull_off_menu** clause in the **menu_item** definition
Dividing line between commands on a menu	include a **dividing_line** or **thick_dividing_line** clause in the **menu** definition
Command that is disabled if not applicable	include a **enable_if** clause in the **menu_item** clause for the command

See also:

- "Menu clauses and commands" (page 343)
- "Mnemonics and accelerator keys" (page 344)
- "Pull-down menus" (page 345)
- "Pop-up menus" (page 346)
- "Disabling menu commands" (page 347)
- "Removing menus" (page 348)
- "Defining the scope of rules" (page 286)

- **menu, menu_item, pull_off_menu, dividing_line,** and **thick_dividing_line** in the **xdt3**(XC) manual page

Menu clauses and commands

The commands on a menu, and the action performed when any command is chosen, are determined by the **menu** and **menu_item** clauses. These have the format:

```
menu: menuname
{
   menu_item clause
   ...
   menu_item clause
}
```

NOTE The text your provide for *menuname* is an internal reference, and does not appear as the title of the menu. To give a title to a menu, do not specify an action for the first menu command. In this case the command will automatically be centered rather than left-aligned.

Each **menu_item** clause contains a **title** clause specifying the name of the command, and a **select_action** clause specifying the action to occur when the command is chosen:

```
menu_item
{
   title=cmdname;
   select_action { script }
}
```

Here *cmdname* gives the name of the menu command, and *script* is the action to be performed if it is chosen.

Instead of a script, the **menu_item** clause can reference a **pull_off_menu** clause, in which case choosing it displays the cascade menu of that name:

```
menu_item
{
   title=cmdname;
   pull_off_menu=name;
}
```

For example, the **Sort** cascade menu would be defined by the following clause:

```
menu_item
{
   title=Sort;
   pull_off_menu=Sort_Cascade;
}
```

The **menu_item** clause also has a short form, as follows:

```
menu_item: cmdname
{
   script
}
```

This is equivalent to:

```
menu_item
{
   title=cmdname;
   select_action
   {
      script
   }
}
```

See also:

- **menu, menu_item, pull_off_menu, select_action,** and **title** in the **xdt3**(XC) manual page

Mnemonics and accelerator keys

Mnemonics and accelerator keys let you choose a menu command without using the mouse.

The *cmdname* assigned to a **menu_item** clause can end in a string of the form:

_m_key_keytext

The single character *m* defines a mnemonic for the menu command. The character *m* must occur in the name of the command, and the first occurrence of the character in the name will be shown underlined on the menu. No two commands should have the same mnemonic in a single menu.

The menus on a menu bar can also have mnemonics. To choose a menu command using its mnemonic, press ⟨Alt⟩ and the mnemonic for the menu, then the mnemonic for the menu command itself. For example, if the **File** menu has mnemonic **F** and the **Open** menu command has mnemonic **O**, you would press ⟨Alt⟩ then **O**.

The string *key* defines an accelerator key for the menu command, using the standard OSF/Motif syntax. For example, `Ctrl<key>F` represents the accelerator key ⟨Ctrl⟩**F**.

The string ***key*** is an internal way of representing a key press, and so is not displayed. Instead, the string ***keytext*** will be right-justified in the menu.

```
menu_item: New File_F_Ctrl<key>F_Ctrl+F
```

To choose a menu command you can press the accelerator key combination directly. No two commands available from the same menu bar should have the same accelerator key.

Pull-down menus

The Desktop resource file defines three special menu names: **DesktopMenuBar, DirMenuBar,** and **TreeMenuBar.** Menu rules with these names define the menu bars for desktop windows, directory windows and treeview windows, respectively.

Each **menu_item** clause defines the name of one of the pull-down menus on the menu bar, and should contain a **pull_off_menu** clause to define the commands on that pull-down menu.

By default, the Desktop provides menu bars, in desktop windows, directory windows and treeview windows, and these menu bars provide pull-down menus for all of the most frequently-used commands that users need to perform within the Desktop.

The commands on these menus, and the actions they perform, are defined by the rules for your user type. You do not need to change this file to configure your own menus.

See also:

- "Pop-up menus" (page 346)
- "Changing the behavior for different types of user" (page 288)
- **menu_item** and **pull_off_menu** in the **xdt3**(XC) manual page

Pop-up menus

Pop-up menus are defined using the same syntax as pull-down menus, and are displayed by the **popup** command.

For example, a simple pop-up **FooBar** menu could be defined as follows:

```
menu: foobar_menu
{
   menu_item: FooBar {} %// menu title
   dividing_line;

   menu_item: Foo
   {
      fyi Foo
   }
   menu_item: Bar
   {
      fyi Bar
   }
}
```

The accompanying **icon_rules** clause would resemble:

```
icon_rules
{
   foobar /F
   {
      title=FooBar;
      trigger_action: menu
      {
         popup foobar_menu
      }
   }
}
```

In the above example, **foobar_menu** refers to the **menu** clause of that name already defined. The **FooBar** menu would appear when you hold down mouse button 3 over the **FooBar** icon, which represents a file called *foobar*.

See also:

- "Pull-down menus" (page 345)
- **icon_rules** and **popup** in the **xdt3**(XC) manual page

Disabling menu commands

A menu command can be disabled, to indicate to the user that it is not applicable in the current situation.

It is better to disable an inapplicable menu command than to allow the user to choose it and then display an error dialog box.

To disable a menu command, include an **enable_if** clause in the **menu_item** clause defining the command. This has the form:

```
enable_if { script }
```

In this example, ***script*** is a script that is executed when the menu is displayed. If it returns a "true" status the command is enabled; otherwise it is disabled.

For example, the following menu command is only enabled if at least one icon has been selected:

```
menu: DesktopMyMenu
{
  menu_item
  {
    title=MyCommand;
    enable_if
    {
      sels=`(query selections `(query thread_info -i $thread_name(2)))
      -gt $#sels 0
    }
    select_action
    {
      ...
    }
  }
}
```

See also:

- **enable_if** and **menu_item** in the **xdt3**(XC) manual page

Removing menus

To turn off a menu, execute a **dynamic_rule** command with an empty menu clause for that menu.

See also:

- **dynamic_rule** in the **xdt3**(XC) manual page

Chapter 25
Writing Deskshell commands

This chapter describes the "Deskshell script language" that you use to describe the actions you want to perform in rules. It assumes some familiarity with shell programming.

Deskshell is a general-purpose language, with control constructs, constants and variables, and built-in commands to perform standard actions. Deskshell allows you to write powerful Desktop rules that will execute totally within the Desktop, without the need to start up a separate shell. Using Deskshell, operations such as **gti** and **yni** can be incorporated into rules without performance penalties, and with the added benefit that Desktop rules incorporating Deskshell commands are completely portable across different UNIX installations, since they avoid the ambiguities of the Bourne shell.

Specifically, this chapter discusses:

- Deskshell syntax (page 350)
- Deskshell operators (page 354)
- Deskshell control constructs (page 360)
- Deskshell function definitions (page 360)
- how Deskshell commands are executed (page 361)

See also:

- Appendix C, "Deskshell command summary" (page 415)
- **deskshell**(XC) manual page
- **deskcommands**(XC) manual page

Deskshell syntax

The syntax of Deskshell is similar to the standard Bourne shell language, but with a simplified syntax and more consistent semantics. In particular:

- the use of lists, instead of strings, for variable values avoids the need to treat strings containing spaces in a special way
- substituted values are never re-interpreted, so special characters can be included in variable values without problems
- there is only one string quoting character, simplifying the syntax

See also:

- "Quoting strings" (this page)
- "Comments" (page 351)
- "Wildcards" (page 351)
- "Using variables" (page 352)
- "Variable substitutions" (page 352)
- "Subsets" (page 353)
- "Function arguments" (page 353)
- "Initialization" (page 354)

Quoting strings

Strings that contain special characters or spaces should be quoted with the single-quote character. For example, to display the text "Press return", you could use the command:

```
echo 'Press return'
```

To include a single quote in quoted text it should be repeated, as in:

```
echo 'Don''t press return'
```

No substitution takes place within quoted strings.

The following characters are special, and cannot be included in a string unless they are quoted:

space, tab, or newline

| bar

' single quote

& ampersand

() parentheses

$ dollar sign

{} braces (curly brackets)

' backquote

< less than

> greater than

^ circumflex

: colon

; semicolon

\ backslash

% percent

= equals

hash

The following characters are interpreted as wildcards unless they are in a quoted string:

* asterisk

[open bracket

? question mark

Comments

Within a script, comments may be included by preceding them with **%//**. All characters to the end of the line are ignored. For backward compatibility with previous versions of the Desktop, the " # " character can also be used, but *only* within Deskshell scripts.

Wildcards

When a command is executed, any argument containing a wildcard is expanded into a list of filenames matching the wildcard, or an empty list if none match. The wildcard characters have the following meanings:

* any string except " / " or leading dot

? any single character except " / " or leading dot

[*chars*] any one of the characters in ***chars***, so for example **[xyz]** matches either *x*, *y* or *z*. The " / " symbol must not appear in the brackets.

[!*chars*] any single character except those in ***chars***, " / " and leading dot. For example **[!xyz]** matches any character other than *x*, *y*, *z*, / and leading dot.

Using variables

In rule files, you can use variables to keep track of numbers and text strings. A variable name can consist of any sequence of letters, digits, and, underscores provided the first character is not a digit.

NOTE Variable names beginning with two underscores should be avoided, as they are used in the standard rules. However, you can use names beginning with one underscore and then a letter or digit.

Variables do not have to be specially defined, and you can give them a value using an equals sign. For example, the following might specify a counter value for the subsequent repetition of a command:

```
count=10
```

NOTE There must *not* be any space on either side of the = sign.

For maximum flexibility, variables can be set to a list of values. The list is specified by putting all the elements of the list in brackets. So, for example, the following could be used to make the variable **editors** equal to a list of the filenames of all the editors on the system:

```
editors=(vi xedit ed)
```

Variables are local to an executing script unless prefixed with a ":".

Variable substitutions

The value of a variable can be substituted into a script with:

```
$name
```

So, for example, **$editors** will have the value:

```
vi xedit ed
```

You can find the number of elements in a list with **variable**. For example:

```
elements=$#editors
```

sets **elements** to "3".

Note that:

```
var=()
```

sets **var** to a list with no elements, so **$#var** is "0", whereas:

```
var=''
```

sets **var** to an empty string, so **$#var** is "1".

Subsets

You can extract specific elements from a list by putting one or more element numbers after the variable name in brackets. Using the example in "Variable substitutions" (page 352):

```
$editors(2)
```

will have the value:

```
xedit
```

NOTE There must *not* be any spaces between the variable name and the opening bracket.

More than one subscript can be given. For example, the following:

```
days=(mon tue wed thur fri sat sun)
echo $days(3 1 3)
```

will produce:

```
wed mon wed
```

Subscripts that are out of range are ignored, so the same result is obtained with:

```
echo $days(3 0 1 8 3)
```

The element numbers may come from another variable or any other construct, so:

```
$days($#days)
```

is permitted and has the value

```
sun
```

Function arguments

The special variable "*" holds the list of arguments to the current function. For example, in a **drop_in_action: drop**" script, the "*" variable will contain a list of the filenames of the icons that were dropped into the window.

The number of icons dropped is given by **$#***. Each of the individual elements in "*" can be obtained with **$*(1)**, **$*(2)**, and so forth. For convenience, these can be abbreviated to **$1**, **$2**, and so forth.

Initialization

When the desktop starts, the value of each UNIX environment variable is copied into the Deskshell variable of the same name. These variables start with a list of one string.

The variable **path** is set from the environment variable **PATH**, by splitting its value at each colon. Thus, if the environment variable **PATH** has the value ".:/bin" then **path** will be set to the two strings "." and "/bin".

All other variables are set to contain no strings.

Operators

The following operators are provided in Deskshell:

=	assignment
<, > and >>	redirection
' (backquote)	substitution
^ (circumflex)	concatenation
|	pipeline
&& and ||	conditionals
& and ;	termination
::	list mark

NOTE The operator precedence determines in which order multiple commands are evaluated.

Spaces may be included on either side of all of the above operators apart from "=" and "^", which *must not* have spaces on either side of them.

See also:

- "Assignment" (page 355)
- "Redirections" (page 355)
- "Command substitution" (page 356)
- "List substitution" (page 356)
- "Concatenation" (page 357)

- "Command terminators" (page 357)
- "Pipelines" (page 358)
- "List mark" (page 358)
- "Conditionals" (page 359)
- **deskshell**(XC) manual page for information on operator precedence

Assignment

The " = " operator assigns a value to a variable. For example:

```
count=10
```

assigns "10" to **count**, and

```
fib=(1 1 2 3 5)
```

assigns the list "(1 1 2 3 5)" to **fib**.

See also:

- " = " in the **deskshell**(XC) manual page

Redirections

A redirection causes a UNIX file descriptor to be redirected to a different file. The following options are available:

< ***filename***	the file is opened on descriptor 0, standard input, for reading only; it must already exist
> ***filename***	the file is opened on descriptor 1, standard output, for writing, and is truncated; it is created if necessary
>> ***filename***	the file is opened on descriptor 1, standard output, for append only; it is created if necessary

The following variants are available for each redirection. In each case " < " can be replaced by " > " or " >> ".

