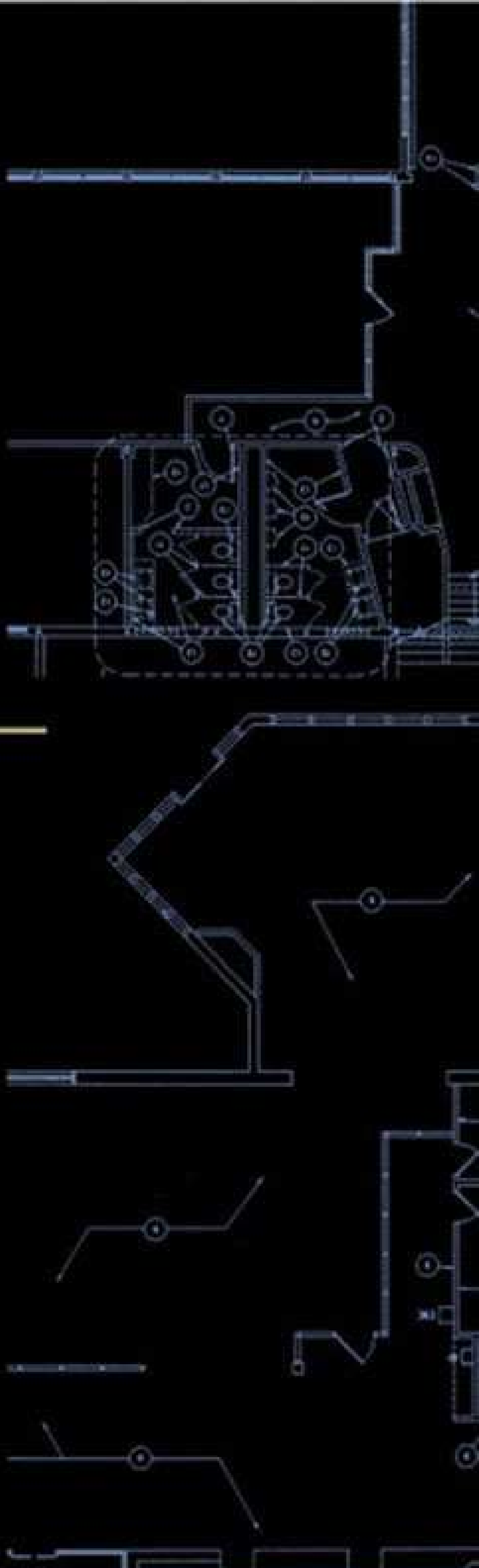
JESSE LIBERTY'S
from scratch
PROGRAMMING SERIES

X Window Programming *from* scratch



Learn X Window programming while building an application using Linux and C

QUE

J. Robert Brown

JESSE LIBERTY'S *from* scratch PROGRAMMING SERIES

X Window Programming *from* scratch

J. Robert Brown

QUE®

A Division of Macmillan USA

201 West 103rd Street,
Indianapolis, Indiana 46290

X Window Programming from Scratch

Copyright © 2000 by Que Corporation

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained herein.

International Standard Book Number: 0-7897-2372-7

Library of Congress Catalog Card Number: 00-100691

Printed in the United States of America

First Printing: August 2000

02 01 00 4 3 2 1

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Que Corporation cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The authors and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of the CD or programs accompanying it.

Associate Publisher
Tracy Dunkelberger

Acquisitions Editor
Katie Purdum

Development Editor
Hugh Vandivier

Managing Editor
Thomas F. Hayes

Senior Editor
Susan Ross Moore

Copy Editor
Cynthia Fields

Proofreaders
Harvey Stanbrough
Megan Wade

Technical Editor
Ed Petron

Team Coordinator
Vicki Harding

Media Developer
Michael Hunter

Interior Design
Sandra Schroeder

Cover Designers
Maureen McCarty
Anne Jones

Copy Writer
Eric Borgert

Production
Darin Crone
Steve Geiselman

Contents at a Glance

Table of Contents

About the Author

J. Robert Brown started his path to a career in software development by earning a college scholarship for Performing Arts in his homeland of central Ohio, where he held the misguided belief that he could be a movie star.

After years of either sleeping in his car or working three jobs concurrently to fund his way through an Electrical Engineering program, he found himself in Europe in the late 1980s working for the Department of Defense.

As a field engineer maintaining the mobile computer systems responsible for collecting and processing intelligence data, he realized that the position required too much manual labor. In 1991, he made his way through a Computer Science program at the European Division of the University of Maryland and although he didn't exactly finish in the top 10% of his class, he believes strongly that he helped those who did to get there.

John was invited to join Los Alamos National Laboratory as a Computer Scientist in 1996 where he remained until only recently. He now works for GTE Data Sources near Tampa, Florida.

Dedication

There are people who exist in the world who, once you've encountered them, change you forever. Through the strength of their character, depth of their spirit, or simply their presence in the world, they leave a lasting impression. I fear that we have one fewer such individual today due to the loss of Shel Silverstein in May, 1999. I hope for everyone there is someone who touches his or her life as Shel's works have touched mine.

Those without whom my life would mean less and this effort would not have been possible are my dear mother, Cindy Baker; my brother and best friend, Scott Brown; the absolute love of my life, Mikeala Elise; and the person who gave her to me, Kinnina McCray.

Acknowledgments

X Window Programming from Scratch is the result of efforts by many people. I am filled with awe and gratitude for the level of professionalism and quality the staff at Que publishing brought to this effort: specifically, Hugh "*Red*" Vandivier, Susan "*I need this back by Monday*" Moore, Cynthia "*Did this change your meaning?*" Fields, and Katie "*You'd better meet the deadline*" Purdum. And although no one likes to be told they have made a mistake, technical editor Ed "*Um, you might want to check this*" Petron was able to point out oversights in a way that never came close to wounding my ego. With the help of these and many others behind the scenes, this text is much better than I could ever have made it on my own.

There are others who contributed indirectly to the project by offering their friendship, encouragement, and patience as I tried to keep my head above water: namely, Mike Koscielniak, Cindy Sievers, and Jennifer Brown.

Tell Us What You Think!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

As an Associate Publisher for Que, I welcome your comments. You can fax, email, or write me directly to let me know what you did or didn't like about this book[md]as well as what we can do to make our books stronger.

Please note that I cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail I receive, I might not be able to reply to every message.

When you write, please be sure to include this book's title and author as well as your name and phone or fax number. I will carefully review your comments and share them with the author and editors who worked on the book.

Fax:	317.581.4666
Email:	`quetechnical@macmillanusa.com`
Mail:	Tracy Dunkelberger Associate Publisher Que Corporation 201 West 103rd Street Indianapolis, IN 46290 USA

Introduction

Welcome to *X Window Programming from Scratch*. You'll soon discover that there is much more to the text than the X Window System.

Because the X Window System is an environment, it rests upon a programming language as well as on an operating system. Skills using these as well as elements of object-oriented methodology, trigonometric and geometric functions, and the PostScript language will be detailed in this text.

Clearly, this text is not for the faint of heart.

Instead of requiring you to purchase separate manuals for the X Window System, programming language, operating system, and so forth, the intent of this text is to provide an erector set for computer programmers. Within the covers of this text are all the pieces necessary for accomplishing a Graphics Editor project.

Section One, "Starting Points," introduces the many pieces and encourages you to examine the ones with which you are less familiar.

Section Two, "Graphics Editor Application," leads you through the assembly of the pieces into a functional and rewarding project. You will be challenged to further the project and integrate the editor into other applications that you may be responsible for professionally.

Beyond the Title

Window-based user interfaces are the mainstream in professional level software development, and the X Window System holds its share of the community.

The title of this book indicates the attention that this text will pay to learning the X Window System, but there is more in store than just learning X.

The programming language selected for use in the text to employ the X Window System is the C programming language. This decision was made because of the frequency in the professional community of using C when doing X Window System programming.

Other languages can be used, but the approach is less direct and not suitable for introducing the environment.

The X Window System is *non-proprietary*. Not only can it be used with any operating system, but the source code is freely available as well.

Therefore, the next decision to make was what operating system to use. The decision was obvious: The targeted operating system is Linux, although the project has been tested on several operating systems.

According to the January 2000 issue of *ComputerWorld*, of all PC operating systems in use, Linux holds the lead with 38% of the market and growing. A close second is Windows NT with 25% of the market.

Honing skills in the Linux operating system is imperative for continued success in the software development profession.

Software Checklist

The required components for accomplishing the Graphics Editor project in this text include

- Linux Operating System
- C Compiler
- X Window System

Linux Operating System

You have the following choices when acquiring the Linux operating system:

Downloading Linux from the Internet

The Linux operating system is freely available for download from the Internet. However, it is not small, and if being pulled through a modem, the process may take awhile. Further, if you download it, you have to work out the kinks of installing and supporting it.

Generally, when downloading Linux the process of installing it consists of transferring the many files comprising the Linux installation components to floppy disk. Using a utility called `rawwrite.exe` provided with the download, you must create a *boot* disk and a *root* disk. The images from which you create these disks are selected based on the system you are installing Linux on and the features of the hardware you want to be supported.

For instance, unique images exist for a variety of network cards, modems, video cards, and so forth. Review the `README` file to understand the differences between available images before creating your startup disks.

After an image is selected and the necessary disks are created, you can put the boot disk into your floppy drive and reboot to begin the first of many installation attempts.

I believe strongly that the following generalization is true:

No one ever installs Linux once.

After several iterations and plenty of research, you'll have a mostly functional operating system.

Alternately, you can purchase a distribution of Linux.

Linux Distributions

Linux is a free operating system protected by the GNU General Public License. However, many vendors offer for sale a Linux *distribution*.

A distribution that requires you to pay more than the cost of the media and postage is generally enhanced in one of many ways.

Either by adding an installation program that automates the selection of the proper Linux kernel, or by adding features to the environment such as utilities to configure your windowing environment or system options, the vendors earn and justify the costs of a distribution.

The vendors advancing Linux include Red Hat, Slackware, Debian, SuSE, and others.

Having experience with all of them, I am unabashedly (and free of charge) going to recommend Red Hat's Linux.

The kernel (core of the operating system) packaged with Red Hat's distribution is not different from the version you could download from the Internet or purchase in another distribution; however, the ease with which the Red Hat distribution installs, configures, and updates is worth the investment.

During the authoring of this text and the development of the Graphics Editor project, I used Red Hat 6.1. This version is packaged for very easy installation and configuration with the XFree86 X Window System and the GNU C Compiler.

The default X Windowing environment provided by the installation of Red Hat 6.1 is the GNOME desktop using the Enlightenment Window Manager. This is reflected in the screen shots used in the text.

To learn more about vendors distributing Linux as well as the availability for downloading the Linux operating system, issue the following command in the search field of your favorite Internet portal:

```
url: Linux
```

Because of the popularity of the Linux operating system, you will have to sift through many hundreds of matches.

Optionally, you can go directly to `http://www.redhat.com` and read about their latest release.

C Compiler

If you opt not to use a version of Linux provided by a vendor, it can be necessary to download and install a C language compiler separately.

Packaged with most distributions of Linux (including Red Hat) is the GNU C Compiler.

Because Red Hat's distribution of Linux used during the writing of the text includes the GNU C Compiler, this is the primary compiler used during the development of the project.

The project has been tested using other compilers and the text addresses compiler differences for managing the project, so feel free to use any C compiler available to you.

If you are using a version of Linux (or UNIX) that does not include a C compiler, it will be necessary to acquire one. The GNU C Compiler is available for free download from the Internet.

Refer to the Free Software Foundation Web site (`http://www.fsf.com` or search for `url:gnu` using your favorite browser) for information on sites providing the latest version of the compiler and its associated tools.

X Window System

The X Window System is free for downloading and is also provided with every distribution of Linux purchased from the vendors mentioned above.

Visit `www.xfree86.com` for the latest runtime and development environments included with most Linux distributions.

Optionally, you might be able to `ftp` (File Transfer Protocol) the X Window System from a variety of sites on the Internet, including `gatekeeper.dec.com` and `uunet.uu.net`.

Who Should Read This Book?

The text has several starting points, allowing readers in the range of seasoned professional developer to serious hobbyist to enter at the point they are most comfortable.

Whether the choice is to start at Chapter 1 and learn the basics of the Linux operating system or to delve into Part Two and begin structuring the project, readers with a wide range of abilities will benefit from this text.

This book is intended for those seeking to

- Learn or further C programming skills
- Create Graphical User Interfaces with the X Window System
- Employ the Linux operating system for software development
- Gain a greater knowledge of computer graphics programming
- Challenge their computer problem-solving skills

The learning curve for the text rises very quickly. Therefore readers should have some understanding of structure programming concepts.