< ***filename***	read from ***filename***
<[***number***] ***filename***	read from ***filename*** on descriptor ***number***
<[***new=number***]	make ***new*** a duplicate of ***number***
<[***number=***]	close descriptor ***number***

For example:

```
for_info < $static_arg
```

displays the text from file **$static_arg** in a dialog box, and:

```
gti 'Enter name:' > name
```

saves the user's name in a file *name*.

See also:

- **$static_arg** in the **deskshell**(XC) manual page

Command substitution

Many Deskshell and UNIX commands perform an action and print a result. You can trap the output of such a command, and use it in a Deskshell script, with the **'{...}** construction.

The text output from the command is split into a list of strings at the character specified in **$ifs(1)**, or the characters space, tab or newline if **ifs** is unset.

For example:

```
size='{ls -s $1}
```

assigns the string representing the size and name of the file *$1* to the variable **size**.

See also:

- **'{...}** and **$ifs(1)** in the **deskshell**(XC) manual page

List substitution

Certain Deskshell commands generate text output. These may be used directly, without requiring a separate process, using the form:

```
var='(basename $list)
```

which sets **var** to a list of the basenames of the files in **$list** without requiring another process to be run.

Functions can also generate text output for direct use using:

```
'(myfunction arg1)
```

or

```
'myfunction
```

Command and list substitutions may be nested to any depth.

See also:

- "Processing filenames in rules" (page 298) for information on basenames

Concatenation

Two words or lists can be concatenated, or joined together, using the "^" (circumflex or caret) character.

Lists can only be concatenated if they both contain the same number of elements, or if one of them only contains one element or is empty, as illustrated by the following examples:

a	b	a^b
w	x	wx
w	(x y z)	(wx wy wz)
(w x y)	(a b c)	(wa xb yc)
(w x y)	( )	(w x y)
(w x y)	(a b)	***illegal***

NOTE There should *not* be any spaces around the "^" character.

Several concatenations can be included in one expression, as in:

```
s.^(in proc out)^.main.^(c h s)
```

which evaluates to:

```
(s.in.main.c s.proc.main.h s.out.main.s)
```

Deskshell allows circumflexes to be omitted when the context makes it unambiguous. For example, **$file^.c** can be written as **$file.c** instead.

See also:

- "^" in the **deskshell**(XC) manual page

Command terminators

Each command in a script is terminated by " ;" or " &". If the command is terminated with " &", it is run in a separate thread in the background. Otherwise it is run in the current thread.

The " ; " can be omitted after the last command on a line.

See also:

- "Threads" (page 362)
- " ; " and " & " in the **deskshell**(XC) manual page

Pipelines

Two commands can be linked by a pipeline using the " | " operator. The two commands will be executed in separate new threads, and the output of the first command will become the input to the second command.

When a pipeline is executed, Deskshell waits for all the "children" (commands in the pipeline) to terminate. The individual statuses are converted to strings, and these are all stored in the variable **status**, in the same order as the commands in the pipeline. For example, the pipeline **true | false** generates the status "(0 1)".

The " | " can be followed by a construct to specify the output and input file descriptors to be used, as in:

| [***output*=*input***]

In this case the brackets are part of the operator. If ***=input*** is omitted it defaults to "0".

See also:

- "Threads" (page 362)
- " | " and **status** in the **deskshell**(XC) manual page

List mark

Deskshell commands which take lists as arguments, for example **list intersect** and **list count**, use the list mark " :: " to separate each argument.

For example:

```
list count a b c :: d e f g
```

gives the result:

```
( 3 4 )
```

If the list mark " :: " was not included, for example in the form:

```
list count $list1 $list2
```

the result would be a single value, as the values of the two variables would combine to become a single list.

See also:

- **list count** and **list intersect** in the **deskcommands**(XC) manual page

Conditionals

The " && " separator executes a command only if the previous command returned a "true" status.

The " | | " separator executes a command only if the previous command returned a "false" status. Thus:

This example:	Can also be written:
a | | b	if a; else b; fi
a && b	if a; then b; fi
a && b | | c	if if a; then b; fi else c; fi

After any of the separators " & ", " | ", " | | ", or " && ", a newline is ignored. Thus the last example above can be written:

```
a &&
b ||
c
```

The operators " | ", " & ", " ; ", " :: ", " && ", and " | | " can also be preceded or followed by spaces and tabs.

See also:

- " && " and " | | " in the **deskshell**(XC) manual page

Control constructs

Deskshell provides the following control constructs, which perform similar functions to the corresponding commands in other languages such as C or Pascal:

for	executes a script for each value of a list
while	executes a script while a condition is true
until	executes a script until a condition becomes true
if	executes a script depending on the value of a condition
case	executes a script depending on the value of a variable

See also:

- **deskshell**(XC) manual page

Function definitions

Functions can be defined to perform frequently-needed sequences of commands. The syntax is:

function ***name*** { ***script*** }

which assigns ***name*** as the name of ***script***.

The script is not evaluated at this point, though it will be parsed and checked for syntax errors.

The assignment can be canceled with:

```
function name { }
```

The list of arguments to the function is passed to the function in the variable "*".

A function can return a result, which can be used with the '( ) syntax. For example:

```
function makeday
{
    return $*^'day'
}
```

defines a function to append **'day'** to each of its arguments. Thus:

```
A=Sun
B='(makeday $A)
```

sets **$B** to '"Sunday"'.

Status

When a command is executed, it generates a status (a numerical value between 0 and 1023), indicating the results of executing the command. That status is converted to a string and stored in the variable **status**. When a pipeline is executed, the individual statuses are collected and the strings all stored in the variable **status**, in the same order as the commands in the pipeline.

A status value of zero is taken to mean "true", and a non-zero value to mean "false".

The value of **status** is used implicitly by the following commands:

- if ... then ... else ... fi
- while ... do ... done
- until ... do ... done
- " && "
- " || "

For example, you can test whether a user pressed **Cancel** in a **gti** dialog box as follows:

```
file=`(gti 'Enter filename:')
if -ne $status 0; then exit; fi
```

See also:

- **status** and " && " in the **deskshell**(XC) manual page
- **gti** in the **deskcommands**(XC) manual page

How Deskshell commands are executed

This section describes how Desktop commands are executed within threads, to enable different actions to be processed concurrently. It explains how each thread inherits its context from the thread that invokes it, and explains how you can pass values between threads using global variables.

See also:

- "Threads" (page 362)
- "The state of threads" (page 362)
- "Local variables" (page 363)
- "Global variables" (page 364)

Threads

A "thread" is an instance of a Deskshell script being executed. When the Desktop is running, there is always at least one thread being executed — the "system thread".

A new thread is created in the following situations:

- for each desktop window
- for each directory window
- for each Treeview window
- when the user triggers an icon
- when a script executes a command in the background, using the "&" operator
- when a script performs an **actions_of**, **drop_in_actions_of**, or **menu_actions_of** command
- when a pipeline is executed

See also:

- **actions_of**, **drop_in_actions_of**, and **menu_actions_of** in the **deskcommands**(XC) manual page

The state of threads

Threads can be in one of three different states:

executing	commands within the thread are being executed in sequence
suspended	system and window threads can be suspended until a specified condition is satisfied, or the thread receives a signal terminating it

waiting — the thread is waiting for a program to run. While waiting, it cannot receive any signals until the thread is unblocked. A thread is also blocked by a **sleep** command, by internal commands such as **for_info** and **gti** that wait for the user, and by running a pipeline.

See also:

- **for_info**, **gti**, and **sleep** in the **deskcommands**(XC) manual page

Local variables

Each thread can use local variables, which are distinct from the local variables in other threads. For example, consider the following **icon_rules** clause:

```
icon_rules
{
   demo /F
   {
      trigger_action: activate
      {
         xs=x
         until == $xs xxxxxx
         do
            for_info 'In activate: '^$xs
            xs=x^$xs
         done
      }
      trigger_action: alt_activate
      {
         xs=x
         until == $xs xxxxxx
         do
            for_info 'In alt_activate: '^$xs
            xs=x^$xs
         done
      }
   }
}
```

If the user double-clicks on the **demo** icon with mouse button 1 and then again with mouse button 2, two threads will be created to execute the commands in the two **trigger_action** clauses. In whatever order the user presses **OK** on each dialog, the two instances of the local variable **xs** are kept separate, and changes to its value in one thread do not affect its value in the other thread.

See also:

- "Global variables" (this page)
- **icon_rules** and **trigger_action** in the **xdt3**(XC) manual page

Global variables

Global variables can be accessed by all threads in a Desktop session. They are created by prefixing the variable name with " : ".

For example:

```
:count=10
```

creates a global variable **count** with the value "10".

Global variables can be accessed with the constructs ***$:var***, ***$#:var***, and so forth. Global variables are distinct from local variables with the same name, so you can have:

```
count=12
```

and

```
:count=10
```

as separate variables. However, ***$var*** will give the value of the global variable ***var*** if no local variable has been defined.

See also:

- "Local variables" (page 363)

Variable overriding

A local variable can be given a value for the duration of one command by prefixing the command with the variable assignment. For example:

```
var=2
var=1 for_info 'var=' $var
for_info 'var=' $var
```

will first display the value of ***var*** as "1", and then as "2".

If the command is a function call, then the new value applies for the duration of the function call, after which the variable reverts to its previous value. It makes no difference whether or not the function also alters the value.

How environments are inherited

The parent of each thread, (i.e. the thread that created it), defines the environment that it inherits. The environment consists of the local variables and functions of the parent thread, and the name of the current directory. The new thread can also create its own set of local variables and functions, and its own current directory.

Each thread has a name, and when it is created, its name is inserted at the front of the local variable **thread_name**. Thus, within any thread, **$thread_name(1)** is the name of the thread, **$thread_name(2)** the name of the parent thread, and so forth. The name of a thread can be queried using **query thread_info**.

See also:

- **thread_name** in the **deskshell**(XC) manual page
- **query thread_info** in the **deskcommands**(XC) manual page

System thread

When the Desktop is first run, the system thread is created. The name of this thread is the empty string.

The system thread executes the following sequence of commands:

- initial actions in system rule file
- initial actions in user rule file
- suspend
- final actions in user rule file
- final actions in system rule files

The initial and final actions of dynamic rules are run by the system thread, as if they were signals.

The system thread stays in the suspended state until the Desktop shuts down, using the **die** command.

See also:

- "Signals" (page 368)
- **die** in the **deskcommands**(XC) manual page

Window threads

Each desktop, directory or treeview window has an associated thread called the "window thread". These are always children of the system thread. Window thread names end with the name of the desktop or directory window. To get the window name, use the **query thread_info** command.

Each window thread executes the following sequence of commands:

- initial actions in directory or desktop rule file
- suspend
- final actions in directory or desktop rule file

The window thread is suspended until the window is closed.

All other threads are children of the thread that created them. For example, double-clicking on an icon in a desktop window creates a new thread which is a child of the desktop window thread.

See also:

- **query thread_info** in the **deskcommands**(XC) manual page

Background threads

A thread can also be created by running a command in the background with the " & " operator. In this case the thread is a child of the thread that created it.

For example, the following illustration shows the inheritance tree for a thread created when the user double-clicks on a **clock** icon in the home directory window with the following **icon_rules** clause:

```
icon_rules
{
   clock
   {
      trigger_action: activate
      {
         for_info hello &
         xclock
      }
   }
}
```

Thread **T1** will create the new thread **T2**. This will display the **for_info** dialog box. Thread **T1** will then run **xclock**, and will stay blocked until the **xclock** window is closed. Thread **T1** will then terminate.

Thread **T2** will block until the user closes the **for_info** dialog by clicking on the **OK** button.

Several commands may be run in the background by enclosing them in "{ }" and putting " & " after the closing bracket, as in:

```
{ string='Hello'
  for_info $string
} &
```

In the parent thread the name of the child thread is placed in the local variable **last_background_action** and the parent does not wait for the child to terminate.

See also:

- **icon_rules** in the **xdt3**(XC) manual page
- **for_info** in the **deskcommands**(XC) manual page
- " &", "{ }", and **last_background_action** in the **deskshell**(XC) manual page

Pipelines

The pipeline operator, " | ", creates two child threads from the parent thread, and the parent is blocked until both children have finished executing.

For example, the command:

```
sort < F | uniq > F2
```

sorts records from the file *F* and sends the output to the **uniq** command, which then removes duplicate lines, and sends the output to a file *F2*.

See also:

- pipeline operator (" | ") in the **deskshell**(XC) manual page

Executing actions within the same thread

The **do_actions_of**, **do_drop_in_actions_of**, **do_menu_actions_of**, and **source** commands execute the specified actions within the same thread, rather than in a separate thread.

You might want to do this to keep control over the order in which the actions are executed. For example, the following script sends the example **myprint** trigger, below, to a sequence of icons in the variable **$***:

```
for i in $*
do
   do_actions_of myprint $i
done
```

Because a **do_actions_of** command is used, each **myprint** action executes and completes before the next one is started.

NOTE Because the action commands are executed in the same thread, changing a local variable in the trigger action for **myprint** will affect local variables in the current thread. Generally this is not a problem because only loop variables will be used in both scripts, and these are protected.

See also:

- **do_actions_of**, **do_drop_in_actions_of**, **do_menu_actions_of**, and **source** in the **deskcommands**(XC) manual page

Signals

Signals provide a way of passing messages between two threads. Any thread can send a signal to any other thread using the **kill** command.