Conventions Used in This Book

Some of the unique features in this series include

Geek Speak. An icon in the margin indicates the use of a new term. New terms appear in the paragraph in *italics*.

how tōō prō nouns′ it

How To Pronounce It. You'll see an icon set in the margin next to a box that contains a technical term and how it should be pronounced. For example, "`cin` is pronounced *see-in*, and `cout` is pronounced *see-out*."

EXCURSIONS

Excursions. These are short diversions from the main topic being discussed, and they offer an opportunity to flesh out your understanding of a topic.

Concept Web. With a book of this type, a topic can be discussed in multiple places as a result of when and where we add functionality during application development. To help make this all clear, we've included a *Concept Web* that provides a graphical representation of how all the programming concepts relate to one another. You'll find it on the inside front cover of this book.

Note

Notes give you comments and asides about the topic at hand, as well as full explanations of certain concepts.

Tip

Tips provide great shortcuts and hints on how to program more effectively.

Warning

Warnings warn you against making your life miserable and help you avoid the pitfalls in programming.

Code listings are provided throughout the book. Each code listing has a heading, and these are numbered sequentially within a chapter.

In addition, you'll find various typographic conventions throughout this book:

- Commands, variables, and other code stuff appear in text in a special `monospaced font`.
- In this book, I build on existing listings as we examine code further. When I add new sections to existing code, you'll spot it in **`bold monospace`**.
- Commands and such that you type appear in **boldface type**.
- Placeholders in syntax descriptions appear in a *`monospaced italic`* typeface. This indicates that you will replace the placeholder with the actual filename, parameter, or other element that it represents.
- This symbol ➥ at the start of a line of code means that a single line of code is too long to fit on the printed page. Continue typing all characters after the ➥ as though they were part of the preceding line.

Getting Started

If you're motivated by the many benefits previously outlined and have assembled the necessary software, you are ready to begin.

"Part One: Starting Points" provides the information needed to help you get started.

Starting Points

The complexity of the Graphics Editor can introduce many concepts that are unfamiliar to you.

I address as well the likelihood that I write for an audience gathered from a variety of backgrounds, interests, and experience levels. Therefore, you, the reader, must choose where you enter the text.

You might be comfortable with some of the ideas and disciplines employed by the Graphics Editor project and not with others. However, another reader might have confidence in areas you have not been exposed to during your pursuits.

Therefore, the first portion of this text provides a variety of starting points. Choose the one that best addresses your needs.

Where to Begin

I recommend that you spend sufficient time reviewing the areas that are new or less familiar to you and perhaps give only a cursory review of the subjects in which you are already confident.

What's at the End

Upon completing this section of the text, I expect all readers to have the same foundation, understand the same vernacular, and be prepared for the next section of the text.

Part I

Absolute Zero

In this chapter

Chapter 1

UNIX for Developers

The development of each phase of the Graphics Editor requires use of a unique skill set. As an application developer, you must become acquainted with many aspects of computer problem solving. Knowledge of the operating system, programming language, windowing environment, and project management is critical to excel in the trade. This chapter introduces the operating system, compiler, editor, and project files in sufficient depth to put you on the path of completing the Graphics Editor project.

Structured application development (writing computer programs using an applied methodology) begins with the operating system. At this level, you arrange, navigate, edit, and compile your source code. If your organizational skills are poor, you won't be able to locate the file you need to advance the functionality of your application. An inability to utilize the capabilities of an editor fully means you will be making changes (assuming you can find the file) one character at a time when you could be manipulating pages or regions. Worse, when the compiler tells you `syntax error: line 1162`, you have to press the down arrow key 1,162 times to fix the problem because you don't know the go-to command.

Looking into the Linux operating system, you see that many tools (commands) are available to you. Some are sharp and can do damage if used incorrectly; others are obscure and hard to understand, and not all of them are useful for programmers.

In the following sections we'll take a few of the more pertinent tools out of the box and look at them together. If you are already well acquainted with the Linux operating system, feel free to move ahead to the next section.

The `man` Command

Having something to do with everything, `man` (short for *manual*) offers help for most all Linux commands. Think of `man` as a librarian and ask it questions when you want to read the manual for a specific topic.

The output of `man` is called a *man page*.

Examining the use of `man` will give insight into the way that most every command in Linux is formed. The basic syntax follows the pattern:

```
command -flag(s) parameter(s)
```

A ***`flag`*** is usually a single letter following a hyphen (-) meant to instruct the command to alter its behavior. Further, the command acts upon the `parameter(s)`.

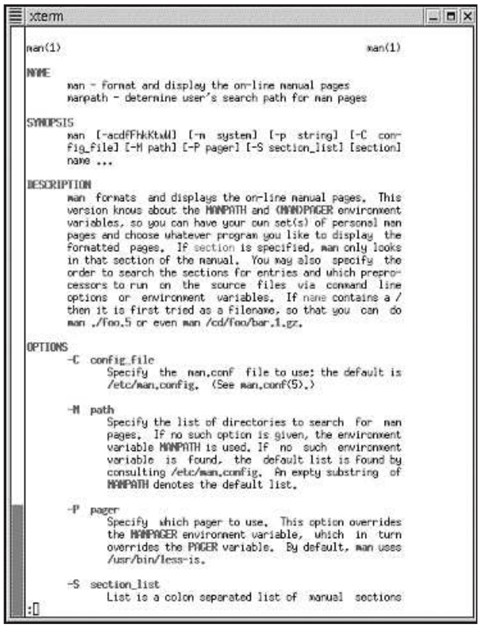
Consider an example where `man` is the command, there are no flags specified, and ***`man`*** is the parameter to be acted on:

```
bash[1]: man man
```

Issuing the command `man man` results in the `man` command using its default behavior to display the man page for itself. Only a portion of the command output is shown in Figure 1.1; however, it is important that you execute the command on your system and review the entire output.

Figure 1.1

Sample of the `man man` *command.*



```
man(1)                                                      man(1)

NAME
       man - format and display the on-line manual pages
       manpath - determine user's search path for man pages

SYNOPSIS
       man  [-acdfFhkKtwW]  [-m  system]  [-p  string]  [-C  con-
       fig_file] [-M path] [-P pager] [-S section_list] [section]
       name ...

DESCRIPTION
       man  formats  and displays the on-line manual pages.  This
       version knows about the MANPATH and (MAN)PAGER environment
       variables, so you can have your own set(s) of personal man
       pages and choose whatever program you like to display  the
       formatted  pages.  If section is specified, man only looks
       in that section of the manual.  You may also  specify  the
       order to search the sections for entries and which prepro-
       cessors to run  on  the  source  files  via  command  line
       options  or  environment  variables.  If name contains a /
       then it is first tried as a filename, so that you  can  do
       man ./foo.5 or even man /cd/foo/bar.1.gz.

OPTIONS
       -C  config_file
              Specify  the  man.conf  file to use; the default is
              /etc/man.config.  (See man.conf(5).)

       -M  path
              Specify the list of directories to search  for  man
              pages.  If no such option is given, the environment
              variable MANPATH is used. If  no  such  environment
              variable  is  found,  the  default list is found by
              consulting /etc/man.config.  An empty substring  of
              MANPATH denotes the default list.

       -P  pager
              Specify  which pager to use.  This option overrides
              the MANPAGER environment variable,  which  in  turn
              overrides the PAGER variable.  By default, man uses
              /usr/bin/less-is.

       -S  section_list
              List is a colon separated list of  manual  sections
:
```

Note

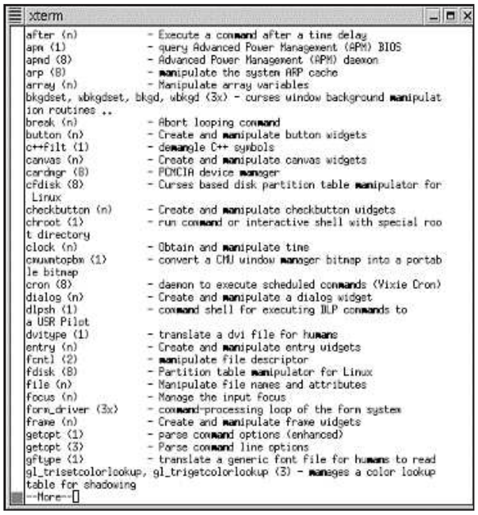
Commands distinguish between flags and parameters by use of the hyphen (-). A hyphen indicates to the command that what immediately follows is a flag.

You can know what flags or parameters a command accepts by looking at its man page. Anything placed between square braces ([ ]) in the Synopsis area of a command's man page is optional input. Reading the man page for `man`, you see the flags `acdfFhkKtwW` fall within an optional list; however, a topic *name* does not, and therefore must be specified.

If you use `man` followed by `-k`, the output is much different (as shown in Figure 1.2). Instead of getting the man page for a topic, you get a list of pages containing the string specified.

Figure 1.2

Sample of `man -k` *output.*



```
after (n)              - Execute a command after a time delay
apm (1)                - query Advanced Power Management (APM) BIOS
apmd (8)               - Advanced Power Management (APM) daemon
arp (8)                - manipulate the system ARP cache
array (n)              - Manipulate array variables
bkgdset, wbkgdset, bkgd, wbkgd (3x) - curses window background manipulat
ion routines ..
break (n)              - Abort looping command
button (n)             - Create and manipulate button widgets
c++filt (1)            - demangle C++ symbols
canvas (n)             - Create and manipulate canvas widgets
cardmgr (8)            - PCMCIA device manager
cfdisk (8)             - Curses based disk partition table manipulator for
 Linux
checkbutton (n)        - Create and manipulate checkbutton widgets
chroot (1)             - run command or interactive shell with special roo
t directory
clock (n)              - Obtain and manipulate time
cmuwmtopbm (1)         - convert a CMU window manager bitmap into a portab
le bitmap
cron (8)               - daemon to execute scheduled commands (Vixie Cron)
dialog (n)             - Create and manipulate a dialog widget
dlpsh (1)              - command shell for executing DLP commands to
a USR Pilot
dvitype (1)            - translate a dvi file for humans
entry (n)              - Create and manipulate entry widgets
fcntl (2)              - manipulate file descriptor
fdisk (8)              - Partition table manipulator for Linux
file (n)               - Manipulate file names and attributes
focus (n)              - Manage the input focus
form_driver (3x)       - command-processing loop of the form system
frame (n)              - Create and manipulate frame widgets
getopt (1)             - parse command options (enhanced)
getopt (3)             - Parse command line options
gftype (1)             - translate a generic font file for humans to read
gl_trisetcolorlookup, gl_trigetcolorlookup (3) - manages a color lookup
table for shadowing
--More--
```

EXCURSION

An Alternative to the `man -k` *Command*

A special notice of the `-k` option to `man`: On some UNIX systems `man -k` is a separate command called `apropos`, which is…well, more appropriate! Although the `man` command and its `-k` flag will always exist, `apropos` can be an abbreviated option to the `man -k` command on your version of UNIX.

The `man` command is a suitable first command to discuss because it will assist you in learning other commands to add to your repertoire. Feel free to issue the `man` command for any commands or keywords that appear in upcoming sections.

Organization and Navigation

The organization of a new project must be carefully planned. Consider as an example the placement of files used by the Linux operating system. With great consistency, the directory structure and file locations in one version of UNIX (within a family) will resemble another.

Note

A UNIX *family* is a branch within the evolutionary development tree of the operating system (see Figure 1.3). In other words, following the inception of UNIX at AT&T Bell Labs, UNIX has taken on a life of its own, growing like a tree where the efforts of the original authors serve as the trunk. Further developed at both Berkeley and AT&T, every version of UNIX falls under one of these two flavors. A UNIX family which follows from the work at Berkeley is called BSD UNIX, whereas those that follow from AT&T are known as SYSV. Linux happens to be a BSD UNIX that borrowed things from SYSV.

how too prō nouns′ it

SYSV is pronounced as if it were written *System 5*.

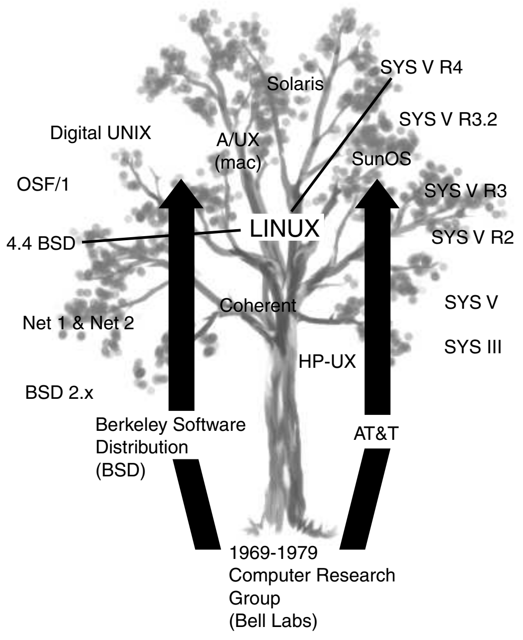


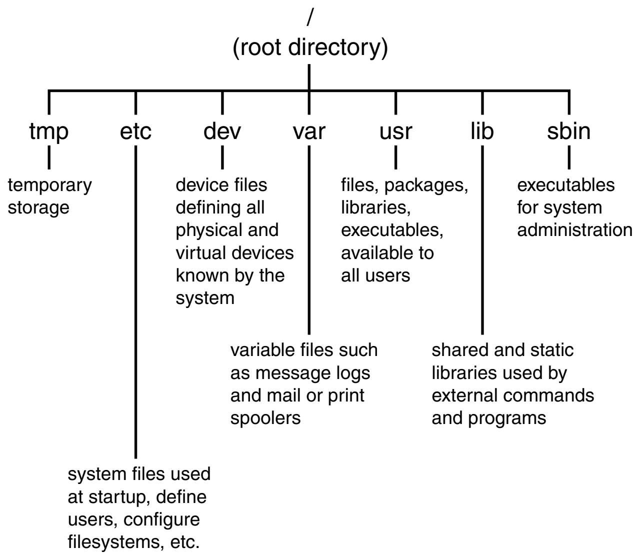
Figure 1.3

The UNIX family tree.