Signal handlers are defined in exactly the same way as functions, with names that *must* begin with **sig**.

When a thread receives a signal, the function with the same name as the signal is called, interrupting any other action that the thread is carrying out. When the function returns, the previous action continues. If no function has been defined, or inherited from parent threads, the signal is ignored.

For example, the following program shows how a signal can be used to give the user the option of canceling a lengthy **compress** operation:

```
function sigstop
{
   stop=true
}

{
   if yni Cancel?
   then kill sigstop $threadname(2)
   fi
} &

stop=false
for i in $dynamic_args
do
   compress $i
   if $stop then exit fi
done
```

This command compresses icons that are dropped onto a hypothetical **compress** icon. Before compressing any of the icons, it puts up a **yni** dialog box in the background to give the user the option of canceling the command at any time. If the user chooses **Yes**, the background thread sends the specially defined signal **sigstop** to its parent thread, **$thread_name** (2), and this has the effect of setting the variable **stop** to "true". The next time the **if $stop ...** line is read, the operation will be canceled.

See also:

- **$thread_name**(2) in the **deskshell**(XC) manual page
- **kill** and **yni** in the **deskcommands**(XC) manual page

Standard signals

A number of standard signals are defined in the system thread, and these can be used in other threads for particular functions. The default signal is **sigint**, and this is used if no signal name is given in the **kill** command. By default, this displays a **for_info** dialog box and then terminates the thread with an **exit** command.

Threads that need to exit in a particular way can redefine **sigint**. The example in "Signals" (page 368) could also have been written to use **sigint** rather than the user defined signal **sigstop**.

The **sigexit** signal is sent to the thread when it terminates. Normally no **sigexit** function is defined, so this has no action and the thread terminates immediately. However, you can define a **sigexit** function if you need to perform some cleaning up when the thread exits, such as deleting temporary files.

The signal **sigkill** causes a thread to terminate immediately, without calling the **sigexit** or **sigkill** functions.

See also:

- **exit** in the **deskshell**(XC) manual page
- **for_info** and **kill** in the **deskcommands**(XC) manual page

Chapter 26
Mapping mouse triggers for the Desktop

The actions that are taken in response to a mouse action (clicking, dragging, or holding one or more mouse buttons) are defined by associating them with a trigger name instead of a specific physical mouse button. For example, the action "select an icon" is associated with the trigger name **"activate"** instead of being directly defined as the mouse action "double-click mouse button 1."

This mapping of mouse actions with trigger names allows you to define action sequences for any type of mouse containing one to five mouse buttons.

NOTE Although the SCO OpenServer system offers this flexibility in mapping triggers, you are advised against modifying this mapping indiscriminately. The trigger mapping is optimized to the particular mouse supplied with your system. Make changes with caution because any change you make has possible effects on other trigger functionality. Furthermore, the mouse trigger mappings are currently defined so that they are OSF/Motif compliant for three-button mice. Altering these trigger mappings will alter this compliance.

There are three resources that define mouse actions for the Desktop: **triggers.maxUpTime**, **triggers.thresholdDownTime**, and **triggers.maxMotion**. These resources control the time (milliseconds) that a mouse button can be up before a trigger ends (used to judge whether two button presses are a double-click or two independent clicks), the time (milliseconds) that a mouse button can be held before it is considered a hold instead of a click, and the distance (pixels) that the mouse pointer can move before a mouse button press is considered a drag, respectively. For more information on these resources, see Chapter 10, "Configuring mouse behavior" (page 195).

See also:

- "Modifying the mouse trigger mappings" (this page)

Modifying the mouse trigger mappings

To change the mouse trigger mapping, perform the following steps. For more information on each of these steps, see the sections immediately following this procedure.

1. Open the desired resource file for editing:
   - */usr/lib/X11/app-defaults/XDesktop3* for system-wide changes
   - **$HOME**/*XDesktop3* for individual changes
2. Redefine the trigger mapping, using this syntax:

   trigger [: ***modifiers***] [/ ***context***] = ***{trigger_name*** | ***action}***

   where ***modifiers*** can be **c, s, l,** or **m1 - m5** and ***context*** can be **b, p,** or **t.** When you are finished, save your changes and exit the resource file.
3. Restart the Desktop.

Step 1: Editing the resource file

You can change the default trigger mappings so that all users on your system use the new definitions, or you can simply change the mappings for an individual user.

NOTE Changing the trigger mappings should not be taken lightly. The trigger mapping is optimized to the particular mouse supplied with your system. Furthermore, the mouse trigger mappings are currently defined so that they are OSF/Motif compliant. Altering these trigger mappings will alter this compliance.

The default trigger mappings are defined in */usr/lib/X11/app-defaults/XDesktop3*. You must have *root* privileges to edit this file. It is good practice to make a backup copy of the file before making changes to it.

Individual users can also change the mouse trigger mappings for their own use by copying the trigger mapping section from */usr/lib/X11/app-defaults/XDesktop3* to a file called **$HOME**/*XDesktop3*. This file is used to specify personal resource specifications that are used by the Desktop. Unlike the **$HOME**/.*Xdefaults-**hostname*** file (used for many resource specifications), which is specific to a given host machine, the Desktop consults **$HOME**/*XDesktop3* on any host.

NOTE The *XDesktop3* file does not exist in the user's home directory by default. If this file is not currently present, you must create it before you can redefine the trigger mappings.

If you create this file for a user from the *root* account, you must assign the file the correct ownership permissions. Run the **chown** command to assign the correct owner and the **chgrp** command to assign the correct group to the *XDesktop3* file. If you created this file yourself, these steps are unnecessary.

When the Desktop starts, it checks to see if an *XDesktop3* file exists in **$HOME**. If such a file does exist, the resource values specified in the user resource file take precedence over any values assigned to the same resource for the system, or in the resource database.

See also:

- "Methods for specifying resources" (page 87) for a more detailed explanation of the resource files mentioned above

Step 2: Redefining the trigger mapping

Type the new trigger mapping using the syntax described below. The entry must begin with the resource ***triggers*mapping** and each trigger string must be followed by a semicolon, " ; ". The mapping may span multiple lines if all but the last line ends with a backslash, " \ ".

The syntax for a trigger mapping string is:

trigger [: ***modifiers***] [/ ***context***] = ***{trigger_name | action}***

where ***modifiers*** can be **c, s, l,** or **m1 - m5** and ***context*** can be **b, p,** or **t.**

Here are the meanings of the various flags and the other arguments:

- The ***trigger*** component of a trigger mapping string defines a mouse action by specifying one to five comma-separated steps (button presses). Each step can include presses of one or more mouse buttons.

  For example, a double-click of mouse button 1 is represented by:

  ```
  1,1
  ```

  A chording of mouse buttons 1 and 3 (pressing both buttons simultaneously) is represented by:

  ```
  13
  ```

  When a step includes more than one button, the step ends when all buttons are released; the order in which the buttons are pressed does not matter.

  If a trigger contains more than five steps, it is ignored.

- The ***modifiers*** component of a trigger mapping string, if any, defines dependencies the trigger has on the modifier keys ⟨Ctrl⟩, ⟨Shift⟩, and ⟨CapsLock⟩.

  The modifiers are:

Character	Meaning
c	⟨Ctrl⟩ key
s	⟨Shift⟩ key
l	⟨CapsLock⟩ key
m1-m5	system-defined modifiers

  If a modifier key is not specified, it is ignored unless the ***modifiers*** list is preceded by one of the following:

Character	Meaning
~	specified modifier keys must not be pressed
!	specified modifier keys must be pressed and other modifier keys may not be pressed

  For example, specify a press of mouse button 2 in which the ⟨Shift⟩ and ⟨CapsLock⟩ keys may not be pressed with this ***modifier***:

```
2:~sl
```

  Specify a press of mouse button 1 in which the ⟨Ctrl⟩ key must be pressed and no other modifier key may be pressed with this ***modifier***:

```
1:!c
```

- The ***context*** component of a trigger mapping string, if any, defines where the mouse pointer must be located for the trigger to be recognized.

  The context specifications are:

Character	Meaning
b	directory, desktop, or treeview window background
p	icon picture
t	icon title (name)

  For example, specify a press of mouse button 2 on an icon picture and title with this ***context***:

```
2/pt
```

- The ***trigger_name*** is one of the two component choices on the right side of a trigger mapping string. A ***trigger_name*** can refer to a trigger in rule files (**trigger_action** and **drop_in_action** clauses) and in Deskshell commands (**actions_of** and **drop_in_actions_of** commands).

For more information on the **trigger_action** and **drop_in_action** clauses, see the **xdt3**(XC) manual page. For more information on using Deskshell commands, see Chapter 25, "Writing Deskshell commands" (page 349).

The ***trigger_name*** can be one of the following one-letter identifiers, which specify the type of button press in the trigger:

Character	Meaning
s	static trigger (implies a single- or double-click)
d	dynamic trigger (implies drag)
h	hold trigger

The one-letter identifier can be followed by a number from 1 to 3 to represent the corresponding mouse button, or by a space and a name. Any name can be assigned except a single letter followed by numbers.

For example, this ***trigger_name*** assigns the identifier h3 when button 3 is pressed on an icon picture or title (name):

```
3/pt=h3
```

This ***trigger_name*** assigns the name **deselect** to a press of mouse button 1 on a desktop or directory window background:

```
1/b=s deselect
```

- The ***action*** is one of the two component choices on the right side of a trigger mapping string. Use an ***action*** when you want to associate a specific action directly with a trigger.

  An ***action*** includes a one-letter identifier to specify the type of action:

Character	Meaning
m	menu
r	rename
s	selection

Menu actions

Use a menu action if you want a menu to be displayed when a hold trigger occurs. Specify a menu action with **m** ***menuname***.

For example, this ***action*** displays the Desktop **Help** menu, defined by the "DesktopHelpMenu" rule, when a hold trigger occurs:

```
m DesktopHelpMenu
```

Rename actions

Use a rename action to invoke a **rename** command. A rename action takes no argument, but the trigger must occur on an icon and the last step must be a click:

```
r
```

Selection actions

Use a selection action to select one or more icons. If the pointer is in a directory window, the main Desktop, or another desktop window, and not on an icon, the action affects all icons in the window.

Specify the type of selection using:

Sequence	Meaning
!s or !rs	select icon(s) and unselect any previous selections
+s or +rs	select icons(s) and add to previous selection list
-s or -rs	unselect icon(s) from previous selection list
~s or ~rs	toggle select/unselect

Use the **rs** sequences to specify that the icon selections are to be made with a rectangle that the user drags from the point of origin to surround the selected icons.

For example, this ***action*** specification lets the user select one or more icons with a rubber-band selection rectangle, then unselects any previously selected icons:

```
1/b:bpt=!rs
```

See also:

- Chapter 19, "Defining Desktop triggers" (page 309) for a summary of the names and meanings of the currently defined triggers

Step 3: Restarting the Desktop

Once you have made the desired resource changes, you need to restart the Desktop so the newly defined values will be read. Select **Restart Desktop Session** from the main Desktop **File** menu. You are prompted to confirm that you want to restart the Desktop by a dialog box; click on **Yes**.

The Desktop starts again and reads your new resource values.

Appendix A

OSF/Motif window manager resources

Because the window manager is a major component of the SCO OpenServer Graphical Environment, there are a number of window manager resources that you may be interested in using. This appendix describes these resources.

NOTE The OSF/Motif resources discussed here are relevant to the SCO Panner window manager in both **pmwm** and **mwm** modes. However, some of the resources have different default values in the different modes. Where this is true, both values are described. (See "Selecting between SCO Panner and OSF/Motif modes" (page 220) for information on the different SCO Panner window manager modes.)

The **pmwm** mode uses a number of resources that are not applicable to **mwm** mode. These resources are described in "Setting SCO Panner resources" in *Using SCO Panner*.

The window manager uses three categories of resources:

- **Specific appearance and behavior resources:** These resources specify overall window manager appearance and behavior, such as keyboard and mouse behavior, icon size and placement, focus policies, and window frame size and shape. These resources do not control individual window manager components such as color or font style.

  The syntax for defining this category of resource is:

  Pmwmresource_name: resource_value***

  or

  Mwmresource_name: resource_value***

- **Component appearance resources:** These resources control the appearance of window frames, window manager menus, and icons. Pixmaps, colors, and fonts are the most commonly configured component appearance resources.

The syntax for defining this category of resource is:

Pmwm[component*]**resource_name*: *resource_value***

or

Mwm[component*]**resource_name*: *resource_value***

The ***component*** argument can take one of the following values:

client	indicates the window frames of all clients
feedback	indicates the dialog boxes displayed by the window manager
icon	refers to the icon box
menu	refers to the menus displayed by the window manager

You can omit the ***component*** argument when specifying a component appearance resource. If you do, the resource specification is defined for all of the window manager components.

To configure the title area of a client window frame specifically, use this syntax:

Pmwm*client*title*_resource_name: resource_value_

or

Mwm*client*title*_resource_name: resource_value_

To configure the appearance of all window manager menus specifically, use this syntax:

Pmwm*menu*_menuname*resource_name: resource_value_

or

Mwm*menu*_menuname*resource_name: resource_value_

- **Client-specific resources:** These resources control the appearance and behavior of the windows that are associated with a client or a class of clients. You can use these resources to customize the behavior of the window manager for individual clients.

  The syntax for defining this category of resource is:

  Pmwm*_client*resource_name: resource_value_

  or

  Mwm*_client*resource_name: resource_value_

  Here ***client*** identifies the client to which the resource applies. You can use either the client's binary name or class name. ***resource_name*** is the actual window manager resource variable you want to specify. Note that you can only use a client-specific window manager resource variable for resource specifications of this category.

The following sections describe the resources that you can use to customize the window manager in both **pmwm** and **mwm** modes. These resources are listed in reference tables, which organize the resources according to the aspect of the window manager that they configure. Following each reference table, in alphabetical order, is a description of each of the resources mentioned in the table. These description sections indicate if a resource belongs to the specific appearance and behavior category, the component appearance category, or the client-specific category.

Resources for configuring window focus policies

The following resources control colormap and keyboard input focus policies:

Table A-1 Focus policy resources

Name	Class	Value Type	Default
autoKeyFocus	AutoKeyFocus	true/false	true
autoRaiseDelay	AutoRaiseDelay	milliseconds	500
colormapFocusPolicy	ColormapFocusPolicy	string	keyboard
deiconifyKeyFocus	DeiconifyKeyFocus	true/false	true
enforceKeyFocus	EnforceKeyFocus	true/false	true
execshell	ExecShell	string	null
focusAutoRaise	FocusAutoRaise	true/false	true
keyboardFocusPolicy	KeyboardFocusPolicy	string	explicit
passButtons	PassButtons	true/false	false
passSelectButton	PassSelectButton	true/false	true
raiseKeyFocus	RaiseKeyFocus	true/false	false
startupKeyFocus	StartupKeyFocus	true/false	true
xGranularity	XGranularity	pixels	0
yGranularity	YGranularity	pixels	0

These resources are described in more detail below:

autoKeyFocus (Class: **AutoKeyFocus**)
This resource applies only when the **keyboardFocusPolicy** resource is set to "explicit." This resource controls what happens to the focus when the current active window is iconified. If the **autoKeyFocus** resource is "true," the focus automatically goes to the window that previously had the focus. (This is a specific appearance and behavior resource.)

autoRaiseDelay (Class: **AutoRaiseDelay**)
If the **focusAutoRaise** resource is "true" and the **keyboardFocusPolicy** resource is set to "pointer," the **autoRaiseDelay** resource is read. This resource specifies the number of milliseconds the window manager should wait before raising a window (bringing the resource to the top of the stack of windows) once the window has received the input focus. The default is 500 milliseconds. (This is a specific appearance and behavior resource.)

colormapFocusPolicy (Class: **ColormapFocusPolicy**)
This resource controls the colormap focus for the window whose colormap is currently installed and used for displaying everything in a server. The **colormapFocusPolicy** resource can take one of the following three values:

- "keyboard" means the window with input focus has colormap focus.
- "pointer" means the window with the pointer has the colormap focus.
- "explicit" means that the colormap has to be explicitly selected for a window.

To allow explicit selection of a colormap, assign a button or key to the function named **f.focus_color**. (See Chapter 12, "Customizing the window manager" (page 219) for more information on the **f.focus_color** function.) The default value of the **colormapFocusPolicy** resource is "keyboard." (This is a specific appearance and behavior resource.)

deiconifyKeyFocus (Class: **DeiconifyKeyFocus**)
If this resource is set to "true" and **keyboardFocusPolicy** is "explicit," a window receives input focus when it is deiconified, or converted to normal size from an icon. The default value is "true." (This is a specific appearance and behavior resource.)

enforceKeyFocus (Class: **EnforceKeyFocus**)
If this resource is "true," the window manager sets the input focus to a selected window even if it is a globally active window (a window that can be operated without setting focus to it.) If the resource is "false," input focus is not set to any globally active window (such as a scroll bar). This resource is "true" by default. (This is a specific appearance and behavior resource.)

execshell (Class: **ExecShell**)
This resource indicates the shell that the window manager uses when it executes a new client. The possible shells values are "/bin/.sh", "/bin/.ksh", and "/bin/.csh". The default value is "null", which specifies to execute a client from the user's home shell. (This is a specific appearance and behavior resource.)

focusAutoRaise (Class: **FocusAutoRaise**)
If this resource is "true," the window manager raises a window to the top of the stacking order when the window receives the input focus. The default value depends on the **keyboardFocusPolicy** resource, but in most cases is "true". If **keyboardFocusPolicy** is "explicit," **focusAutoRaise** is set to "true"; otherwise, **focusAutoRaise** is toggled to "false." However, you can assign a "true" or "false" value to this resource yourself, regardless of the value **keyboardFocusPolicy** is using. (This is a client-specific resource.)

keyboardFocusPolicy (Class: **KeyboardFocusPolicy**)
This resource specifies how the window manager should assign the input focus to a window, so the window with the input focus receives your keystrokes. This resource can take one of two values:

- "explicit" means you indicate the focus window by pressing the first mouse button with the pointer in the window.
- "pointer" means the keyboard focus follows the mouse pointer.

The default setting for **keyboardFocusPolicy** is "explicit." (This is a specific appearance and behavior resource.)

passButtons (Class: **PassButtons**)
If this resource is "true," the window manager passes button-press events to the client, even after the events are used for some window manager functions. The default value is "false." The window manager does not forward button-press events that it uses for window management functions. (This is a specific appearance and behavior resource.)

passSelectButton (Class: **PassSelectButton**)
This resource indicates whether a button-press that assigns input focus to a window is passed as an event to that window. By default this resource is "true," which means that the window manager passes the button-press event to the window after giving the keyboard focus to that window. This resource applies only when **keyboardFocusPolicy** is "explicit," because this is the only case that requires you to transfer input focus by clicking on a window. (This is a specific appearance and behavior resource.)

raiseKeyFocus (Class: **RaiseKeyFocus**)
This resource is available only when the **keyboardFocusPolicy** resource is set to "explicit." When this resource is "true," it specifies that a window raised by the **f.normalize_and_raise** function also receives the input focus. The default value is "false." (This is a specific appearance and behavior resource.)

startupKeyFocus (Class: **StartupKeyFocus**)
If this resource is "true" and **keyboardFocusPolicy** is set to "explicit," the window manager transfers input focus to a window when it is mapped. This resource is "true" by default. (This is a specific appearance and behavior resource.)

xGranularity (Class: **XGranularity**)
This resource indicates where your window will be redrawn when you move the sides of it to a non-standard location on the background. This feature improves the redraw rate of your window. The new location is specified as a value (*x*) that represents the number of horizontal pixels that comprise an interval between standard redraw locations. For example, if you move your window to horizontal pixel number 15 and the **xGranularity** resource is set to "9," the window is redrawn at pixel number 18, the nearest location that is a multiple of 9. The default value is "0". (This is a specific appearance and behavior resource.)

yGranularity (Class: **YGranularity**)
This resource indicates where your window will be redrawn when you move either the top or bottom to a non-standard location on the background. This feature improves the redraw rate of your window. The new location is specified as a value (*y*) that represents the number of vertical pixels that comprise an interval between standard redraw locations. For example, if you move your window to vertical pixel number 7 and the **xGranularity** resource is set to "3," the window is redrawn at pixel number 6, the nearest location that is a multiple of 3. The default value is "0". (This is a specific appearance and behavior resource.)

Resource for specifying window manager fonts

The **fontList** resource specifies the fonts that are used in all window manager decorations. The class for the **fontList** resource is **FontList**.

When specifying this resource, use the full font name, font name wildcards, or a font alias for the resource value. The default is the "-*-helvetica-medium-r-normal--12-*-*-*-*-*-iso8859-1" font. (See Chapter 7, "Changing fonts" (page 125) for more information on how to specify font resources.)

This resource can be used to specify lists of fonts, to accommodate the possibility that some systems may contain a set of fonts, while other systems contain a different set. If you list multiple fonts, they must be separated by white space.

Resources for coloring windows, icons, menus, and mattes

The following resources control the colors that are used in active and inactive window frames, icon images, menus, and mattes:

Table A-2 Color resources

Name	Class	Value type	Default
Windows, icons, and menus			
activeBackground	Background	color	scoActiveBackground
activeBottomShadowColor	Foreground	color	black
activeForeground	Foreground	color	scoActiveForeground
activeTopShadowColor	Background	color	scoActiveTopShadow
background	Background	color	scoBackground
bottomShadowColor	Foreground	color	black
foreground	Foreground	color	scoForeground
topShadowColor	Background	color	scoTopShadow
Mattes			
matteBackground	Background	color	value of *background
matteBottomShadowColor	Foreground	color	black
matteForeground	Foreground	color	value of *foreground
matteTopShadowColor	Background	color	scoBackground
matteWidth	MatteWidth	pixels	0
Icon images			
iconImageBackground	Background	color	value of *background
iconImageBottomShadowColor	Foreground	color	black
iconImageForeground	Foreground	color	value of *foreground
iconImageTopShadowColor	Background	color	value of *topShadowColor

NOTE Most of these color resources specify a palette resource variable, instead of a specific color. For example, the **activeForeground** resource specifies a value of "scoActiveForeground." These palette resource variables are replaced with a color value, depending on the color choices you make with the **scocolor** client. See Chapter 6, "Changing colors" (page 99) for more information.

The resources listed in Table A-2, "Color resources" (this page) are described in more detail below:

activeBackground (Class: **Background**)
This resource specifies the color of the active window manager window frame. The default value is "scoActiveBackground." (This is a component appearance resource.)

activeBottomShadowColor (Class: **Foreground**)
This resource specifies the color of the lower and right bevels of the active window frame. The default value is "black." (This is a component appearance resource.)

activeForeground (Class: **Foreground**)
This resource specifies the color of text in the active window frame. The default value is "scoActiveForeground." (This is a component appearance resource.)

activeTopShadowColor (Class: **Background**)
This resource specifies the color of the upper and left bevels of the active window frame. The default value is "scoActiveTopShadow." (This is a component appearance resource.)

background (Class: **Background**)
This resource specifies the background color used in all components of the window manager, particularly the background of windows. The default value is "scoBackground." (This is a component appearance resource.)

bottomShadowColor (Class: **Foreground**)
This resource specifies the color of the lower and right bevels in all window frames. The default value is "black." (This is a component appearance resource.)

foreground (Class: **Foreground**)
This resource specifies the color of text used in all components of the window manager, particularly in windows. The default value is "scoForeground." (This is a component appearance resource.)

iconImageBackground (Class: **Background**)
This resource specifies the background color for the window manager icon image. The default value is the color specified by the ***background** or ***icon*background** window manager resources. (This is a client-specific resource.)

iconImageBottomShadowColor (Class: **Foreground**)
This resource specifies the color used to create the bottom shadow of the icon image. The default value is "black." (This is a client-specific resource.)

iconImageForeground (Class: **Foreground**)
This resource specifies the foreground color of the icon image. The default value is the color specified by the ***foreground** or ***icon*foreground** window manager resources. (This is a client-specific resource.)

iconImageTopShadowColor (Class: **Background**)
This resource specifies the color used to create the top shadow of the icon image. The default value is the color specified by the ***topShadowColor** window manager resource. (This is a client-specific resource.)

matteBackground (Class: **Background**)
This resource specifies the background color of the matte. The matte is a three-dimensional border between the client's window and the window frame added by the window manager. This resource is used only if **matteWidth** is greater than zero. The default value is the color specified by the ***background** or ****client******background** window manager resources. (This is a client-specific resource.)

matteBottomShadowColor (Class: **Foreground**)
This resource specifies the color used to create the bottom shadow of the matte. This resource is used only if **matteWidth** is greater than zero. The default value is "black." (This is a client-specific resource.)

matteForeground (Class: **Foreground**)
This resource specifies the foreground color of the matte. This resource is used only if **matteWidth** is greater than zero. The default value is the color specified by the ***foreground** or ****client******foreground** window manager resources. (This is a client-specific resource.)

matteTopShadowColor (Class: **Background**)
This resource specifies the color used to create the top shadow of the matte. This resource is used only if **matteWidth** is greater than zero. The default value is the color specified by the ***topShadowColor** window manager resource. (This is a client-specific resource.)

matteWidth (Class: **MatteWidth**)
This resource specifies the width of the matte, in pixels. The default value is zero; no matte appears by default. (This is a client-specific resource.)

topShadowColor (Class: **Background**)
This resource specifies the color of the top and left bevels in all window frames. The default value is "scoTopShadow." (This is a component appearance resource.)

Resources for shading windows, icons, menus, and mattes

The following resources control the shading elements of windows, icons, menus, mattes, and icon images. Shading resources are most valuable when used with a monochrome display.

Table A-3 Shading resources

Name	Class	Value type	Default
Windows, icons, and menus			
activeBackgroundPixmap	BackgroundPixmap	pixmap	varies †
activeBottomShadowPixmap	BottomShadowPixmap	pixmap	varies †
activeTopShadowPixmap	TopShadowPixmap	pixmap	varies †
backgroundPixmap	BackgroundPixmap	pixmap	varies †
bottomShadowPixmap	BottomShadowPixmap	pixmap	varies †
topShadowPixmap	TopShadowPixmap	pixmap	varies †
cleanText	CleanText	true/false	true
Mattes			
matteBottomShadowPixmap	BottomShadowPixmap	pixmap	*bottomShadowPixmap
matteTopShadowPixmap	TopShadowPixmap	pixmap	*topShadowPixmap
Icon images			
iconImageBottomShadowPixmap	BottomShadowPixmap	pixmap	*icon*bottomShadowPixmap
iconImageTopShadowPixmap	TopShadowPixmap	pixmap	*icon*topShadowPixmap

† The default values for these resources are calculated dynamically, depending on your display and the values assigned to other color resources. For example, a monochrome display is assigned different default values than a color display.