If you want to affect the way your operating system works, you know (based on the flavor of UNIX) where to find the configuration file needed (see Figure 1.4). Logical organization and consistency mean you don't have to search for the correct directory and guess at the name of the file.

Figure 1.4

A sample of a UNIX layout.


/
(root directory)
tmp
etc
dev
var
usr
lib
sbin
temporary storage
device files defining all physical and virtual devices known by the system
files, packages, libraries, executables, available to all users
executables for system administration
variable files such as message logs and mail or print spoolers
shared and static libraries used by external commands and programs
system files used at startup, define users, configure filesystems, etc.

How does something with the complexity of an operating system compare to application development?

Professional-level software development implies a life cycle: The software has a starting point, it grows (sometimes painfully), it is changed, and eventually it matures. Even if you are the only programmer to support this cycle, you will have other projects during the life of this one. The organization and structure of the project should enable you to find the directory, file, or function you need by a process of induction.

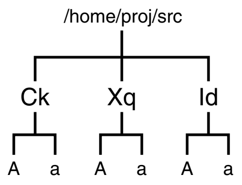
geek speak

Using *induction* to find a file or function means that by observing the conventions used in any piece of the application, you can surmise (guess) the name and location of the module you are seeking. Contrast this to a *deductive* process where you would examine every directory and every file one at a time, stopping only when you find what you are looking for.

The example of poor project structure illustrated in Figure 1.5 might be an extreme, but I've witnessed similar structures used in real-world projects. Clearly, moving from one directory or subdirectory to another is arbitrary. There is no clear way to know which place to go to find anything.

Figure 1.5

An example of poor project structure.

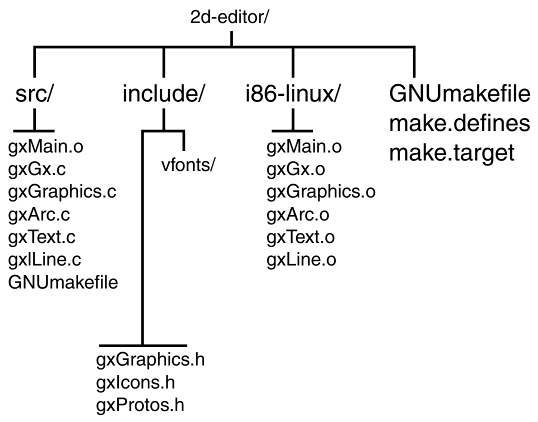


Reuse of directory names at multiple points in the poor project structure illustrated in Figure 1.5 and the seemingly meaningless names chosen make it impossible to induce the location of a file or function within the project. In fairness, even meaningless directory names could contribute to good organization if, for instance, all files and functions named in the directory were prefaced with the directory name.

After you've decided on the basic organization of your project, you are ready to open your Linux toolbox and begin. The structure of the Graphics Editor project is shown in Figure 1.6.

Figure 1.6

The Graphics Editor project layout.



Directories

As George Carlin pointed out, having a place for our stuff is a long-standing concern of mankind. The age of technology offers relief from this dilemma, however, because directories provide locations for storing the files created as part of a project. In fact, directories are the basic building blocks to a project's structure, allowing you to apply the organization that was carefully planned.

The `mkdir` Command

The Linux command to create a directory is `mkdir`. It stands for "make directory." Issuing the command with only a name causes the new directory to be placed in the current working directory. If you give an explicit or relative path to the command, the new directory is placed there.

Note

An *explicit* path is one where every component of the path is spelled out:

```
mkdir /home/users/jbrown/2d-editor
```

In a *relative* path, shortcuts are used to specify the location. UNIX provides two valid shortcuts for your use. They include a single dot for referring to the current directory (`.`) and two dots for referring to the current directory's parent directory (`..`). For instance, by using `../` in a directory specification, the path will be relative to the directory one level up:

```
mkdir ../2d-editor
```

In this example, the new directory `2d-editor` is placed in the directory above the current one.

Alternately, the *command shell* being used can provide other shortcuts.

EXCURSION

What Is a Command Shell?

A *command shell* (or simply *shell*) is assigned to every user of a UNIX system. The shell provides a window into the system, enabling you to issue commands, view the output, and navigate the file system.

Commands available to a user are grouped into two categories: *internal* and *external*.

External commands are provided by the UNIX operating system and will be available to all users regardless of the shell assigned because external commands reside as executable files on the UNIX file system.

Internal commands are shell specific, meaning that shells differ in the commands they provide internally. Generally, internal commands offered by a shell determine its suitability for scripting as discussed in Appendix A, "Command Shells and Scripting."

Just as shells differ in the internal commands they understand, they differ in the syntax or conventions they accept. These differences determine, in part, the shortcuts that the shell understands.

Table 1.1 provides a brief description of common UNIX shells. A more detailed discussion of command shells, conventions, and scripting can be found in Appendix A.

Table 1.1 UNIX Shells

Shell Name	*Command*	*Description*
Bourne	`sh`	The Bourne shell is the original shell provided with UNIX. Internal commands understood by `sh` and its derivative `ksh` (Korn Shell) include `export`, `for`, `while`, and `case`. These shells provide no navigational shortcuts.
C	`csh`	The C shell, and its variation the `tcsh (T C Shell)`, follow closely the syntax of the C programming language. Internal commands include `setenv`, `foreach`, and `while`.

continues

Table 1.1 Continued

		Shortcuts include the tilde (`~/`) for referencing your home directory or `~userid/` for referencing another user's home. For instance, `csh` and `tcsh` accept the following relative path: `mkdir ~/2d-editor`. This command creates a directory called `2d-editor` in your home directory.
Bourne Again	`bash`	The Bourne Again Shell is a combination of `sh`, `ksh`, and `csh` conventions. As the default shell assigned to user accounts under Linux, `bash` is a very capable shell. The shortcuts understood by `csh` are also known to `bash`.

The `rmdir` command

To remove a directory, use the command `rmdir`:

```
rmdir /home/users/jbrown/2d-editor
```

The `rmdir` command, which stands for "remove directory," only works if the directory is empty; this is important to remember. To remove a directory with something in it, issue the `rm -r` command:

```
rm -r /home/users/jbrown/2d-editor
```

The `rm` command with the `-r` flag is one of the sharpest tools in our box! You use the `rm` command to remove a file, but when told to be recursive (`-r`), directories and their contents are considered for removal as well, even if the directory contains other directories and those directories contain other directories, and so on. The `-r` flag sends the `rm` command to the lowest level of the directory structure, where it begins removing everything it finds as it works its way back up to the starting point. This is illustrated in Figure 1.7.

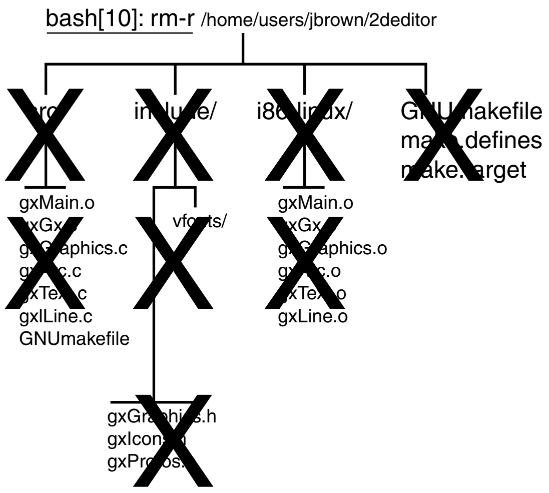
When using the `rm` command, you must specify the name of what you want to remove.

Note

The way to specify names for command completion under Linux can be very flexible when employing wildcards. Of course, this flexibility makes the `rm` command even more dangerous.

Figure 1.7

Recursive rm command.



Unlike other operating systems, Linux has no undo or undelete command for recovering files removed in error. When you unintentionally delete something (notice I said *when* and not *if*), remember two magic words and utter them over and over until you get a response from your system administrator. The words are *backups*. I guess that's only one word. You may have to append *please* to it for it to be magic!

Wildcards for command-line completion use a single character to represent multiple characters.

Commonly understood wildcards and their functions include:

*	Replaces any number of characters (broadest substitution)—for example, `bash[2]: rm ca*` expands to remove *capable, cannibal, canister, cat, cap, cal,* and *cab.*
?	Replaces only a single character—for example, `bash[2]: rm ca?` expands to remove only *cat, cap, cal,* and *cab.*
[a-d]	Replaces only occurrences of a single letter which falls between the specified range for example, `bash[2]: rm ca[a-d]` removes only *cab.*

Not to minimize the amount of caution that should be used when exercising the `rm` command, especially if using wildcards, but a safety net, called *permissions*, is built into Linux. A look at permissions follows in the next section on directory listings.

The `ls` Command

Requesting a directory listing determines whether anything is in a directory. The listing command is `ls`, and it accepts many flags to alter its behavior or output.

The `-l` flag passed to `ls` tells it to issue a long listing. A long listing includes information about permissions, ownership, modification date, and size. The default output for `ls` (no flags specified) is to list only the names of the files. Any file with a period (`.`) as the first character of its name is considered a *hidden file*, and you must explicitly request that `ls` display such a file by passing the `-a` flag.

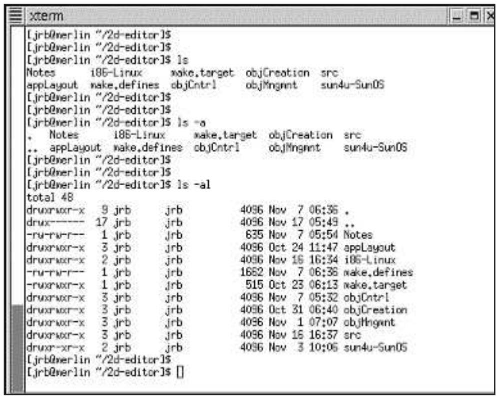
Note

When specifying multiple flags to a command, group them together following a single hyphen. Everything after the hyphen and until the next space is interpreted as flags and acted on accordingly. Issuing `ls -al` requests that the listing command displays all files in a long listing format.

Figure 1.8 shows sample use of the `ls` command, including examples of the `-l` and `-a` flags.

Figure 1.8

The `ls`, `ls -a`, and `ls -al` commands.



```
[jrb@merlin ~/2d-editor]$
[jrb@merlin ~/2d-editor]$
[jrb@merlin ~/2d-editor]$ ls
Notes      i86-Linux    make.target  objCreation  src
appLayout  make.defines  objCntrl     objMngmnt    sun4u-SunOS
[jrb@merlin ~/2d-editor]$
[jrb@merlin ~/2d-editor]$
[jrb@merlin ~/2d-editor]$ ls -a
.   Notes      i86-Linux    make.target  objCreation  src
..  appLayout  make.defines  objCntrl     objMngmnt    sun4u-SunOS
[jrb@merlin ~/2d-editor]$
[jrb@merlin ~/2d-editor]$
[jrb@merlin ~/2d-editor]$ ls -al
total 48
drwxrwxr-x   9 jrb      jrb          4096 Nov  7 06:36 .
drwx------  17 jrb      jrb          4096 Nov 17 05:49 ..
-rw-rw-r--   1 jrb      jrb           635 Nov  7 05:54 Notes
drwxrwxr-x   3 jrb      jrb          4096 Oct 24 11:47 appLayout
drwxrwxr-x   2 jrb      jrb          4096 Nov 16 16:34 i86-Linux
-rw-rw-r--   1 jrb      jrb          1662 Nov  7 06:36 make.defines
-rwxrwxr-x   1 jrb      jrb           515 Oct 23 06:13 make.target
drwxrwxr-x   3 jrb      jrb          4096 Nov  7 05:32 objCntrl
drwxrwxr-x   3 jrb      jrb          4096 Oct 31 06:40 objCreation
drwxrwxr-x   3 jrb      jrb          4096 Nov  1 07:07 objMngmnt
drwxrwxr-x   3 jrb      jrb          4096 Nov 16 16:37 src
drwxr-xr-x   2 jrb      jrb          4096 Nov  3 10:06 sun4u-SunOS
[jrb@merlin ~/2d-editor]$
[jrb@merlin ~/2d-editor]$ []
```

geek speak

Notice that when displaying all (`ls -a`) files in a directory you see `.` and `..` listed. The `.` (current directory) and `..` (parent directory) are special files Linux uses to maintain information about these directories. When you ask for a listing of your current directory, `ls` displays some of the information contained in the `.` file.

how too prō nouns' it

A `.` in UNIX is always pronounced as *dot*.

Permissions

Permissions serve several functions within the Linux operating system. They provide security, safety, and control. However, as with any tool, they must be used correctly. You must give thought to the permissions that are applied to the files and directories you create and manage.

You can find permissions for a specific file or directory in the first column of a long listing (`ls -l`) as you saw in Figure 1.8. Several groupings make up the permission field. Dissecting them into their groupings will make permissions very easy to understand.

```
drwxrwxr-x   9   jrb   jrb   4096   Nov 7 05:32   ObjCntrl
```

The first character of permissions is the *designator field*. This character indicates the type of entry in the listing. By the `d` in the first column of the example, you know that the entry is a directory. Other possible characters found in this position are `-` (hyphen) when the entry is a file or `l` when the entry is a symbolic link. I haven't discussed links yet but you'll soon see their usefulness to multi-platform application development. For now, remember what it means when you see an `l` in the designator field of permissions.

```
drwxrwxr-x   9   jrb   jrb   4096   Nov 7 05:32   ObjCntrl
```

The next grouping to consider in the permissions is the *owner access* field. As the name implies, this determines the accessibility for the owner. Clearly, owning the entry means that as owner you can alter any access field within the permissions. However, access doesn't just limit; it also protects. If used properly, you will have a safety net that prevents accidents.

The first character of the owner access field governs read permission. The presence of the `r` in the first column of this field indicates that read access has been granted. If access were not granted, there would be a `-` (hyphen) placeholder. The second character of the owner access field determines write permission, where a `w` indicates the capability to write to the file, and a `-` (hyphen) placeholder means that no write permission is granted. Last is execute permission as seen with either an `x` indicating permission to execute or a `-` for no permission to execute.

Note

Having execute permission is not always meaningful. For instance, writing a letter to a loved one and assigning execute permission to the file does not cause anything useful to happen. The file does not become a command by simply granting execute permission. (If accomplishing things on a computer were so easy, you probably wouldn't need this book.) However, if the file you create is a series of valid Linux or shell commands, giving execute permission creates a script or shell script for which execute permission is meaningful. (Shell scripting is discussed in Appendix A.)

```
drwxrwxr-x   9   jrb   jrb   4096   Nov 7 05:32   ObjCntrl
```

Following the owner access field is the field governing *group access*. In addition to a unique ID assigned to all Linux user accounts, everyone is placed into at least one group. This grouping is useful for allowing files and directories to be shared.

EXCURSION

What Group *Am I In?*

You have several ways to determine into which groups you have been placed. For instance, you can execute the `id` or the `groups` command. The output of the `id` command provides a lot of information about your Linux account, including the group assignments. The `groups` command will simply list the groups to which you are assigned.

The group access field manages sharing within a group. Applying the same read/write/execute pattern as for the owner access field, you can quickly determine that in the example anyone belonging to the same group as the group ownership of this entry may read, write, and execute.

EXCURSION

How to Determine Group Ownership

The SYSV family of UNIX does not show what the group assignment for an entry in a directory listing is unless you explicitly ask. To request that group ownership be displayed in the output of the `ls` command, you must pass the `-g` flag.

```
drwxrwxr-x   9   jrb   jrb   4096   Nov 7 05:32   ObjCntrl
```

The last field represents permissions that pertain to a collection of users known as the *world*. Any user who does not own the entry and is not assigned to the same group as the group ownership is considered part of the world. Users accessing the entry based on the merit of world permissions include anyone who shares an account on the system (and possibly network). For this reason, world permissions are generally not very giving, as you have no way to know or control those in the world beyond these permissions.

In the example, those in the world may only read and execute the entry. Note that a `-` placeholder resides in the write column, indicating no write access is granted.

Beyond the Obvious

What exactly can a user do with read, write, and execute permission?

Read Permission

Granting *read permission* enables users not only to view what has been placed into a file or directory, but also to copy it. In fact, you must have read permission to execute the `cp` (copy) command successfully.

Write Permission

Write permission allows users to modify any entry for which they have permission to write. Most importantly, though, having write permission enables users to remove a file or directory. You must have write permission to execute the `rm` or `mv` (move) command successfully. For this reason, if you are not actively changing a file or directory, removing write permission also removes your ability to delete a file in error.

Execute Permission

Being able to execute a file, as discussed earlier, means that each of the commands the script contains is performed as if it were typed at the command line. In the case of a compiled program, execute permission allows the user to run the program. However, if the entry is a directory, execute permission is required to change into that directory. If you do not have execute permission for a directory, it is effectively closed and the `cd` command will fail.

chmod

With an understanding of the power and utility of permissions, you must know how to change them. The `chmod` command, which stands for "change mode," allows you to modify the permissions of anything you own.

how tōō prō nouns' it

The `chmod` command is pronounced as if it were written *change mod*.

Syntax of the `chmod` command is direct.

```
chmod [ugoa][+ or -][rwx] name
```

where:

Item	*Means*
u	user/owner
g	group
o	world/others
a	all
+	add

`-`	remove
`r`	read
`w`	write
`x`	execute
`name`	item to be changed

For instance, the following command changes the permissions of `ObjCntrl` from the examples above to add write permission for the world:

```
bash[5]: chmod o+w ObjCntrl
```

To remove execute permission for both group and world, you could use the following command:

```
bash[5]: chmod go-x ObjCntrl
```

Any questions? Just ask `man` (`man chmod`).

The `cd` Command

A discussion on organization and navigation would not be complete if I didn't mention something about navigation. The `cd` or *change directory* command navigates around the Linux file system. It expects a single parameter to indicate to which directory you wish to change. If you execute the `cd` command without any parameters, you are returned to your home directory.

Note

Unlike other operating systems, Linux is case sensitive. When trying to change to a directory you know exists but Linux disagrees with, check the case because `./src` and `./Src` are unique names to Linux.

With an understanding of basic Linux commands, you are ready to advance to commands specific to software development.

The C Compiler

The C Compiler, known as `cc`, translates *source code* that a programmer has written into *object code* that the machine understands.

Source code can be read and written by you, a programmer. It spells out programming language elements such as keywords, variables, start of body markers, and end of body markers. A computer, however, knows nothing of the letters or characters that are meaningful to programmers. Instead, the computer expects that when you enter the letter *A* as part of a program solution, it will be translated into the machine code equivalent 0100 0001. Accomplishing this translation is the function of the compiler..

EXCURSION

A Closer Look at Machine Code

Machine code is ultimately a collection of 0s and 1s, which is all any machine can understand. At this level the collection of 0s and 1s is represented in the *binary* numbering system. Unlike the everyday *decimal* numbering system that consists of ten digits (0–9), the binary numbering system has only two digits, 0 and 1. Use of binary numbering to represent machine code can be very tedious to humans, so we also use the *hexadecimal* (16 digits) and *octal* (8 digits) numbering systems.

Table 1.2 shows various characters represented using the different numbering systems.

Table 1.2 Data Representation

Character	*Decimal*	*Binary*	*Hexadecimal*	*Octal*
1	49	0011 0001	0x31	\0061
2	50	0011 0010	0x32	\0062
>	62	0011 1110	0x3E	\0076
@	64	0100 0000	0x40	\0100
A	65	0100 0001	0x41	\0101
B	66	0100 0010	0x42	\0102
a	97	0110 0001	0x61	\0141

Determining which numbering system to use is largely a matter of convention based on how the data is used. This decision is also influenced by the size of the data being displayed. For instance, representing addresses in computer memory is most often done employing the hexadecimal numbering system.

Data representation is largely for the convenience of programmers; this is important to remember. The computer only understands 0s and 1s.

Object Files

Stating that the compiler translates the letter *A* from the source file into the machine code equivalent for the object file is an over-simplification.

It is true that *A* is translated to its machine code equivalent, but only if it is a literal. If it is a keyword, it is translated into an *op code* (code specifying an operation). If *A* is part of a variable name, it is translated into an *offset* (distance from the value of an internal control pointer). If part of a function name, *A* is translated to a *jump point* (distance from the base address of the application to where the function definition exists). If *A* is a function parameter, it is translated as an offset of the *stack pointer* (an internal control pointer that manages program execution), and on and on.

The complexity of the compiler is boggling, and I've only given a very high-level overview of its role. Entire books are dedicated to the subject of compiler design. For this reason, compilers are not generally given away for free, except, of course, the GNU C Compiler (`gcc`), which I will discuss shortly.

To review, an object file consists of only the machine codes (*op codes*, *jump points*, *offsets*, and so forth) translated from what you've written in the source file using a high-level language such as C.

Object files have the suffix `.o` so that you can distinguish them from files you author. Any attempt to edit or view the contents of an object file will make it instantly clear that there is nothing in it that you can affect.

how tōō prō nouns′ it

Object files, because of their `.o` extension, are referred to as *dot-oh's*.

Object files are platform specific: You cannot use an object file generated on an x86 PC platform on a platform of a different architecture. This is important to know.

The object file is the translation of what was written in the source file into a format that the computer understands. Not all computers do things in the same way. If you had an 80286-generation computer and you now own a Pentium, it is clear to you that processors are very different. The most notable difference in this example is speed. Performance, however, is not always a noticeable difference, but when it is, many factors contribute to it, such as the language set available to the processor or the manner in which it groups data into internal representations.

The Pentium processor is boasted to be a 32-bit processor, whereas the 80286 processors were only 16-bit architecture. The different number of bits (0s and 1s) supported by the processor means that the word size (internal bit groupings) will be different for each. Although a Pentium may be able to understand a 16-bit word size (grouping), an 80286 can never understand a 32-bit word.

EXCURSION

Computer Processor Instruction Sets

A processor's instruction set consists of the *op codes* (operational codes) available to it. These instructions are internal to the processor and dictate everything the processor knows how to do. Basic processor instructions include directions for moving data, arithmetic operations, program flow control, and more.

Instruction sets fall into two categories: Reduced Instruction Code Set (RISC) or Complex

Instruction Code Set (CICS)). RISC-based processors know fewer commands than CICS processors but operate more quickly as a result. In other words, the op codes contained within one processor's instruction set may not be the same as those of another processor's of a different architecture (and similar doesn't count). The differences in instruction sets of varying architectures introduce another level of binary incompatibility.

How does all this affect the object files? If you generated an object file on a 32-bit platform, the calculated offsets to memory for representing variables, for instance, would not align to valid memory addresses on the 16-bit system, nor would the jump-to addresses. The distance of the jump would likely be well outside the intended user memory area on the 16-bit platform. This illustrates, of course, one manner in which object files are platform specific.

I've barely mentioned the internal representation of data by a processor. In addition to word size (data grouping), some processors expect the order of the bytes (four-bit grouping) of data to be formatted differently from others.

One method of byte ordering is least significant bit first, called *little-endian*, and another is most significant bit first, called *big-endian*.

geek speak

Think of little-endian as *little end first*, big-endian as *big end first*, and endian as *end-ed*.

EXCURSION

Waiter, I'd Like To Order Some Bytes.

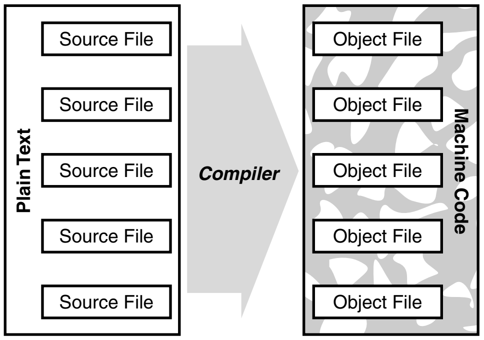
The determination of which byte order to employ for a given platform's architecture is largely a decision of the design engineers. It has occurred in the history of processor design that the decision to elect one method of byte ordering over another was based solely on a desire to avoid patent infringement.

Source Files

Previously, I spoke at length on the complexity of an object file as generated by the compiler. Figure 1.9 demonstrates the compile process with emphasis on the source files as *plain text*, meaning that they are understandable to programmers and contain no word processor type formatting. The figure further illustrates that the output of the compiler is not in human readable format, but rather in a format that satisfies a computer's requirements.

Figure 1.9

An illustration of the compile process.



Knowing the complexity of the compile process that a source file undergoes to be translated into something the computer can understand should make you appreciate that the compiler considers every character of a source file. In effect, as the compiler parses the source file it anticipates the next character or token it expects. If what is anticipated is not what it reads from the file, a syntax error is issued. For this reason, the source files must be plain text to prevent the extra control symbols used in word processing from confusing the very literal compiler.