All of the resources in this section, with the exception of **cleanText**, require a pixmap as a value. The following list describes the pixmap values that you can assign to these resources:

background	the background color (solid)
foreground	the foreground color (solid)
25_foreground	a mix of 25 percent foreground to 75 percent background
50_foreground	a mix of 50 percent foreground to 50 percent background
75_foreground	a mix of 75 percent foreground to 25 percent background
horizontal_tile	horizontal lines alternating between the foreground and background colors

slant_left	diagonal lines slanting to the left, alternating between the foreground and background colors
slant_right	diagonal lines slanting to the right, alternating between the foreground and background colors
vertical_tile	vertical lines alternating between the foreground and background colors

Figure A-1 provides examples of how each of these pixmaps are displayed.

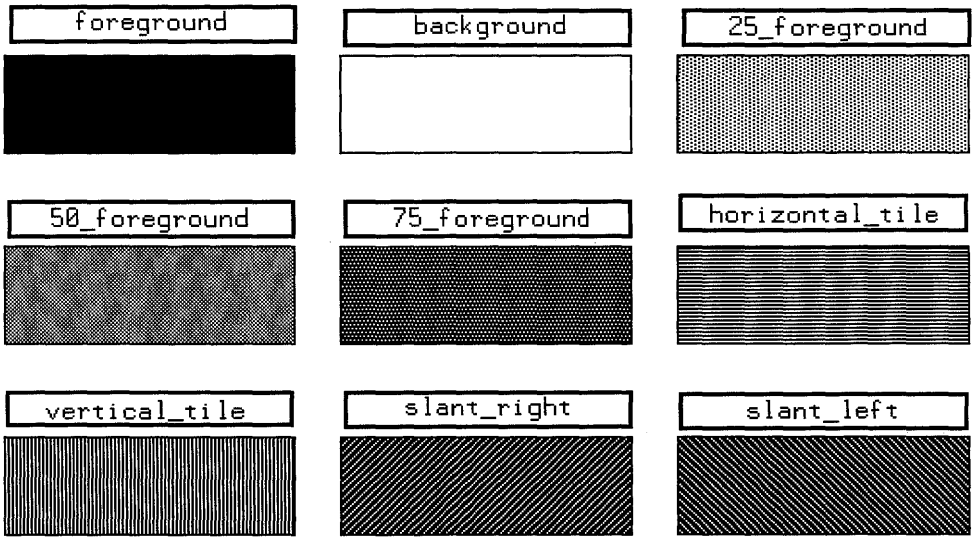

Figure A-1 Examples of valid pixmap values

The resources listed in Table A-3, "Shading resources" (page 386) are described in more detail below:

activeBackgroundPixmap (Class: **BackgroundPixmap**)
This resource specifies the pixmap used as the background in the window manager decorations of an active window. (This is a component appearance resource.)

activeBottomShadowPixmap (Class: **BottomShadowPixmap**)
This resource specifies the pixmap used for the lower and right bevels of the active window frame. (This is a component appearance resource.)

activeTopShadowPixmap (Class: **TopShadowPixmap**)
This resource specifies the pixmap used for the upper and left bevels of the active window frame. (This is a component appearance resource.)

backgroundPixmap (Class: **BackgroundPixmap**)
This resource specifies the background pixmap used to decorate the window frame of an inactive window. (This is a component appearance resource.)

bottomShadowPixmap (Class: **BottomShadowPixmap**)
This resource specifies the pixmap used in the lower and right bevels of all inactive window manager frames. (This is a component appearance resource.)

cleanText (Class: **CleanText**)
This resource can be used to make text easier to read on monochrome systems where a **backgroundPixmap** resource is specified. If this resource is set to "true," text appearing in a window's title and in the window manager's dialog boxes is displayed with a clear background. If this resource is "false," text is drawn directly on top of the existing background, even if the background uses a pattern. The default value for this resource is "true." (This is a specific appearance and behavior resource.)

iconImageBottomShadowPixmap (Class: **BottomShadowPixmap**)
This resource specifies the pixmap used for the bottom shadow of the icon image. The default value is the pixmap specified by the ***icon*bottomShadowPixmap** window manager resource. (This is a component appearance resource.)

iconImageTopShadowPixmap (Class: **TopShadowPixmap**)
This resource specifies the pixmap used for the top shadow of the icon image. The default value is the pixmap specified by the ***icon*topShadowPixmap** window manager resource. (This is a client-specific resource.)

matteBottomShadowPixmap (Class: **BottomShadowPixmap**)
This resource specifies the pixmap used for the bottom shadow of the matte. This resource is used only if **matteWidth** is greater than zero. The default value is the pixmap specified by the ***bottomShadowPixmap** or the ***client**bottomShadowPixmap** window manager resource. (This is a client-specific resource.)

matteTopShadowPixmap (Class: **TopShadowPixmap**)
This resource specifies the pixmap used for the top shadow of the matte. This resource is used only if **matteWidth** is greater than zero. The default value is the pixmap specified by the ***topShadowPixmap** or the ***client**topShadowPixmap** window manager resource. (This is a client-specific resource.)

topShadowPixmap (Class: **TopShadowPixmap**)
This resource specifies the pixmap used in the top and left bevels of all inactive window frames. (This is a component appearance resource.)

Resources for window decorations

The following resources are used to declare applicable functions and decoration elements for a client:

Table A-4 Window decoration resources

Name	Class	Value type	Default
clientDecoration	ClientDecoration	string	all
clientFunctions	ClientFunctions	string	all
transientDecoration	TransientDecoration	string	title border resizeh
transientFunctions	TransientFunctions	string	move resize

These resources are described in more detail below:

clientDecoration (Class: ClientDecoration)
This resource specifies the amount of decoration (buttons and frames) that the window manager applies to a client's top-level window. The value of this resource is a combination of one or more of the following names:

- "all" includes all decorations listed below.
- "border" displays the window border.
- "maximize" adds the maximize button, including the title bar.
- "menu" displays the Window menu button, including the title bar.
- "minimize" adds the minimize button, including the title bar.
- "resizeh" shows the border with resize handles.
- "none" suppresses all decoration.
- "title" adds the title bar and a border to the window.

The default for the **clientDecoration** resource is "all." You specify new values for this resource in two ways:

- Enable selected decorations. For example, if you want the **xclock** client's window to have a title bar with a **Window** menu and a resizable border only:

```
Pmwm*XClock*clientDecoration: menu resizeh
```

- Disable selected decorations. The syntax requires a minus sign to precede the first value. For example, if you want all the decorations except the maximize and minimize buttons, you would specify:

```
Pmwm*XClock*clientDecoration: -maximize minimize
```

(This is a client-specific resource.)

clientFunctions (Class: **ClientFunctions**)
This resource specifies which of the window manager functions apply to a client's top-level window. (See "Using window manager functions" (page 223) for more information.) The value of this resource is a combination of one or more of the following names:

- "all" includes all functions listed below.
- "close" refers to the **f.kill** function.
- "maximize" refers to the **f.maximize** function.
- "minimize" refers to the **f.minimize** function.
- "move" refers to the **f.move** function.
- "none" suppresses invocation of all functions.
- "resize" refers to the **f.resize** function.

The default for the **clientFunctions** resource is "all." You specify new values for this resource in two ways:

- Enable selected functions. For example, if you want to invoke only the **f.move** and **f.resize** functions on the **xclock** window, you can set this resource as follows:

```
pmwm*xclock*clientFunctions: move resize
```

- Disable selected functions. The syntax requires a minus sign to precede the first value. For example, if you want to apply all functions except **f.maximize** and **f.minimize**, you would specify:

```
pmwm*xclock*clientFunctions: -maximize minimize
```

Note that if you disable functions that are used by **Root** and **Window** menu options, or by window manager button and key bindings, the affected menu options are removed from menus and the affected button and key bindings no longer work.

(This is a client-specific resource.)

transientDecoration (Class: **TransientDecoration**)
This resource controls the amount of decoration that the window manager places around a transient (temporary) window, identified by the WM_TRANSIENT_FOR property on the window. The syntax for specifying this resource is the same as that for the **clientDecoration** resource. The default value for this resource is "title border resizeh," which means that transient windows appear with a title bar (without the Window menu button, minimize button and maximize button), a window border, and resize handles. (This is a specific appearance and behavior resource.)

transientFunctions (Class: **TransientFunctions**)
This resource specifies the window manager functions that the window manager allows for a transient (temporary) window, identified by the WM_TRANSIENT_FOR property on the window. The syntax for specifying this resource is the same as that for the **clientFunctions** resource. The default value for this resource is "move resize," which means that the window manager applies the functions **f.move** and **f.resize** to transient windows. (See "Using window manager functions" (page 223) for more information.) (This is a specific appearance and behavior resource.)

Resources for controlling window size and position

The following resources control the size and location of windows:

Table A-5 Window size and position resources

Name	Class	Value type	Default
Size resources			
frameBorderWidth	FrameBorderWidth	pixels	5
limitResize	LimitResize	true/false	true †
maximumClientSize	MaximumClientSize	w x h	fill the screen
maximumMaximumSize	MaximumMaximumSize	w x h	2X screen
resizeBorderWidth	ResizeBorderWidth	pixels	varies
resizeCursors	ResizeCursors	true/false	true
Position resources			
clientAutoPlace	ClientAutoPlace	true/false	true
interactivePlacement	InteractivePlacement	true/false	false
moveOpaque	MoveOpaque	true/false	false
moveThreshold	MoveThreshold	pixels	4
positionIsFrame	PositionIsFrame	true/false	true
positionOnScreen	PositionOnScreen	true/false	false
showFeedback	ShowFeedback	string	all
Other resources			
enableWarp	EnableWarp	true/false	true

† In **pmwm** mode, this resource is set to "false".

These resources are described in more detail below:

clientAutoPlace (Class: **ClientAutoPlace**)
This resource affects how the window manager places a client's window on the screen. If **clientAutoPlace** is "true," the window manager positions each

window with the upper left corner of the frame offset horizontally and vertically so that no two windows completely overlap. The default for **clientAutoPlace** is "true." (This is a specific appearance and behavior resource.)

enableWarp (Class: **EnableWarp**)
If this resource is "true," the window manager moves the mouse pointer ("warps" it) to the center of the window being resized and moves through keyboard accelerators (key combinations that activate menu options without displaying the menu.) If **enableWarp** is "false," the pointer is left at its previous position. The default setting is "true." (This is a specific appearance and behavior resource.)

frameBorderWidth (Class: **FrameBorderWidth**)
This resource specifies the width, in pixels, of the window frame border. This border width includes the three-dimensional shadows. The default value is 5 pixels. (This is a specific appearance and behavior resource.)

interactivePlacement (Class: **InteractivePlacement**)
If this resource is "true," the window manager prompts you for the position of each new window. You must press the mouse button to indicate where the window should be placed. By default, this resource is "false" and the window manager does not prompt you for the window position. (This is a specific appearance and behavior resource.)

limitResize (Class: **LimitResize**)
If this resource is "true," you cannot resize a window so it is larger than the maximum size. The default value for **pmwm** mode is "false," and "true" for **mwm** mode. (This is a specific appearance and behavior resource.)

maximumClientSize (Class: **MaximumClientSize**)
This resource sets the size of the client's window when it is maximized. Its value is ***width*** x ***height***, in pixels. For example, if you have a display that uses a resolution of 800x600 and you want a client window that can be resized to twice the size of your screen, you would supply this resource a value of "1600x1200". If this resource is not set, the maximum size is such that the window fills the screen. Values assigned to this resource override values specified for the **maximumMaximumSize** resource. (This is a client-specific resource.)

maximumMaximumSize (Class: **MaximumMaximumSize**)
This resource sets the upper limit on the maximum size that you can specify for a client window. The dimensions are given in pixels. For example, if you set this resource to "800x600," client windows cannot be larger than 800x600 pixels. The default value is twice the size of your screen. For example, if your display is 800x600 pixels, the default value for this resource would be "1600x1200". (This is a specific appearance and behavior resource.)

moveOpaque (Class: **MoveOpaque**) This resource is used to control the appearance of a window while it is being dragged. When "true," the entire window moves instead of just the wire frame. The default value is "false." (This is a specific appearance and behavior resource.)

moveThreshold (Class: **MoveThreshold**)
This resource controls how sensitive the window manager is to mouse drag operations. The value is interpreted as the number of pixels by which the mouse must move before the window manager reacts to it. The default value is 4 pixels. (This is a specific appearance and behavior resource.)

positionIsFrame (Class: **PositionIsFrame**)
This resource specifies how the window manager interprets the information about a client window's position as it appears in the WM_NORMAL_HINTS property or in geometry specifications. If this resource is "true," the position is taken to be that of the frame placed around the client window by the window manager; otherwise the position is that of the client window alone. The default value is "true." (This is a specific appearance and behavior resource.)

positionOnScreen (Class: **PositionOnScreen**)
If this resource is "true," the window manager places a client window entirely inside the screen. If the window's size exceeds the screen size, the window manager places the upper left corner of the window within the boundaries of the screen. The default value is "false," in which case, the window is located according to its defined geometry, even if that location is off the screen. (This is a specific appearance and behavior resource.)

resizeBorderWidth (Class: **ResizeBorderWidth**)
This resource specifies the width, in pixels, of a window frame that allows you to resize the window by dragging the border. For **mwm** mode, the default value is 5, 7, or 10 pixels, depending on the resolution of your display. For **pmwm** mode, the default is 8 pixels. (This is a specific appearance and behavior resource.)

resizeCursors (Class: **ResizeCursors**)
If this resource is "true," the cursor changes shape to indicate that the resize operation is available whenever the mouse pointer enters the window frame. By default, this resource is set to "false"; the cursor does not change shape when the pointer is focused on the window frame. (This is a specific appearance and behavior resource.)

showFeedback (Class: **ShowFeedback**)
This resource specifies when the window manager displays feedback information, which includes dialog boxes and boxes displaying window size and position during move and resize operations. The value of this resource is a combination of one or more of the following names:

- "all" shows all feedback information.
- "behavior" uses feedback to confirm any changes in the window manager's behavior.
- "kill" shows a dialog box when a SIGKILL signal is received.
- "move" shows position during moves.
- "none" suppresses all feedback.
- "placement" shows position and size during initial placement of window.
- "quit" displays a dialog box for confirming a request to exit the window manager.
- "resize" shows size when window is being resized.
- "restart" shows a dialog box to confirm any attempt to restart the window manager.