Even the most basic text formatting performed by a word processor is accomplished by inserting control characters into the document so that the word processing application knows when to turn on and off the formatting. These control characters are, in fact, characters (not usually printable) that would be considered by the compiler as input. The choice of editor is important, as is how you choose to save the file if the editor supports multiple file formats.

An editor is available in all versions of UNIX to ensure that your authored source files contain only text. This is because it is incapable of word processing functions such as bolding, italicizing, and underlining.

The `vi` Editor

A popular choice among UNIX programmers is the `vi` editor. The fact that it is readily available on every version of UNIX, it does not require a lot of system resources, and it is fairly easy to master makes it a solid choice.

Certainly, other editors exist that satisfy the requirement of being able to save source files without any extra formatting data. A popular editor called GNU Emacs will enable you to write and save source files as required in addition to telling your fortune and generating whimsical quips. However, we must concern ourselves with availability and system requirements. Emacs is very large and will not always be installed on the system on which you must work.

Note

Even if `vi` is not your choice for an everyday editor, knowing its basic operations is useful because the day will come when you are sent to a customer site or asked for help by a colleague and your normal editor is not available. Nothing shatters confidence more than seeing someone not know how to edit a file.

To start `vi`, simply execute the command specifying the name of the file that you want to edit:

```
bash[2]: vi gxArc.c
```

If the file didn't exist previously, `vi` informs you that it was created. When loading an existing file, `vi` informs you of the number of characters read.

There are three modes in `vi`: *command mode*, *last-line mode*, and *edit mode*. `vi` always starts in command mode waiting for input. To move from command mode to edit mode, you must enter an appropriate command. Some commands enabling you to enter edit mode from command mode are shown in Table 1.3:

Table 1.3 `vi` Commands to Enter Edit Mode

`o`	begin inserting below current line
`O`	insert above current line
`a`	append following current cursor position
`i`	insert at current cursor position

After you have entered edit mode using one of these commands, everything you enter is inserted into the file.

To leave edit mode and return to command mode requires pressing the Esc key.

Other commands available in command mode are listed in Table 1.4.

Table 1.4 Other `vi` Commands Available in Command Mode

`x`	deletes character under cursor (when preceded by a number (`#x`), which deletes `#` of characters
`c<space>`	changes the current character
`cE`	changes everything from current cursor position until the end of the current word
`c$`	changes all characters from current position until the end of line
`dd`	deletes a line (can be preceded by a number to delete multiple lines)
`d$`	deletes from current cursor position until end of line

continues

Table 1.4 Continued

`yy`	yanks (cuts) current line (when preceded by a number (`#yy`) it will yank that many lines)
`p`	pastes everything in cut buffer (result of `yy` or `#yy`) after current line
`P`	pastes everything in cut buffer above current line
`0`	goes to the beginning of the current line
`$`	goes to the end of the current line
`/`*`str`*	forward searches for the next occurrence of *`str`*
`?`*`str`*	reverse searches for the previous occurrence of *`str`*
`G`	goes to the end of the file
`:`	enter last-line mode

Last-line mode of `vi` is identifiable by the presence of a colon at the bottom left corner of the window. This colon serves as an indicator of the last-line mode. As with edit mode, to return to command mode, press the Esc key.

Commands available in last-line mode are listed in Table 1.5.

Table 1.5 `vi` Last-Line Commands

`:w`	writes the changes made to the current file
`:w` *`filename`*	writes the current file as *`filename`*
`:q`	quits this session of `vi`
`:q!`	quits the current session without saving changes
`:e` *`file`*	loads *`file`* for editing
`:e!`	reverts to the last saved version of the current file (discards all unsaved changes)
`:r` *`file`*	inserts a copy of *`file`* at the current cursor position
`:n`	when editing multiple files, tells `vi` to go to the next file in the list
`:#`	# indicates a line of the file for `vi` to go to (as in `:1162` to go to line 1162)

Learning `vi` simply takes practice, but these basic commands will get you started.

Now able to create a project structure and edit source files, you are ready to learn how to manage building a project for multiple platforms.

The `make` Utility

The Linux `make` utility provides a means of managing projects, such as the Graphics Editor, that span multiple files. As illustrated in Figure 1.9, having many source files requires that each one be passed to the compiler for translation into a corresponding object file. Issuing the C Compiler command (`cc`) for each source file is a cumbersome effort prone to mistakes and oversights. I have purposely postponed discussion of the syntax of the `cc` command because of its complexity. This complexity, however, may be masked if viewed as a function of the `make` utility.

The `cc` Command

Note

Under Linux a reference to `cc` is synonymous with the GNU C compiler `gcc`. If you are using another version of UNIX, `gcc` and `cc` might not be synonymous at all but might be two separate commands. Knowing which C compiler will be used is critical because how you construct the command line to invoke them is very different. For instance, `gcc` understands the flag `-W` as a means of determining the level of warnings that the compiler issues. Other C compilers have different flags for specifying warning levels. You pass `gcc -ansi` to instruct it to be ANSI C-compliant, but the Sun WorkShop Pro compiler expects a `-X`*n* (where *n* is a directive for how ANSI C-compatible you want it to be).

EXCURSION

ANSI C Versus K&R C

The C programming language was written by Dennis Ritchie at AT&T in the early 1970s. With Brian Kernighan, he continued its development under what became known as K&R C. Unfortunately, some of the language details were unspecified, ambiguous, or incomplete. In 1983, the ANSI committee formed, and five years later C was standardized in what is now known as ANSI C.

The understanding to be taken from this discussion is the complexity involved when invoking a C compiler. If you use only Linux, the task is greatly simplified because you only must learn the `gcc` command and the flags and parameters that it understands. If your goal is to write software that is compatible with any family of UNIX, you must recognize compiler differences and how they affect the configuration of the `make` utility.

With an appreciation for the fact that there are differences in C compilers, we focus now on the `gcc` command, identifying what it has in common with other C compilers and contrasting differences where necessary.

`gcc -c`

The syntax of the `gcc` command follows most other UNIX commands with a few exceptions. As with other commands you've seen, `gcc` accepts flags that instruct it to alter its behavior. Common to all C compilers is the `-c` flag, which tells the compiler to generate the object file.

With the command

```
bash [3]: gcc -c gxArc.c
```

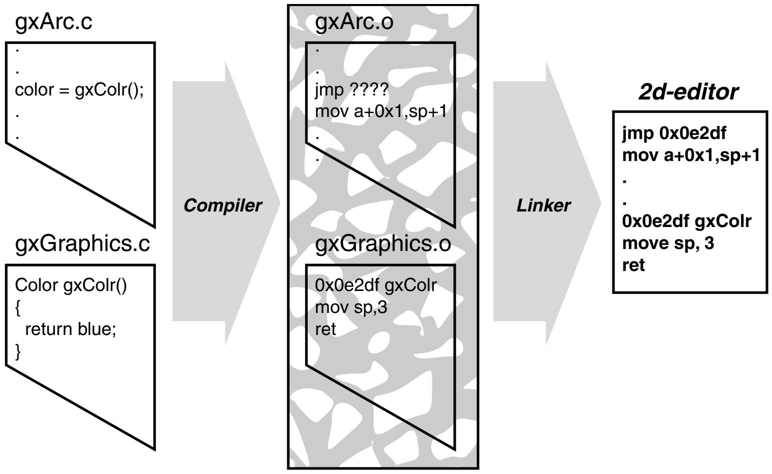
`gcc` stops after generating the `gxArc.o` file.

Why should you need to instruct the compiler to generate an object file? All along I've told you that the compiler automatically generates object files. The compiler does generate object files, but it also does more. The compiler links these object files together, forming an executable program.

As Figure 1.10 shows, an object file is the translation of a source code file, but it is not necessarily a complete program.

Figure 1.10

The compile and link process illustrated.



Modules such as `gxColr` might be invoked in one source file but actually defined in another. Notice in `gxArc.c` in Figure 1.10 that the variable `color` is assigned the result of module `gxColr`; however, the definition of `gxColr` is contained in `gxGraphics.c`. After the compiler converts the source code to object files, `gxArc.o` does not know the actual jump-to address for finding the `gxColr` module. Another task of the compiler is to link these references together so the program flows without dead ends. In `gxArc.o`, the jump-to address for `gxColr` is considered *unresolved*. Objects must go through the linker phase of compiling to determine what the external jump-to points should be.

Note

A source file invoking a function (module) contained in another source file creates an *external dependency*. If, at the end of the link phase, any external dependencies are left unresolved (meaning that invocations from one file could not be matched up with definitions in another), the program will not successfully link. The error reported will be `unresolved externals`.

EXCURSION

A Program Must Know Where It Is Going

When a program invokes a function that doesn't exist or can't be found, the result is a *bus error*. It is similar to getting on a bus to go somewhere and finding out that the address you were given was incorrect. Often, bus errors are a result of corrupted memory. Rather than the function not existing, which is something the linker must validate, the address given as the jump-to was corrupted during execution and therefore the function couldn't be found.

gcc -o

Knowing that the `-c` flag passed to `gcc` instructs the command to stop after creating the object file, another flag common to all C compilers is the `-o` flag. The `-o` flag must be followed by a `filename` because it instructs the command what to name the output.

```
bash[4]: gcc gxArc.o gxGraphic.o -o 2d-editor
```

By convention, the name of the object file is the same name as the source file, differing only by the extension. However, if you didn't instruct the compiler to stop after creating the object files and allowed it to continue through the link phase to produce an executable, you would certainly want to give the executable a name. Without `-o filename`, the compiler would name the program `a.out` instead of something meaningful.

gcc -g

Another interesting flag instructs `gcc` to provide debugging information in the objects that it creates. The `-g` flag is known as the *debug flag*. Without it, the compiler streamlines the object data by stripping out information that a debugger would need to enable you to evaluate variables, set breakpoints, and examine program flow.

gcc -W

As mentioned earlier, the `-W` flag that instructs `gcc` what level of warnings to issue while compiling the source code is a flag unique to `gcc`. Invoking `gcc` with `-Wall` requests the most stringent level of warnings and aids in the prevention of some programming errors.

Note

Programmers introduce three types of errors when writing source code: *syntax errors, semantic errors,* and *logic errors*.

The first level of errors is *syntax errors*, which the compiler easily finds. Syntax errors are created when you fail to obey the specifications of the language. For instance, in English, failing to place a period at the end of a sentence is a syntax error.

The second type of error is a *semantic error*, which occurs when you don't say what you mean to say. The `-Wall` flag passed to `gcc` is a great aid in discovering some semantic errors, because `gcc` analyzes context and usage as well as syntax.

Unfortunately, the third type of error, called *logic errors*, can only be found by exhaustively testing the program.

gcc -D

The `-D` flag is common to all compilers. This flag requires that a *directive* immediately follow it. This directive is then defined as a constant throughout the source file. Examples of how to employ compiler directives are provided in the next chapter.

```
bash[5]: gcc -g -ansi -Wall -c -DUSE_COLOR gxArc.c
```

As you can see, the examples for invoking the `gcc` command are becoming progressively more complex. In the preceding example, you are instructing `gcc` to include debugging information (`-g`), enforce the ANSI coding standard (`-ansi`), report all warnings (`-Wall`), stop after generating the object file (`-c`), and define the directive *`USE_COLOR`*. All of this is for the single source file *`gxArc.c`*.

The `-I`, `-l`, and `-L` flags are the last flags to consider before looking at how the `make` utility simplifies use of the `gcc` command.

gcc -I

The `-I` flag tells `gcc` where to find header files. Immediately after the `-I` flag you must provide a directory path that `gcc` will follow to find the header files that the source code has included.

Note

Like the `gxColr` function shown in Figure 1.10 to illustrate external dependencies, not all functions invoked in a source file will be present in the same file. These external function references must have a declaration before use; otherwise, the compiler makes assumptions about them that may not match with the actual definition. The declaration of a function before its definition is called a *forward declaration* or *prototype* and is generally placed in a header file. A *header file* is a source file that ends with a `.h` extension.

how tōō prō nouns′ it

Following their naming convention, header files are called *dot-h files*.

Because header files can be placed almost anywhere the author deems appropriate, use of the `-I` flag instructs `gcc` where to search for them.

Because header files can be used to prototype different functions from different libraries (packages), multiple `-I` flags may be passed to `gcc`.

```
bash[6]: gcc  -g  -ansi -Wall -c -I ../src/include -I /usr/X11R6/include gxArc.c
```

This example instructs `gcc` to look in two places, `../src/include` and `/usr/X11/R6/included`, for the header files that are included in the source file `gxArc.c`.

gcc -l

The `-l` flag tells `gcc` what extra libraries to include in the link phase of compiling. Using again the example of `gxColr` from Figure 1.10, not all functions that you invoke need to be in the source file where you use them. Further, they don't have to be in any source file that you author.