The default for the **showFeedback** resource is "all." You specify new values for this resource in two ways:

- Enable selected feedback. For example, if you want feedback during move and resize operations only, you can specify this resource as follows:

```
Pmwm*showFeedback:  move resize
```

- Disable selected feedback. The syntax requires a minus sign to precede the first value. For example, if you want feedback in all cases except during move, resize, and placement, you would specify:

```
Pmwm*showFeedback:  -move resize placement
```

(This is a specific appearance and behavior resource.)

Resources for configuring window manager icons

The following resources are used to configure window manager icons (not to be confused with the Desktop icons):

Table A-6 Icon resources

Name	Class	Value type	Default
iconAutoPlace	IconAutoPlace	true/false	true
iconClick	IconClick	true/false	true
iconDecoration	IconDecoration	string	all
iconImage	IconImage	pathname	varies
iconImageMaximum	IconImageMaximum	w x h	50 x 50
iconImageMinimum	IconImageMinimum	w x h	16 x 16
iconPlacement	IconPlacement	string	right bottom
iconPlacementMargin	IconPlacementMargin	number	null
lowerOnIconify	LowerOnIconify	true/false	true
useClientIcon	UseClientIcon	true/false	false

These resources are described in more detail below:

iconAutoPlace (Class: **IconAutoPlace)**
This resource controls where the window manager places the icon for a minimized window. If the resource is "true," the window manager places all icons in a specific area of the screen, determined by the **iconPlacement** resource. If this resource is "false," you can place the icons anywhere on the screen. The default setting for this resource is "true." (This is a specific appearance and behavior resource.)

iconClick (Class: **IconClick)**
If this resource is the default value of "true," the **Window** menu of an icon is displayed and left visible when you click on the icon. If this resource is set to "false", no Window menu is displayed when clicking on an icon. (This is a specific appearance and behavior resource.)

iconDecoration (Class: **IconDecoration)**
This resource affects the amount of decoration on the icon. The value of the resource can be a combination of the following values:

- "all" includes all settings below.
- "label" indicates that only the label, truncated to the width of the icon, is displayed.
- "image" means that only the image of the icon is displayed.

- "activelabel" specifies that the complete label, not truncated, is shown when the icon is active. For icons appearing in the icon box, the default value of **iconDecoration** is "image label." For icons displayed on the screen when the icon box is not activated, the setting is "image label activelabel." (If your display supports resolution lower than 800x600, the default value is "label" only.)

(This is a specific appearance and behavior resource.)

iconImage (Class: **IconImage**)
This resource specifies the name, including the full path, of an X bitmap file that the window manager uses as the icon for a client when the client's window is minimized. By default, the window manager displays a built-in, standard icon image for all applications. Note that the **useClientIcon** resource affects this resource. If **useClientIcon** is "true," an image supplied by the client application takes precedence over an icon you specify here. Also, see "Other resources for controlling windows" (page 399) for information on the **bitmapDirectory** resource, which specifies the default pathname for bitmap files. (This is a client-specific resource.)

iconImageMaximum (Class: **IconImageMaximum**)
This resource takes a value of the form *w* x *h*, where *w* and *h* specify the maximum width and height of an icon's image. The default is "50x50," in pixels. The maximum allowed is "128x128." (This is a specific appearance and behavior resource.)

iconImageMinimum (Class: **IconImageMinimum**)
This resource takes a value of the form *w* x *h*, where *w* and *h* specify the minimum width and height of an icon's image. The default value for this resource is "16x16," in pixels. This value is also the minimum size supported by the window manager. (This is a specific appearance and behavior resource.)

iconPlacement (Class: **IconPlacement**)
This resource specifies where the window manager should place the icons. The value is a sequence of two keywords of the form:

primary secondary

Here ***primary*** and ***secondary*** can take one of the following values:

- "top" specifies top-to-bottom placement.
- "bottom" specifies bottom-to-top placement.
- "left" specifies left-to-right placement.
- "right" specifies right-to-left placement.

The ***primary*** layout specifies where an icon is placed (in a row or a column) and in which direction. The ***secondary*** layout specifies where to place new rows or columns. The default value for **iconPlacement** is "right bottom,"

which means that the icons are placed from right to left on the screen, with the first row at the bottom, and any new rows added in the bottom-to-top direction. (This is a specific appearance and behavior resource.)

iconPlacementMargin (Class: **IconPlacementMargin**)
This resource specifies the margin, in pixels, between the edge of the screen and the icons appearing at the edge of the screen. The default value is variable, depending on the size of your display. The window manager determines the maximum number of icons that can fit in each row and column on your screen, including the space that should be used to separate the icons. This space value is then assigned to the **iconPlacementMargin** resource as its default value. You can change this resource by specifying a positive value. (This is a specific appearance and behavior resource.)

lowerOnIconify (Class: **LowerOnIconify**)
If this resource is "true," the window manager places a window's icon at the bottom of the stack when the window is minimized. This resource is "true" by default. (This is a specific appearance and behavior resource.)

useClientIcon (Class: **UseClientIcon**)
If the **useClientIcon** resource is "true," an image supplied by the client takes precedence over an icon you specify through the **imageIcon** resource. The default value is "false." (This is a client-specific resource.)

Resources for configuring the icon box

The following resources configure the icon box:

Table A-7 Icon box resources

Name	Class	Value type	Default
fadeNormalIcon	FadeNormalIcon	true/false	false
iconBoxGeometry	IconBoxGeometry	[columns x rows][±xoff±yoff]	1x6-0-0
iconBoxName	IconBoxName	string	iconbox
iconBoxTitle	IconBoxTitle	string	Icons
useIconBox	UseIconBox	true/false	false

These resources are described in more detail below:

fadeNormalIcon (Class: **FadeNormalIcon**)
If this resource is "true," the window manager grays out an icon that has been normalized. The **useIconBox** resource must be set to "true", for this resource to function. The default setting is "false", in which case the appearance of an icon is normal. (This is a specific appearance and behavior resource.)

iconBoxGeometry (Class: **IconBoxGeometry**)
This resource is a geometry specification for the icon box. The resource takes the value *[columnsxrows][±xoff±yoff]*, where ***columns*** and ***rows*** represent the size of the icon box, in icons, and *±xoff* and *±yoff* represent the ***x*** and ***y*** coordinates of the window. For example, if you specify the value of "4x3+0-0," the window manager creates a box large enough to hold three rows of four icons across and positions the box at the lower left corner of the screen. The default value for this resource is "1x6-0-0." (This is a specific appearance and behavior resource.)

iconBoxName (Class: **IconBoxName**)
This resource specifies the name that is used to set resources for the icon box. The default name is "iconbox." If you specify a new value for this resource, the window manager ignores any resources that use the name "iconbox." (This is a specific appearance and behavior resource.)

iconBoxTitle (Class: **IconBoxTitle**)
This resource specifies a string that is displayed in the title of the icon box. The default name is "Icons." (This is a specific appearance and behavior resource.)

useIconBox (Class: **UseIconBox**)
If this resource is "true," the window manager places all icons in an icon box. If this resource is "false," the window manager places the icons on the Desktop, or, if the Desktop is not running or is running in a window, on the Root window. The default value is "false." (This is a specific appearance and behavior resource.)

Other resources for controlling windows

The following resources control miscellaneous aspects of window management and behavior:

Table A-8 Window control resources

Name	Class	Value type	Default
Bindings			
buttonBindings	ButtonBindings	string	DefaultButtonBindings
keyBindings	KeyBindings	string	DefaultKeyBindings
Screen management			
multiScreen	MultiScreen	true/false	false
screens	Screens	string	varies
Client management			
quitTimeout	QuitTimeout	milliseconds	1000 †
saveUnder	SaveUnder	true/false	false
Mouse timing			
doubleClickTime	DoubleClickTime	milliseconds	500
Resource directories			
bitmapDirectory	BitmapDirectory	directory	/usr/include/X11/bitmaps
configFile	ConfigFile	file	.mwmrc ††
Root menu			
rootMenu	RootMenu	string	RootMenu
Window menus			
wMenuButtonClick	WMenuButtonClick	true/false	true
wMenuButtonClick2	WMenuButtonClick2	true/false	true
windowMenu	WindowMenu	string	DefaultWindowMenu

† In **pmwm** mode, this resource is set to "5000".

†† In **pmwm** mode, this resource is set to ".pmwmrc".

These resources are described in more detail below:

bitmapDirectory (Class: **BitmapDirectory**)
This resource specifies a directory that the window manager searches to locate any bitmaps needed by other window manager resources. The default setting of this resource is */usr/include/X11/bitmaps*. (This is a specific appearance and behavior resource.)

buttonBindings (Class: **ButtonBindings**)
This resource specifies a set of button bindings (a table that assigns an action to a button-press) that augments the built-in button bindings of the window manager. The value should be the name of a button binding from the window manager configuration file. The default value of the **buttonBindings** resource is "DefaultButtonBindings," as specified in the */usr/lib/X11/system.pmwmrc* or */usr/lib/X11/system.mwmrc* files. See Chapter 14, "Configuring window manager button bindings" (page 253) for more information. (This is a specific appearance and behavior resource.)

configFile (Class: **ConfigFile**)
This resource specifies the pathname of the window manager configuration file, which is a file with menu definitions, and button and key bindings. If the pathname specified by the **configFile** resource begins with ~/ (the tilde character followed by slash), the window manager considers that pathname to be absolute; otherwise, the path is assumed to be relative to the current directory. Here is how the window manager uses this resource:

1. If the environment variable **$LANG** is set, the window manager looks for the specified configuration file in the directory **$HOME/*lang***, which means in a subdirectory of your home directory, where the name of the subdirectory is specified by the language portion of the **$LANG** environment variable.
2. If the specified configuration file does not exist in **$HOME/*lang*** or if the **$LANG** environment variable is not defined, the window manager looks for that file in **$HOME**.
3. If you do not specify a **configFile** resource or if the specified file does not exist in one of the places listed in the first two steps, the window manager looks for a configuration file named *.pmwmrc* (for **pmwm** mode) or *.mwmrc* (for **mwm** mode). If the **$LANG** environment variable is set, it looks for **$HOME/*lang*/***.pmwmrc* or *.mwmrc*; otherwise, it looks for **$HOME/***.pmwmrc* or *.mwmrc*.
4. If neither a *.pmwmrc* or *.mwmrc* file exist, the window manager looks for a file named *system.pmwmrc* or *system.mwmrc*, first in the directory */usr/lib/X11/**lang*** and then in */usr/lib/X11*.

   Typically, the */usr/lib/X11/system.pmwmrc* or *system.mwmrc* file contains the default configuration for the window manager. You can copy this file to your home directory under the name *.pmwmrc* or *.mwmrc*, and modify it to suit your needs. See Chapter 12, "Customizing the window manager" (page 219) for more information.

(This is a specific appearance and behavior resource.)

doubleClickTime (Class: **DoubleClickTime**)
This resource specifies the maximum time, in milliseconds, that can elapse between two clicks that are to be interpreted by the window manager as a double-click. The default value is 500 milliseconds. (This is a specific appearance and behavior resource.)

keyBindings (Class: **KeyBindings**)
This resource specifies a set of key bindings (a table that assigns an action to one or more key press events) that replaces the built-in key bindings of the window manager. The value should be the name of a key binding from the the window manager configuration file. The default value of the **keyBindings** resource is "DefaultKeyBindings," as specified in the */usr/lib/X11/system.pmwmrc* or */usr/lib/X11/system.mwmrc* file. See Chapter 15, "Configuring window manager key bindings" (page 269) for more information. (This is a specific appearance and behavior resource.)

multiScreen (Class: **MultiScreen**)
If this resource is "true," the window manager controls windows displayed in all screens of a display. The default value is "false," which means the window manager manages only one screen by default. Note that this resource should not be used with the **Xsco** server, which does not support displays with multiple screens. (This is a specific appearance and behavior resource.)

quitTimeout (Class: **QuitTimeout**)
This is the amount of time, in milliseconds, that the window manager waits for a client to respond to a WM_SAVE_YOURSELF message. The client is supposed to reply by updating the WM_COMMAND property. The default value is 1000 milliseconds for **mwm** mode or 5000 milliseconds for **pmwm** mode. This resource applies only to those clients that have a WM_SAVE_YOURSELF atom but do not have a WM_DELETE_WINDOW atom in the WM_PROTOCOLS property of their top-level window. (This is a specific appearance and behavior resource.)

rootMenu (Class: **RootMenu**)
This resource specifies the name of the menu that is displayed when a mouse button is clicked in the Root window. The value of the resource must be the name of a menu defined in the window manager configuration file, the file specified by the resource **configFile**. The default for this resource is "RootMenu," as specified in the system-wide window manager configuration file. See Chapter 13, "Customizing window manager menus" (page 235) for more information.

saveUnder (Class: **SaveUnder**)
This resource indicates whether "save unders" are used for window manager components. For this to have any effect, save unders must be implemented by the X server. If save unders are implemented, as they are in the **Xsco** server, the server saves the contents of windows obscured by windows that have the

save under attribute set. If the **saveUnder** resource has a value of "true," the window manager sets the save under attribute on the window manager frame for any client that has it set. If the value is "false," save unders are not used on any window manager frames. The default value is "false." (This is a specific appearance and behavior resource.)

screens (Class: **Screens**)
This resource specifies the resource names to use for the screens managed by the window manager. If the window manager is managing a single screen, only the first name in the list is used. If the window manager is managing multiple screens (which is not supported by the SCO OpenServer system), the names are assigned to the screens in order, starting with screen 0. Screen 0 is assigned the first name, screen 1 the second name, and so forth. The default screen names are 0, 1, and so on. (This is a specific appearance and behavior resource.)

wMenuButtonClick (Class: **WMenuButtonClick**)
If this resource is "true," the window manager displays the **Window** menu in response to a button click on the Window menu button and leaves it displayed until another button click elsewhere on the screen. If this resource is "false," the **Window** menu remains on the screen only as long as you press and hold the mouse button. This resource is "true" by default. (This is a specific appearance and behavior resource.)

wMenuButtonClick2 (Class: **WMenuButtonClick2**)
If this resource is "true," and you double-click on the Window menu button, the window manager invokes the **f.kill** function to remove the client window. If this resource is "false," double-clicking on the Window menu button only posts that menu. This resource is "true" by default. (This is a specific appearance and behavior resource.)

windowMenu (Class: **WindowMenu**)
This resource specifies the name of the menu that is displayed when the Window menu button is pressed. The value of the resource must be the name of a menu defined in the window manager configuration file, the file specified by the resource **configFile**. The default for this resource is "DefaultWindowMenu," as specified in the system-wide window manager configuration file. See Chapter 13, "Customizing window manager menus" (page 235) for more information. (This is a client-specific resource.)

Appendix B

Desktop resources

Because the Desktop (**xdt3**) is a major component of the SCO OpenServer Graphical Environment, there are a number of **xdt3** resources that you may be interested in using.

The following sections describe the resources that you can use to customize **xdt3**. These resources are listed in reference tables, which organize the resources according to the aspect of **xdt3** that they configure. In most sections, a description of each of the resources follows the table.

NOTE Many of these resources are better changed using the Desktop **Preferences Editor**. See "Using the Preferences Editor" (page 24) for more information.

Resources for changing default rule files and directories

The following resources define the names and locations of the default rule files. (Refer to Chapter 16, "Customizing the Desktop with rules" (page 285) for information on the various types of rule files.) There is also a resource that defines the directories that are searched for bitmap and pixmap files.

NOTE In general, you should not change the rule file resources unless absolutely necessary.

Table B-1 Rule file resources

Name	Class	Default value
directoryRuleFile	DirectoryRuleFile	.xdtdir/*ll_TT*
userRuleFile	UserRuleFile	.xdtuserinfo
systemRuleFile	SystemRuleFile	/usr/lib/X11/IXI/XDesktop/rules/system/xdtsysinfo
pictureDirectory	PictureDirectory	/usr/lib/X11/IXI/XDesktop/bitmaps/xdt_c_large $HOME/.xdt_dir/bitmaps/xdt_large /usr/lib/X11/IXI/XDesktop/bitmaps/xdt_large /usr/include/X11/bitmaps

All resources listed in Table B-1, "Rule file resources" (this page) accept a file name or a full pathname for the resource value.

These resources are described in more detail below:

directoryRuleFile (Class: **DirectoryRuleFile**)
This resource defines the name of the directory rule file. The default value is *".xdtdir/ll_TT,"* where *ll_TT* is set to *"en_US."* If nothing is found, the Desktop then checks for an ".xdtdirinfo" file.

pictureDirectory (Class: **PictureDirectory**)
This resource defines the list of directories that is searched when a picture file with a relative name is specified. You can specify multiple directories with this resource but you must use colons or white space to separate the directories. This list is searched sequentially, so the most frequently accessed directories should be placed at the beginning of the list. The default directories listed in Table B-1, "Rule file resources" (this page) are searched in the order shown.

systemRuleFile (Class: **SystemRuleFile**)
This resource defines the name of the system rule file. This resource should only be changed when it is impossible to install the system rule file in its default location. This resource should never be set in a user's *.Xdefaults-**hostname*** file. The default value for this resource is "/usr/lib/X11/IXI/XDesktop/rules/system/xdtsysinfo".

userRuleFile (Class: **UserRuleFile**) This resource defines the name of the user rule file. The default value is ".xdtuserinfo".

Resource for specifying Desktop fonts

The **fontList** resource specifies the fonts that are used for text in the Desktop. The class for the **fontList** resource is **FontList**.

When specifying this resource, use the full font name, font name wildcards, or a font alias for the resource value. The default is the "-*-helvetica-bold-r-*--14-*-p-*" font. See "Changing Desktop fonts" (page 30) and Chapter 7, "Changing fonts" (page 125) for more information on how to specify font resources.

This resource can be used to specify lists of fonts, to accommodate the possibility that some systems may contain a set of fonts, while other systems contain a different set. If you list multiple fonts, they must be separated by white space.

NOTE The **fontList** resource controls all **xdt3** fonts except the icon label fonts. The resource for specifying icon label fonts is discussed in "Resources for configuring icon labels" (page 408).

Resources for specifying Desktop colors

The following resources control the general color characteristics of the **xdt3** client:

Table B-2 Desktop color resources

Name	Class	Value type	Default
activeBackground	Background	color	scoActiveBackground
activeBottomShadowColor	Foreground	color	black
activeForeground	Foreground	color	scoActiveForeground
activeTopShadowColor	Background	color	scoActiveTopShadow
background	Background	color	scoBackground
bottomShadowColor	Foreground	color	black
desktop.back.background	Desktop.Back.Background	color	scoAltBackground
foreground	Foreground	color	scoForeground
topShadowColor	Background	color	scoTopShadow

NOTE Most of these color resources specify a palette resource variable, instead of a specific color. For example, the **topShadowColor** resource specifies a value of "scoTopShadow." These palette resource variables are replaced with a color value, depending on the color choices you made with the **scocolor** client. See Chapter 6, "Changing colors" (page 99) for more information.

Generally, it is recommended that you use **scocolor** to change your Desktop colors instead of using these resources. See "Changing colors with the Color control" (page 27) for more information.

The resources listed in Table B-2, "Desktop color resources" (page 405) are described in more detail below:

activeBackground (Class: **Background**)
This resource specifies the background color used in directory and desktop window frames, when a window is active. The default value is the **scoActiveBackground** palette resource variable.

activeBottomShadowColor (Class: **Foreground**)
This resource specifies the color of lower and right bevels of the directory and desktop window frame, when a window is active. The default value is "black."

activeForeground (Class: **Foreground**)
This resource specifies the color of text used in directory and desktop window frames, when a window is active. The default value is the **scoActiveForeground** palette resource variable.

activeTopShadowColor (Class: **Background**)
This resource specifies the color of upper and left bevels of the directory and desktop window frames, when a window is active. The default value is the **scoActiveTopShadow** palette resource variable.

background (Class: **Background**)
This resource specifies the background color of directory and desktop windows. The default value is the **scoBackground** palette resource variable.

bottomShadowColor (Class: **Foreground**)
This resource specifies the color used in the lower and right bevels of the directory and desktop window frames. The default value is "black."

desktop.back.background (Class: **Desktop.Back.Background**)
This resource specifies the background color of the main Desktop, including the menu bar. The default value is the **scoAltBackground** palette resource variable.

foreground (Class: **Foreground**)
This resource specifies the color of text used by **xdt3**. The default value is the **scoForeground** palette resource variable.

topShadowColor (Class: **Background**)
This resource specifies the color used in the top and left bevels of the directory and desktop window frames. The default value is the **scoTopShadow** palette resource variable.

Resources for specifying cursor appearance

The following resources control the appearance of the **xdt3** cursor under various circumstances. The default cursors are built into the Desktop, but any of them can be redefined.

Each type of cursor has a *data* and *mask* pixmap component associated with it. Together, these components form a cursor shape. The data pixmap defines the image associated with the cursor and the mask pixmap defines the shape upon which the data pixmap is drawn. To modify a cursor's appearance, you must specify both a data and a mask pixmap for the cursor. If only one pixmap is specified, the resource is ignored.

The first element in each cursor resource specification indicates the type of cursor that you want to define. The different cursors are defined as follows:

alert	when an alert box is displayed (except within the text entry field)
bgTrigger	when a user clicks on the background
busy	when **xdt3** is processing
drag	when a single icon is being dragged
fatal	when an error box is displayed (except within the box)
iconTrigger	when a user clicks on an icon to select it
idle	when **xdt3** is waiting for the user to perform a task
multiDrag	when more than one icon is being dragged
rubber	when a user is using "rubberbanding" to select one or more icons

Table B-3 Cursor appearance resources

Data pixmap			Mask pixmap		
Name	Class	File	Name	Class	File
alert.data	Cursor.Bitmap	explode_d.xbm	alert.mask	Cursor.Bitmap	explode_m.xbm
bgTrigger.data	Cursor.Bitmap	grip_d.xbm	bgTrigger.mask	Cursor.Bitmap	grip_m.xbm
busy.data	Cursor.Bitmap	wait_d.xbm	busy.mask	Cursor.Bitmap	wait_m.xbm
drag.data	Cursor.Bitmap	drag_d.xbm	drag.mask	Cursor.Bitmap	drag_d.xbm
fatal.data	Cursor.Bitmap	fatal_d.xbm	fatal.mask	Cursor.Bitmap	fatal_m.xbm
iconTrigger.data	Cursor.Bitmap	grip_d.xbm	iconTrigger.mask	Cursor.Bitmap	grip_m.xbm
idle.data	Cursor.Bitmap	press_d.xbm	idle.mask	Cursor.Bitmap	press_m.xbm
multiDrag.data	Cursor.Bitmap	mdrag_d.xbm	multiDrag.mask	Cursor.Bitmap	mdrag_m.xbm
rubber.data	Cursor.Bitmap	grip_d.xbm	rubber.data	Cursor.Bitmap	grip_m.xbm

All resources listed in Table B-3, "Cursor appearance resources" (this page) accept filenames as values. Unless an absolute pathname is specified, the default picture directory, as defined by the **pictureDirectory** resource, is searched for the specified file. To change the default picture directory, see "Resources for changing default rule files and directories" (page 403). Pixmap files specify icon dimensions in pixels; no default values are assigned.

See Chapter 9, "Changing cursor appearance" (page 173) for more information on specifying the resources described in this section.

Resources for configuring icon labels

The following resources control general characteristics of icon labels on the Desktop and in directory and desktop windows:

Table B-4 Icon label resources

Name	Class	Value type	Default
font	Font	string	-*-helvetica-medium-r-*--12-*-*-*-p-*-*-*
hilight.background	Hilight.Background	color	black
hilight.foreground	Hilight.Foreground	color	white
normal.background	Normal.Background	color	transparent
normal.foreground	Normal.Foreground	color	black

These resources are described in more detail below:

font (Class: **Font**)
This resource specifies the font that is used by all icon labels. The default value is "-*-helvetica-medium-r-*--12-*-*-*-p-*-*-*".

hilight.background and **hilight.foreground**
(Classes: **Hilight.Background** and **Hilight.Foreground**)
These resources specify the colors that are used for the background and text of selected icons. The default values render "white" text on "black" labels.

normal.background and **normal.foreground**
(Classes: **Normal.Background** and **Normal.Foreground**)
These resources specify the colors that are used for the background and text of unselected icons. The default values render "black" text on "white" labels.