Groupings of functions for a general purpose need only be written once. After they are written, they may be included in a package or library for others to use.

For instance, if you want to find the square root of a number in C, you don't have to write the function to do it. Because it is such a common requirement, the authors of the C language have assembled a library of mathematical functions for your use. You can include in your source file the `math.h` header file which prototypes the square root function `sqrt`. Then, so the link phase of the compiling process knows how to resolve the external dependency, you must include the math library (`libm.a`) by use of the `-l` flag passed to the `gcc` command.

```
bash[7]: gcc gxArc.o gxGraphics.o -lm -o 2d-editor
```

EXCURSION

How to Determine Libraries Names for Use by `gcc`

The `gcc` command expects that libraries specified with the `-l` command follow the naming convention `lib`*`name`*`.a`.

This explains why, in the previous example, `gcc` understands `-lm` to be `libm.a` (the math library) containing the definition of the `sqrt` function.

As with `-I`, multiple `-l` flags may be necessary if you are calling functions from several different libraries.

```
bash[8]: gcc gxArc.o gxGraphics.o -lm -lXaw -lXt -lX11 -o 2d-editor
```

Based on your newly acquired knowledge of the `gcc` command, you should be able to determine what is happening in the previous example. The command `gcc` is being invoked with the files `gxArc.o` and `gxGraphics.o`. (As these are already object files, you must assume that at some point prior to this command `gcc` was invoked with the `-c` flag for each of the related source files `gxArc.c` and `gxGraphics.c`.) Further, `gcc` is being instructed to consider three libraries for resolving external dependencies (`libXaw.a`, `libXt.a`, and `libX11.a`). Lastly, the command specifies that the output should be named `2d-editor` (`-o`).

gcc -L

When instructing `gcc` to include libraries such as `libXt.a` or `libX11.a`, you must also tell `gcc` where to find them. Informing `gcc` of the location of libraries is done with the `-L` flag:

```
gcc gxArc.o gxGraphics.o -L /usr/local/lib -L /usr/X11R6/lib -lm \
-lXt -lX11 -o 2d-editor
```

Having only touched on the more commonly employed flags and parameters used with `gcc`, it is easy to see how cumbersome the command line could become as directives, header paths, and more and more source files are added to the project. Of course, needing to support multiple platforms where the locations of the libraries and header files differ and where the compiler flags vary would be almost impossible to do by invoking the `gcc` from the command line.

The `make` utility enables you to automate the invocation of the compiler (`gcc`) for every source file in your project. Further, `make` enables you to create dependencies or conditions that will force object files to be updated only when necessary. It also enables you to assemble the necessary parameter lists for the `gcc` command based on the operating system and platform being used. Only with the flexibility of `make` will you efficiently structure support for multi-platform development.

In general, `make` is a scripting language. It enables you to define variables, make decisions, and execute commands. The most obvious command for `make` to execute, of course, is the `cc` (or `gcc`) command. The uses of decisions within `make` support the task of multi-platform development. If you want the project to build under Linux *and* Solaris, it will be necessary to tell `make` the environment differences and let `make` decide which to employ based on the current platform. It is really not complicated, as you'll see with your first `make` script.

Makefile

When invoking the `make` utility, you must provide it with a configuration file. If no configuration file is specified on the command line, `make` looks for the presence of a default file called `Makefile`. If you are using the GNU `make` utility `gmake`, the default file will be named `GNUmakefile`.

Note

Just as GNU has its own C compiler, it also has its own `make` utility called `gmake`. Under Linux the `make` and `gmake` utilities are synonymous; however, other versions of UNIX have a different version of `make`. In other words, `gmake` is not part of the standard UNIX environment, but it can be added to any system.

Differences exist between GNU's `gmake` and the `make` utility available to other versions of UNIX. For instance, `gmake` will look for a configuration file called `GNUmakefile`. If `gmake` doesn't find a `GNUmakefile`, it will look for a `Makefile`. The `make` under UNIX, however, only considers `Makefile`. Also, `gmake` is a much more capable utility than the standard version of `make`. Either by employing the capabilities of `gmake` above `make`, or by choosing a configuration filename of `GNUmakefile`, you can impose use of `gmake` as a requirement for building your project. Because `gmake` is easily available from the Internet and is a superior `make` utility, I generally do require it.

When creating your `make` utility configuration file `GNUmakefile`, several areas must be addressed to construct it properly. Listing 1.1 contains the `GNUmakefile` that will be used to build the Graphics Editor project.

Examine the listing carefully, identifying the fields and components discussed during the coverage of the `gcc` command.

Listing 1.1 Graphics Editor Project `GNUmakefile`

```
# 2: # GNUmakefile for 2d-editor project
##

# Read in the system specific environment configuration
include ../make.defines

PROGRAM = 2d-gx
LIBS    = -lXaw -lXt -lX11 -lm

OBJS    = gxMain.o \
          gxGraphics.o \
          gxLine.o \
          gxText.o \
          gxArc.o \
          gxGx.o
```

continues

Listing 1.1 Continued

```


make-target: $(PROGRAM)


$(PROGRAM): $(OBJS)
    @echo "Building $(PROGRAM) for $(TARGET)..."
    @(CC) -g -o $(PROGRAM) $(OBJS) $(X11LIB) $(LIBS)
    @echo "Done"


#
# end of GNUmakefile
#
```

Comments

As with any program that you write, including comments is the courteous thing to do. Whether for your fellow team member who must follow behind you, or for yourself when returning to the code six months later, comments can be critical in effecting something quickly because the clues they provide guide the analysis of what is required.

In the `Makefile` syntax, a pound sign (`#`) is the comment token. The `make` utility ignores anything following a pound sign on a line.

Variables

Variables provide a means of managing the content of your `GNUmakefile` by enabling you to group and define items that are later assembled into more complex entities. Further, `make` configuration files grow in size and complexity proportionately to a project. Some elements of the `GNUmakefile` are repeated for varying conditions and command assembly. The use of variables allows for a single definition of an element that can be used multiple times throughout the file. With the use of variables, modifying an element does not require that you search and replace its every occurrence. Instead, you only have to change the assignment to the appropriate variable.

In Listing 1.1, variables are identified by the equal sign that separates the variable name from the variable value. By convention, variable names within a `Makefile` are capitalized. Following this convention in your own `make` configuration files will help others decipher what you write.

Notice that the variables defined in Listing 1.1 can have a value consisting of a single string, as in

```
PROGRAM = 2d-editor
```

or they can have a value of a list of strings

```
LIBS = -lXaw -lXt -lX11 -lm
```

The `make` utility expects the carriage return/line feed entered when you press Enter to serve as the end-of-line marker. When assigning values to variables, this end-of-line marker implies end of input. Therefore, values of variables consisting of multiple strings must be viewed as if they were contained on a single line. In the case of the variables `LIBS` or `CFLAGS` in the listing, this isn't a problem. However, the value of `OBJS` is moderately large and would wrap to multiple lines if we allowed it.

Wrapping multiple lines is syntactically correct because you haven't pressed Enter (interpreted as end of input); however, it reduces readability and makes it more difficult to edit the field later. To indicate to the `make` utility that the items within the value of `OBJS` are on a single line, delimit the carriage return/line feed with a backslash (`\`) character.

Delimiting instructs the `make` utility to ignore what immediately follows the backslash by not interpreting its meaning. The carriage return/line feed is still present, but it is not considered an end-of-line marker when preceded by the delimiter:

```
OBJS    = gxMain.o \
          gxGraphics.o \
          gxLine.o \
          gxText.o \
          gxArc.o \
          gxGx.o
```

You should now be comfortable with creating and assigning values to variables. To use a variable you've created, a unique syntax is necessary; this unique syntax allows the `make` utility to distinguish your intention clearly.

Employing a variable requires that it be prepended with the dollar sign (`$`). In this way, the `make` utility knows that you want to reference the value of the variable and not the name of the variable. Also, because the variable may have a value with nested spaces, you must surround the variable using parentheses to indicate the entire value should be treated as an entity. A reference to

```
$(LIBS)
```

is seen by the `make` utility as the value

```
-lXaw -lXt -X11 -lm
```

Targets

The use of targets within the `make` utility syntax enables you to have multiple entry points into the script. The power that this provides enables programmers to support multiple requirements from the same `GNUmakefile`.

For instance, compiling the project is a key function of the `make` utility and the reason you construct a `GNUmakefile`. Once the project is built, however, you may want to create a target within your `GNUmakefile` to install the project. You may want to move it from the user account level where only you (depending on permissions) can use it to a level where everyone on the system can benefit from it.

EXCURSION

You Know You're a Geek When…

The Towers of Hanoi is a classic programming problem within computer science used to demonstrate computing power and speed.

The legend is that there are monks in a cave somewhere with wooden discs stacked on one of four pegs. The discs decrease in size with the largest on the bottom and the smallest on top (see Figure 1.11). The monks must move the discs from a peg on one side to the peg on the other following these simple rules:

- Only one disc can be moved at a time
- A disc must always come to rest on a peg
- A larger disk can never rest on a smaller one

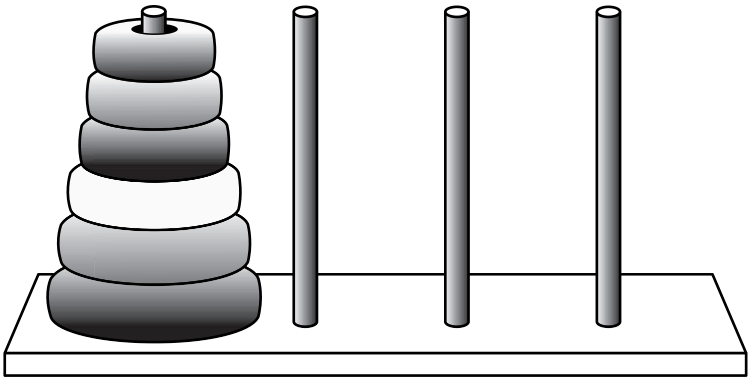


Figure 1.11
The Towers of Hanoi.

The telling of the legend ends by stating the number of discs that must be moved and identifying that once the task is complete, the world will end.

I once met someone who accomplished programming a solution to the Towers of Hanoi by using only the `make` utility syntax.

Two targets are defined in the `GNUmakefile` of Listing 1.1:

```
19: make-target: $(PROGRAM)
```

and

```
22: $(PROGRAM): $(OBJS)
```

Targets are identifiable by a label followed by a colon (`:`). The first target encountered by the `make` utility is considered the default target. When `make` is invoked, if no target is specified, it will begin at the first one it finds.

Issuing the command

```
bash[9]: gmake
```

causes the `make` utility to look for the file `GNUmakefile` and when found, to begin executing at the first target defined within the file.

Whereas, the command

```
bash [10]: gmake 2d-editor
```

causes the `make` utility to look for the target *`2d-editor`* within the `GNUmakefile` and begin execution there.

Remember, the `make` utility sees `$(PROGRAM)` as its value *`2d-editor`*.

The `make` utility considers anything on the same line, following the target label, to be a *dependency* of the target. Because a dependency must be resolved first, `make` evaluates all dependencies before evaluating the body of the target. Based on what the dependency is the evaluation may vary.

For instance, in Listing 1.1, a dependency to `$(PROGRAM)` exists for the target `make-target`.

```
19: make-target: $(PROGRAM)
```

When the `make` utility begins to evaluate `$(PROGRAM)` as a dependency to `make-target`, it sees that `$(PROGRAM)` has dependencies as well.

```
22: $(PROGRAM): $(OBJS)
```

The dependency declared for `$(PROGRAM)` is `$(OBJS)`. Because `$(OBJS)` is not a target defined within this `GNUmakefile`, the `make` utility uses built-in rules to evaluate `$(OBJS)`.

EXCURSION

Generating Object Files from the `make` *Utility's Internal Rule Set*

The built-in rule governing object files is in fact a target internal to the `make` utility. The target uses the base name of the object files as expanded from the `$(OBJS)` variable. Effectively it says, in order to get a `.o` (*dot-oh*), look for a `.c` (*dot-see*) of the same *base name* and run `gcc -c` against it. If a `.o` already exists, the time stamp of the associated `.c` is compared and `gcc -c` is run only if the `.c` is newer than the existing `.o`. Whew, what a mouthful!

A target *body* begins on the line immediately following the target label. Each line of the body is prefaced with at least one tab character. The body ends when make finds a line not beginning with a tab.

Note

The use of tabs and spaces can be confusing to a new programmer. The places where make syntax allows a space or expects a tab varies, depending on which part of the make configuration file you are in. For instance, when writing a target body, a character that is not a tab as the first character on a line is seen as terminating the body. However, when assigning a value to a variable and delimiting the end-of-line marker so the value may span multiple lines, consecutive lines may begin either with a space or a tab.