Resources for controlling Desktop appearance and behavior

The following resources control the default appearance and behavior of the Desktop, and to a certain degree, desktop windows:

Table B-5 Desktop resources

Name	Class	Value type	Default
desktop.menubar	Desktop.Menubar	menu rule	DesktopMenuBar
exitConfirmEnabled	ExitConfirmEnabled	true/false	true
exitEnabled	ExitEnabled	true/false	true
exitOnClose	ExitOnClose	string	main
iconGrid.aisleWidth	IconGrid.AisleWidth	pixels	4
iconGrid.xOffset	IconGrid.XOffset	pixels	2
Maindt.geometry	Desktop.Geometry	[width x height][±xoff±yoff]	unspecified
isRoot	IsRoot	true/false	true
isRoot.borderColor	IsRoot.BorderColor	color	white
isRoot.borderWidth	IsRoot.BorderWidth	pixels	4
isRoot.focusToggle	IsRoot.FocusToggle	key sequence	Shift Alt<Key>F2

NOTE The Desktop is an **XmPrimitive** widget called **base**. It has an ancestor that is an **XmForm** widget called **desktop**. These elements may be included in resource specifications to change the bitmap patterns and colors of individual portions of the Desktop.

The resources listed in Table B-5, "Desktop resources" (page 409) are described in more detail below:

desktop.menubar (Class: **Desktop.Menubar**)
This resource specifies the menu rule, located in the system rule file, that defines the contents of the Desktop menu bar. The default value is "Desktop-MenuBar." See Chapter 24, "Configuring Desktop menus" (page 341) for more information on menu rules and Desktop menu bars.

exitConfirmEnabled (Class: **ExitConfirmEnabled**)
This resource specifies whether or not you are prompted to confirm an exit from the Desktop. A "true" value, which is also the default value, prompts you to confirm the exit. If the **exitEnabled** resource is set to "false," this resource is ignored. This resource is also ignored if the **isRoot** resource is set to "true."

exitEnabled (Class: **ExitEnabled**)
This resource specifies whether or not you can log out from the Desktop. The default value for this resource is "true." This resource is ignored if the **isRoot** resource is set to "true."

exitOnClose (Class: **ExitOnClose**)
This resource specifies some controls over conditions under which you can exit the Desktop. The following values can be used:

- "last" specifies that you can exit only after the last desktop window has been closed.
- "main" specifies that you can exit upon closing the main Desktop.
- "never" specifies that you cannot exit.

The default value for this resource is "main."

iconGrid.aisleWidth (Class: **IconGrid.AisleWidth**)
This resource specifies the number of pixels that exists between **xdt3** icons when they are laid out in grid positions. The default value is 4 pixels.

iconGrid.xOffset (Class: **IconGrid.XOffset**)
This resource specifies the width, in pixels, of the left hand margin when laying out **xdt3** icons in grid positions. The default value is 2 pixels.

Maindt.geometry (Class: **Maindt.Geometry**)
You can use the **Maindt.geometry** resource to determine the size and location of the Desktop. You can also assign this resource with the name of a different desktop window, to specify its size and location. For example, to resize a desktop window called *spreadsheet.dt*, you would specify the **spreadsheetdt.Geometry** resource.

This resource takes the value *[width* x *height][±xoff±yoff]*, where ***width*** and ***height*** represent the size of the window in pixels and ±***xoff*** and ±***yoff*** represent the x and y coordinates of the window.

isRoot (Class: **IsRoot**)
This resource specifies whether or not the Desktop occupies the Root window. If this resource is set to "true," the Desktop is expanded to cover the entire Root window. If this resource is set to "false," the Desktop is displayed in a window.

NOTE It is strongly recommended that you change this behavior using the Desktop **Preferences Editor** instead of modifying this resource. See "Using the Preferences Editor" (page 24).

isRoot.borderColor (Class: **IsRoot.BorderColor**)
This resource specifies the color of the border that is displayed around the Desktop. The default value is "white."

isRoot.borderWidth (Class: **IsRoot.BorderWidth**)
This resource specifies the thickness, in pixels, of the border that is displayed around the Desktop. The default value is 4 pixels.

isRoot.focusToggle (Class: **IsRoot.FocusToggle**)
This resource specifies the key combination that enables the Desktop to grab the keyboard focus. The default value is "⟨Shift⟩⟨Alt⟩⟨F2⟩," which is specified as "Shift Alt<Key>F2".

If the **isRoot** resource is set to "true," you must assign a key to toggle the keyboard focus, otherwise you cannot use the **xdt3** menu accelerators or answer prompts on the Desktop.

Resources for controlling directory appearance and behavior

The following resources control the appearance and behavior of **xdt3** directory windows:

Table B-6 Directory window resources

Name	Class	Value type	Default
directory.menubar	Directory.MenuBar	menu rule	DirMenuBar
directory.enableStatusBar	Directory.EnableStatusBar	true/false	true
directory.statusBarTellFile	Directory.StatusBarTellFile	true/false	true
directory.statusBarTellHidden	Directory.StatusBarTellHidden	true/false	false

NOTE Directories are constructed from a **Dir** widget, all with the name **directory**. **directory** has the following descendents:

- The background is an **XmPrimitive** widget called **back**.
- The scroll bars are **XmScrollBar** widgets called **hscroll** and **vscroll**.
- The status bar is an **XmLabel** widget called **back**.

These elements may be included in resource specifications to change the bitmap patterns and colors of individual portions of directory windows. For example, to specify the color of the background of directory windows, use the following resource specification:

```
XDesktop3*directory*back*background:    resource_value
```

The resources listed in Table B-6, "Directory window resources" (page 411) are described in more detail below:

directory.menubar (Class: **Directory.Menubar**)
This resource specifies the menu rule, located in the system rule file, that defines the contents of the Directory menu bar. The default value is "DirMenuBar." See Chapter 24, "Configuring Desktop menus" (page 341) for more information on menu rules and Directory menu bars.

directory.enableStatusBar (Class: **Dir.EnableStatusBar**)
This resource controls whether or not directory windows have status bars. The default value is "true."

directory.statusBarTellFile (Class: **Directory.StatusBarTellFile**)
This resource specifies whether or not to display the number of files that are contained in a directory window. The default value is "true."

directory.statusBarTellHidden (Class: **Directory.StatusBarTellHidden**)
This resource specifies whether or not to display the number of hidden (or dot) files contained in a directory window. The default value is "false."

Resources for defining message box appearance

The following resources specify the pixmaps that are used when displaying an **alert**, **fatal**, **fyi**, **gti**, or **yni** message. **fyi** messages provide information, **gti** messages prompt you for text input, and **yni** messages prompt you for a yes or no response.

Table B-7 Message box appearance resources

Name	Class	Value type	Default
message.alert.pixmap	Message.Logo.Pixmap	pixmap file	Uses Motif default
message.fatal.pixmap	Message.Logo.Pixmap	pixmap file	Uses Motif default
message.fyi.pixmap	Message.Logo.Pixmap	pixmap file	Uses Motif default
message.gti.pixmap	Message.Logo.Pixmap	pixmap file	Uses Motif default
message.yni.pixmap	Message.Logo.Pixmap	pixmap file	Uses Motif default

Resources for controlling Desktop mouse behavior

The following resources control several aspects of a mouse's behavior when it is used with **xdt3**. Also refer to Chapter 10, "Configuring mouse behavior" (page 195) for more information on using these resources.

Table B-8 Mouse behavior resources

Name	Class	Value type	Default
triggers.maxMotion	Triggers.MaxMotion	pixels	3
triggers.maxUpTime	Triggers.Time	milliseconds	500
triggers.thresholdDownTime	Triggers.Time	milliseconds	700

These resources are described in more detail below:

triggers.maxMotion (Class: **Triggers.MaxMotion**)
This resource controls the number of pixels the mouse pointer must move before a mouse button click is interpreted as a drag action. The default value is 3 pixels.

triggers.maxUpTime (Class: **Triggers.Time**)
This resource controls the length of time, in milliseconds, that a mouse button must be up before a trigger ends. The default value is 500 milliseconds.

triggers.thresholdDownTime (Class: **Triggers.Time**)
This resource controls the time, in milliseconds, that a mouse button must be pressed before a click is considered a "long click" rather than a "short click." The default value is 700 milliseconds.

Resources for mapping mouse triggers

The following resource controls the mouse trigger mechanisms:

triggers.mapping (Class: **Triggers.Mapping**)
Chapter 26, "Mapping mouse triggers for the Desktop" (page 371) discusses triggers (which are double-clicks or drags of the mouse buttons) and how to associate actions with specific triggers using the **trigger_actions** clause in rule files. These trigger actions can be modified or completely redefined.

Although **xdt3** offers flexibility in mapping triggers, you are advised against modifying this mapping on a whim. The trigger mapping has been optimized to the particular mouse supplied with your system. The */usr/lib/X11/app-defaults/XDesktop3* resource file includes alternate trigger mechanisms that are commented out. You may want to try using these definitions as an alternate. See Chapter 26, "Mapping mouse triggers for the Desktop" (page 371) for a detailed discussion on this issue.

Appendix C

Deskshell command summary

The following table lists all the Deskshell commands in alphabetical order. For each command, the following information is provided:

- whether it returns a list of strings
- whether it sets **status** apart from if the arguments are incorrect. "?" indicates that **status** is only set in some circumstances.
- whether it is allowed in a priority thread
- whether it generates a sub-list of the list argument. "yo" indicates that the sub-list is ordered.

Table C-1 Deskshell commands in alphabetical order

Command	Text	Status	Priority	Sub-list
"."	n	?	n	n
-predicate	n	y	y	n
absreadlink	y	n	y	n
act	n	y	n	n
attr	y	y	y	n
basename	y	n	y	n
break	n	n	y	n
bring_to_front	n	y	n	n
btf	n	y	n	n
canonical	y	n	y	n
cd	n	y	y	n
cdt	n	y	n	n

(Continued on next page)

Table C-1 Deskshell commands in alphabetical order
(Continued)

Command	Text	Status	Priority	Sub-list
cdw	n	y	n	n
check	n	y	n	n
chk	n	y	n	n
cldt	n	y	n	n
close_desktop	n	y	n	n
close_directory	n	y	n	n
continue	n	n	y	n
copy_desktop	n	y	n	n
copy_into	n	y	n	n
cpi	n	y	n	n
die	n	y	n	n
dirname	y	n	y	n
display_directory	n	y	n	n
dup	n	y	n	n
dupl	n	y	n	n
duplicate_file	n	y	n	n
duplicate_link	n	y	n	n
duplicate_symlink	n	y	n	n
dupsl	n	y	n	n
dynamic_rule	n	y	n	n
exit	n	?	y	n
extension	y	n	y	n
false	n	y	y	n
fileclass	y	n	y	n
followlink	y	n	y	n
for_info	n	y	n	n
fyi	n	y	n	n
get_attribute	y	y	y	n
get_out	n	y	n	y
get_resource	y	n	y	n
goi	n	y	n	n
gti	y	y	n	n
help	n	y	n	n
idr	n	y	n	n
in_text_window	n	y	n	n
irl	n	y	n	n
join	y	n	y	n
kill	n	y	n	n

(Continued on next page)

Table C-1 Deskshell commands in alphabetical order
(Continued)

Command	Text	Status	Priority	Sub-list
link_into	n	y	n	n
list_count	y	n	y	n
list intersect	y	n	y	yo
list sort	y	n	y	y
list subtract	y	n	y	yo
list uniq	y	n	y	yo
lni	n	y	n	n
make_new_file	n	y	n	n
mark_changed_directory	n	y	n	n
mcd	n	y	n	n
menu	n	y	n	n
merge	y	n	y	n
mkf	n	y	n	n
move_into	n	y	n	n
mvi	n	y	n	n
odt	n	y	n	n
odw	n	y	n	n
open_desktop	n	y	n	n
pbi	n	y	n	n
pixmap_check	n	y	n	n
popup	n	y	n	n
put_back	n	y	n	n
pwd	y	n	y	n
pxc	n	y	n	n
query contents	y	y	y	n
query main_desktop	y	n	y	n
query open_desktops	y	n	y	n
query open_directories	y	n	y	n
query picture	y	y	y	n
query pixmap	y	y	y	n
query selections	y	y	y	n
query size	y	y	y	n
query thread_info	y	y	y	n
query title	y	y	y	n
query visibility	y	n	y	n
rdr	n	y	n	n
readlink	y	n	y	n
relativepath	y	n	y	n

(Continued on next page)

Table C-1 Deskshell commands in alphabetical order
(Continued)

Command	Text	Status	Priority	Sub-list
remove	n	y	n	n
remove_dynamic_rule	n	y	n	n
rename	n	y	n	n
reorganize_desktop	n	y	n	n
report	n	y	n	n
report_status	n	y	n	n
reset	n	y	n	n
resource_line	n	y	n	n
return	n	?	y	n
rgw	n	y	n	n
rmi	n	y	n	n
sel	n	y	n	n
select	n	y	n	n
sequence	y	n	y	n
sh	n	y	n	n
shell	n	y	n	n
shell_window	n	y	n	n
show_greeting	n	y	n	n
sleep	n	n	n	n
sli	n	y	n	n
source	n	?	n	n
split y	n	y	n	
symlink_into	n	y	n	n
tdg	n	y	n	n
tds	n	y	n	n
tidy_desktop	n	y	n	n
tolower	y	n	y	n
toupper	y	n	y	n
true	n	y	y	n
unextended	y	n	y	n
variables	y	n	y	n
variation_class	n	y	n	n
vclass	n	y	n	n
yni	n	y	n	n

Symbols, numbers

A

B

C

D

E

F

G

H

I

K

L

M

Q

R

S

T

U

W

X

Y

1 May 1995
AU20009P001