In following the flow of make in Listing 1.1, you see that when make-target is called, the dependency $(PROGRAM) is immediately evaluated. (As make-target does not have a body, you can assume that its sole purpose is to ensure that $(PROGRAM) is evaluated.) When the make utility evaluates $(PROGRAM), it sees that $(PROGRAM) depends on $(OBJS). The make utility then invokes its internal rule to create the files specified in the value of $(OBJS). When the files in $(OBJS) have been created (or updated), make is ready to evaluate the body of the $(PROGRAM) target.

The first line of the $(PROGRAM) target body is the echo command.

```
23:     @echo "Building $(PROGRAM) for $(TARGET)..."
24:     @(CC) -g -o $(PROGRAM) $(OBJS) $(X11LIB) $(LIBS)
25:     @echo "Done"
26:
```

A standard UNIX command, echo simply prints everything that follows on the same line to the screen. Notice that in the $(PROGRAM) target body, the echo command is preceded by the @ sign.

```
23:     @echo "Building $(PROGRAM) for $(TARGET)..."
```

The @ symbol before a command tells make not to print the command to the screen before executing it. When used, the @ symbol forces make to be silent and simply run the command.

Continuing with the $(PROGRAM) target body, line 24 instructs make how to assemble the cc command for the last phase of compiling called the link phase.

```
24:     @(CC) -g -o $(PROGRAM) $(OBJS) $(X11LIB) $(LIBS)
```

Remember that the $(OBJS) dependency to $(PROGRAM) ensured that all the .o files were constructed or updated. The last step is to link all the object files together and resolve any external dependencies.

In assembling the `cc` command, some variables are used that you haven't yet seen (`CC`, `TARGET`). To discuss them, we'll look at an internal `make` utility command called `include`.

`include`

With large-scale, professional-level development, source files will almost certainly be grouped by common purpose and located in separate directories from source files of differing purposes. Each directory will have its own `make` configuration file because the object file list (`$(OBJ)`) will differ for each directory, as can the external dependencies or library considerations.

When faced with a project structure of many directories or multiple `GNUmakefiles`, it is not necessary to repeat variable declarations for each of the `make` files within your project. Elements common to the entire project can be placed at a central location and then included with the `include` command into each of the project's `make` files.

Note

The term *make file* refers to `GNUmakefile` or `Makefile` (or any `make` configuration file). If you don't use one of the default `make` filenames (`GNUmakefile` or `Makefile`), `make` must be told the name of the file. This is done by specifying the `-f` flag:

```
bash[12]: gmake -f SomeMakeFileName
```

Refer to Figure 1.6, Graphics Editor project layout, and notice the file `make.defines` in the `2d-editor` directory. This is the same `make.defines` file included by the `GNUmakefile` in Listing 1.1.

```
# Read in the system specific environment configuration
include ../make.defines
```

It is the common definition file for variables and functions global to the entire project.

When you specify a file for inclusion into a `make` file, the `make` utility inserts it exactly as if it were typed in at the place of the `include` command. There is no limit to the number of files that can be included into a `make` file, nor is there a limit on the number of times a file can be included.

`make.defines`

Analyzing the `make.defines` file shown in Listing 1.2 gives you a lot to consider. However, as we break it into parts, you'll see how easy it is to understand. Remembering previous discussions of the `gcc` command will help identify components of the `make.defines` file that satisfy the required elements of `gcc`.

Listing 1.2 The `make.defines` for the Graphics Editor Project

```
###
# make.defines
#
# Included in each make file used with the 2d Editor Project
#
# Dependencies on the following environment variables:
#  TARGET = machine-os
#
# The syntax of this file is for use
# with 'gmake' (GNU version of make)
###
TARGET_SPARC_SUNOS  = sun4u-SunOS
TARGET_i86_SOLARIS  = i86pc-SolarisOS
TARGET_i86_LINUX    = i86-Linux

ifdef GxHOME
   GxSRCDIR = ${GxHOME}/src
else
   GxSRCDIR = ../src
endif

vpath %.h ${GxSRCDIR}/include
vpath %.c ${GxSRCDIR}

#
# Configure for Linux running on a PC
#
ifeq ($(TARGET),$(TARGET_i86_LINUX))
    X11INC    = -I/usr/include/X11
    X11LIB    = -L/usr/X11R6/lib
    INCS      = -I${GxSRCDIR}/include

    CC        = gcc
    OPTS      = -ansi -Wall -g
endif

#
# Configure for Solaris running on a Sparc
#
ifeq ($(TARGET),$(TARGET_SPARC_SUNOS))
    X11INC    = -I/usr/openwin/include
    X11LIB    = -L/usr/openwin/lib
    INCS      = -I${GxSRCDIR}/include

    CC        = gcc
    OPTS      = -g -Wall -ansi
endif

#
# Configure for Solaris running on a PC
```

continues

Listing 1.2 Continued

```
50: #
51: ifeq ($(TARGET),$(TARGET_i86_SOLARIS))
52:     X11INC    = -I/usr/openwin/include
53:     X11LIB    = -L/usr/openwin/lib
54:     INCS      = -I${GxSRCDIR}/include
55:
56:     CC        = gcc
57:     OPTS      = -ansi -Wall -g
58: endif
59:
60: #
61: # Force all Makefiles using this file to check the configuration
62: # of the environment before building the target
63: #
64: all: make-env-check make-target
65:
66: #
67: # Check to environment variables need to build are set
68: #
69: make-env-check:
70:     ifndef TARGET
71:         @echo
72:         @echo "TARGET not defined!"
73:         @echo "Set environment variable TARGET to:"
74:         @echo "  sun4u-Sun0s"
75:         @echo "  i86pc-Sun0s"
76:         @echo "  i86-Linux"
77:         @echo
78:         @exit 1
79:     endif
80:
81: clean:
82:     @rm -f *~ *.o $(PROGRAM)
83: #
84: # end of make.defines
85: #
```

Identifying the pound sign (#) as the comment token in `make` file syntax enables us to skip down to line 11 of Listing 1.2 to begin the discussion.

```
11: TARGET_SPARC_SUNOS   = sun4u-SunOS
12: TARGET_i86_SOLARIS   = i86pc-SolarisOS
13: TARGET_i86_LINUX     = i86-Linux
```

Beyond the comments, the file begins with a series of variable declarations. Because you should now be comfortable with declaring variables and assigning them values, this is review.

The variables `TARGET_SPARC_SUNOS`, `TARGET_i86_SOLARIS`, and `TARGET_i86_LINUX` are an anticipation of the platform and operating system combinations supported in this project. As you look through the `make.defines` file, you see that each of these variables is compared to another variable called `TARGET`. Consider the following lines:

```
27: ifeq ($(TARGET),$(TARGET_i86_LINUX))

39: ifeq ($(TARGET),$(TARGET_SPARC_SUNOS))

51: ifeq ($(TARGET),$(TARGET_i86_SOLARIS))
```

This is consistent with the comment at the beginning of the `make.defines` file:

```
6:  # Dependencies on the following environment variables:
7:  #  TARGET = machine-os
```

The environment variable `TARGET` must be set for the `make.defines` file to work. Fortunately, `make` file syntax enables a test of this dependency.

EXCURSION

Promoting a Variable to an Environment Variable

An *environment variable* is much like a variable set within a `make` file. However, the variables set within a `make` file are visible only to the `make` utility. The issue of visibility is known as *scope* and will be discussed at length in the next chapter. In the meantime, any of the variables that you've defined in the `GNUmakefile` can be set in your *environment*. If removed from the `make` file and placed in your environment, the variable is available (visible) to every command and utility and is then called an environment variable.

Some environment variables are provided by the operating system. For instance, to see what command shell you are running, `echo` the environment variable `SHELL`:

```
    bash[13]: echo $SHELL
```

To see all environment variables currently set, use the `env` command. Notice the environment variable `path` in the `env` output. The `path` variable informs the command shell of all the places to search for external commands.

➔ (See Excursion "What Is a Command Shell" under section on "The `mkdir` Command" for a description of internal verses external commands, page 11.

We don't want just any variable set as an environment variable, so choose the variables carefully in order to not clutter the environment.

Knowing which command shell you are currently using will enable you to use the correct syntax for setting an environment variable called `TARGET`. Table 1.3 is a listing of these shells and syntax.

Table 1.3 Environment Variable Syntax

Shells	*Syntax*
`bash`, `sh`, and `ksh`	`export TARGET=i86-Linux`
`csh` and `tcsh`	`setenv TARGET i86-Linux`

As discussed previously, the first target (a label followed by a colon) which the `make` utility encounters is the default; it is executed if no target is specified on the command line.

Notice from Listing 1.1 that the `include` command is the first line (beyond comments) of the `GNUmakefile`. Therefore, any target found in `make.defines` precedes any target found in the `GNUmakefile`. As you look through `make.defines`, notice the target called `all`.

```
all: make-env-check make-target
```

This target has two dependencies that are also targets. The first to be evaluated is the `make-env-check` target, which ensures that the `TARGET` environment variable is set. It does this with the keyword `ifndef`, which stands for *if not defined*. The `ifndef` must be followed by a variable name so that the `make` utility knows what to test for definition. Because the `ifndef` implies a conditional body (what to do if not defined), the body must have an end. The `ifndef` end-of-body marker is the keyword `endif`.

Note

To create conditional bodies in the case that a variable *is* defined, use the `ifdef` keyword. It is used exactly like the `ifndef` keyword, except that the body is acted on when the variable exists in the environment instead of when it doesn't.

If the `make-env-check` completes successfully (meaning the `TARGET` variable is defined), the `make` utility continues on to evaluate the `make-target` dependency of the `all:` target. The `make-target`, as seen in Listing 1.1, is contained in the `GNUmakefile`. The target's sole purpose is to ensure that the `$(PROGRAM)` target executes.

Surprisingly, we are already through a good portion of the `make.defines` file. Having discussed the initial variables, the target `all:`, and the dependencies `make-env-check` and `make-target`, we are ready to look at the test of `GxHOME`.

```
ifdef GxHOME
    GxSRCDIR = ${GxHOME}/src
else
    GxSRCDIR = ../src
endif
```

The `GxHOME` variable ensures that the `GxSRCDIR` variable is set correctly. The `GxSRCDIR` variable enables us to store the source files and object files in different places. This flexibility is crucial in multi-platform development because every platform will need to generate private copies of the object and executable files.

Note

If necessary, review the section "Object Files" for a description of object file incompatibility between platforms, page 19.

Refer to Figure 1.6 where the `i86-Linux` directory for storing object files is separate from the `src` directory. Similarly, you could have an independent directory for every platform or operating system you intend to support, as illustrated in Figure 1.12.

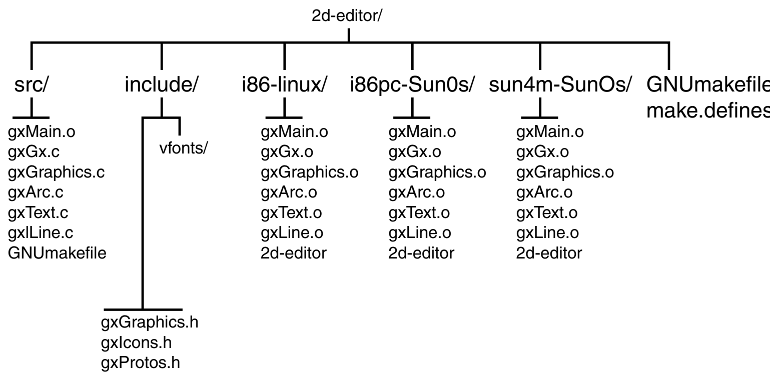


Figure 1.12
Project structure with multiplatform support.

EXCURSION

Building a Project for Multiple Architecture

The `make` file structure we've evaluated for the Graphics Editor project will support the platforms shown in Figure 1.12. Through the use of the `TARGET` environment variable and its successful comparison to one of the environments defined in the `make.defines` file

```
11: TARGET_SPARC_SUNOS      = sun4u-SunOS
12: TARGET_i86_SOLARIS      = i86pc-SunOS
13: TARGET_i86_LINUX        = i86-Linux
```

the project could be built successfully for any of these.

To add platform support for other environments, an additional `TARGET_MACHINE_OS` variable could be added to `make.defines` with a corresponding `ifeq` body to set the necessary variables to the values correct for the new platform.

The separation of object files from source files is possible through the use of the `GxSRCDIR` variable. After the location of the source files is determined, the variable `vpath` can be set for the `.h` and `.c` files of the project.

The `vpath` variable in the `make.defines` works much like the `path` variable in your environment. Just as the `path` variable instructs the command shell where to search for commands, the `vpath` variable tells the `make` utility where to search for source files.

➔ See Excursion "Promoting a Variable to an Environment Variable" for a description of the `path` variable, page 40.

The `vpath` variable informs the `make` utility where to look when it needs to find a `.c` (represented as `%.c`) or a `.h` (`%.h`) file.

```
vpath %.h ${GxSRCDIR}/include
vpath %.c ${GxSRCDIR}
```

This way, the `.c` and `.h` files that `make` needs do not have to be present in the same directory where the `make` command is executed.

By creating a directory of the same name as the `TARGET` value, you can issue the `make` command from within this directory having the results stored in this platform-specific directory separated from the source code.

One last detail necessary to make this separation of object files and source files work is the fact that the `GNUmakefile` must be visible in each of the object directories. It is the `GNUmakefile` and its inclusion of `make.defines` that establishes the necessary targets, defines the `$(OBJS)` file list, and assigns the pertinent `vpath` values.

To accomplish having the `GNUmakefile` visible to each of the object directories, you could use the `cp` (copy) command and place a copy of the `GNUmakefile` in each directory needing it. But what happens when you need to modify the `GNUmakefile`? You would either have to repeat the modification in every place where you have a copy of the file (supposing you could remember them all) or you'd have to re-copy the file to every directory.

In UNIX there is a way to make a single file visible in multiple places. Changing the file at any place where it is visible is the same as changing it at its actual location. The mechanism for making this possible is called a *symbolic link*.

The `ln -s` Command

A symbolic link can be a pointer to a file or a directory. The power of symbolic links is that they are treated exactly like what they point to. In the case of files, you can edit them with the changes getting stored in the actual file.

To create a symbolic link, use the following command:

```
bash[14]: ln -s /home/jbrown/2d-editor/src/GNUmakefile  \
/home/jbrown/2d-editor/i86-Linux/
```

This places a pointer to `GNUmakefile` in the `i86-Linux` directory. You can then change (`cd`) to the `i86-Linux` directory and execute the `gmake` command. The command will find the `GNUmakefile` as if it were located there.

Note

You can use the link command (`ln`) against directories as well as files. The syntax for the command supports relative or explicit paths. The basic syntax for creating a symbolic link follows the form

```
ln -s what where
```

Specifying what you want to link (*`what`*) is always necessary. However, if the `where` is the current directory maintaining the same name as *`what`*, you can omit the `where` designator. As always, see the man page for a complete description of the `ln` command.

You've now completed a close look at the elements required to create, edit, and build a project under the Linux operating system. However, other commands and utilities are available under Linux that are not required but will give you an advantage when accomplishing your development tasks.

System Tools and Useful Commands

Linux is a complex operating system that puts literally hundreds of commands at your disposal. Learning them all will take time, investigation, and practice. Just as a musician always feels someone is more accomplished, you'll never feel like you've learned everything. If ever you feel you know it all, you are probably consulting.

The following section discusses some of the more common Linux commands used in software development.

`grep`, Pipes, Redirection, and `more`

The `grep` command searches files for occurrences of regular expressions. Following this basic syntax forms the command

```
bash[15]: grep [flags] string files(s)
```

When a match is found, `grep` returns (prints to the screen) the entire line that contained the match. Use of the `-n` flag helps in reading the output because it instructs `grep` to include the line numbers where the matches were found.

As mentioned earlier, UNIX is case sensitive. This sensitivity extends to the `grep` command as well. However, unlike UNIX you may instruct `grep` to suppress this and perform caseless searches. Passing a `-i` flag to `grep` instructs that case should be ignored.

A common command used in a development environment is

```
bash[16]: grep -i something *.[ch]
```

that requests `grep` to perform a caseless search (`-i`) on all `.c` and `.h` files for *`something`*.

This command is also useful when using grep is filtering out what you don't want to see. This can be done with an *inverse search* (-v flag) in which you tell grep what not to print, or in other words, what to ignore.

An example of when an inverse search would be useful is in searching for the string man where grep returns multiple lines from the input that includes the string manual. To instruct grep not to return the lines containing manual, you would issue the following command:

```
bash[17]: grep man myfile.txt | grep -v manual
```

Very important to this example is the symbol linking the output of the first grep command to the input of the second. This symbol is called a *pipe* (|), and it is an extremely powerful feature of UNIX because it provides a way of chaining commands together.

The result of the preceding example is that grep searches *myfile.txt* for every occurrence of the string *man*. As I pointed out, the output also contained multiple lines with the string *manual*, which was undesirable. So the output of the first grep is made the input of the second grep by use of the pipe, allowing an inverse search of the string *manual* on the output of the first command. The inverse search removes any line from output of the first grep containing the string *manual*. What you are left with (hopefully) following the second grep are only occurrences of the string *man*.

Now that you're comfortable with the use of the grep command, let's focus a moment longer on the pipe feature of UNIX.

The capability to take the output of one command and make it the input of another is called *redirection*. Through redirection, the pipe symbol (|) can chain together two commands, as seen in the preceding example.

A very useful application of redirection is the more command. When the output of a command or file is too large for the window, you can ask UNIX to break it into window-sized pieces.

```
bash[19]: ls -l | more
```

or

```
bash[20]: more myfile.c
```

It is also possible to redirect the output of a command to a file. The > (greater than) symbol works much like a pipe, but instead of sending the output of one command to another, it sends the output to a file. If the file doesn't exist, it will be created.

```
bash[18]: grep man myfile.txt > grep.output
```

If the file already exists, its contents will be overwritten with the results of this command. To ensure that nothing is lost, you could redirect the output of `grep` so that it appends the file by using >> (two greater than signs):

```
bash[19]: grep man myfile.txt >> grep.output
```

A common way that developers use the power of redirection is to save the output of the `gmake` command so that reported errors are recorded in a file:

```
bash[20]: gmake > make.out
```

The `find` Command

The `find` command allows for searching a directory structure for just about anything. With the `find` command, you can search for a file or directory of a specific name, a file of a certain extension, a file or directory of specific permissions, or a given modification date.

The more powerful a command, the more complex its assembly and execution, as is true with the `find` command.

The basic structure of the `find` command follows the syntax

```
find starting point condition value action
```

From *`starting point`*, the `find` command searches through the entire directory tree for a *`condition`* of some *`value`*, and when satisfied it executes the *`action`*.

The `find` command requires that you provide a *`starting point`* that is a valid directory in your file system.

The *`conditions`* which `find` understands are numerous. The simplest is `-name`. As given in the command syntax, a *`condition`* must have a *`value`*. The `-name` needs the name of a file or directory to search for:

```
find /home -name readme.txt ...
```

Wildcards can be used to specify the *`value`* for the name condition; however, when using wildcards the *`value`* clause must be contained in double quotes (`"`). Otherwise, the command shell will interpret the wildcards, and the `find` command will never see them.

```
find /home -name "*read*" ...
```

Knowing that `find`, like all of UNIX, is case sensitive, you must explicitly assign the value string to the characters you want find:

```
find /home -name "*[Rr][Ee][Aa][Dd]*" ...
```

This would find *README* as well as *readme* or any combination of the two (such as *ReaDme*).

Having properly formed the *starting point* for the find command, the *condition* and *value* to search for, the only remaining element is instructing find what to do when the search is successful.

Current UNIX versions will automatically perform the -print action if no other action is specified.

```
bash[20]: find /home -name "*[Rr][Ee][Aa][Dd]*" -print
```

The find command understands several other actions. One very useful for software developers is the -exec command. This instructs find to execute a command against the matches found. Clearly, in conjunction with -exec, you must tell find what command to execute.

```
find /home -name "*.[ch]" -exec grep MyFunc {} \;
```

Looking closely at this example, you see everything discussed.

This illustrates the find command with a *starting point* of /home, a *condition* of -name, a condition *value* of "*.[ch]" (all .c and .h files), an *action* of -exec (execute a command), and the command of choice being grep. Finally, grep is searching the lines returned by find for any occurrence of the string *MyFunc*.

To represent the matches returned by find to the grep command, use {} (open and close curly braces). It is also necessary to end the command portion of -exec, which is done with a semicolon (;). However, you must hide the semicolon from the command shell by using the delimiter I spoke of earlier. This is necessary because the semicolon has meaning to the shell as well as the find command but is meant to inform find that the command portion of -exec is finished.

Next Steps

With a few Linux commands under your belt, an understanding of the compiling and linking processes, and an introduction to the vi editor for creating source files, you are ready to begin looking at programming constructs. Chapter 2, “Programming Constructs,” will lead you through structuring solutions to software problems by applying common programming constructs and logic processes.

Chapter 2

In this chapter

Programming Constructs

Learning to properly define the problem to be solved in a software application is an unavoidable first step of computer programming. The second step, applying the correct construct to effect the solution, lies at the heart of software engineering.

Identifying and solving a problem at this level is different from the role of a systems analyst in the software development process. Systems analysts are concerned with the design method, feature list, and milestones of an application.

As a software engineer, you are continually challenged with accomplishing the next portion of the application. Perhaps the task is to read a configuration file and initialize a variable set accordingly. Instantly, you should begin thinking about loops, decisions, functions, and data.

This chapter focuses on modeling problems by the programming construct most often employed to solve them. Learning syntax and convention is required to add a new computer language to your skill set; however, programming constructs are constant throughout computer programming. As the boss likes to say, “No one is inventing a new bubble sort.” This means, of course, that there is an appropriate way (construct) for solving specific problems when writing software. Let us review those now.

EXCURSION

Do People Really Sort Bubbles?

A *bubble sort* is a method of ordering data with neighboring data elements in a list compared as the list is traversed. If the element closer to the beginning of the list is greater than its neighbor, the elements are swapped. After multiple iteration, the smaller values bubble to the top (beginning of the list). The list is sorted when nothing is swapped during a traversal of the list.

A *programming construct* is the basic element used in computer problem solving. It is a method, practice, or means of accomplishing a task or representing data.

Knowing when (and how) to apply a construct properly is the challenge of learning computer programming. You add new languages to your abilities by determining how to accomplish decisions, looping, or function calls using this new language.

The following sections focus on creating a foundation of basic programming constructs. If you are already comfortable with programming constructs, feel free to continue to the next chapter.

Decisions

A decision tree begins with a question where the answer may present further questions or an immediate end to the tree.

EXCURSION

Do You Have Apple Pie?

"Great, I'd like apple pie a la mode with vanilla ice cream, and I'd like the pie heated. If you don't have vanilla, I'd like whipped cream but only if it is fresh. If it is not fresh, then nothing. Only the pie, but then not heated."

("When Harry Met Sally," Metro Goldwyn Mayer, 1989)

Figure 2.1 is an example of how to approach decisions, as each part of the process has significant impact on what happens next. Failure on the part of the programmer to account for a choice or condition required by the user can result in the application having a short life span.

Decisions create execution branches, control loop conditions, and dictate overall program flow. With the advent of object-oriented and event-driven programming methods, the model of purely sequential program execution was replaced. New methodologies complicate decision trees, because applications must be resilient enough to account for any circumstance at any moment.

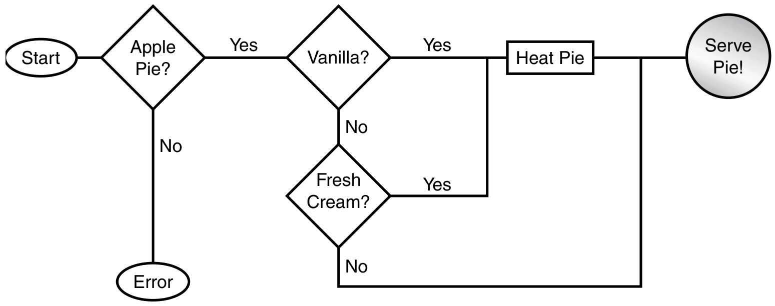


Figure 2.1

Sample decision tree.

Certainly, there is more to applying methodologies than simply making a decision. However, the decision construct controls the state of an application, which in turn determines when actions are valid.

EXCURSION

Creating a State Machine from a Decision Tree

Referring again to Figure 2.1, each *yes* branch could be labeled with a number to indicate the tree's state. Clearly, the action of heating the pie is only valid if we reach the state where it is allowed. This is a function of the decision construct.

Because the Graphics Editor project is X Window-based, event-driven application development will be looked at closely in Chapter 4, "Windowing Concepts." The decision construct is crucial to managing the state of an application, as it controls how a program unfolds and determines when actions are allowed and, often, what action is required.

As illustrated in Figure 2.1, a basic decision consists of a true or false condition. Does the apple pie exist? Is the ice cream vanilla? Is the whipped cream fresh? Each condition is tested and program execution follows one of two branches representing a true action or a false action.

No matter the complexity of the decision tree or means of representing the decision process within the programming language, it can be reduced to this level of simplicity.

The `if` Statement

The `if` statement represents a basic decision. The syntax of `if` follows the pattern

```
  if condition
action
```

The `if` statement always performs the *`action`* based on an evaluation of the terms of the *`condition`* resulting in `true`.

geek speak

The evaluation of a condition in the C programming language results in `true` for any non-zero value. A condition is considered `false` only if it equals zero.

You could be more precise by explicitly testing for the `true`.

```
if condition equals true
  action
```

Notice that when the condition being tested is negated, `if` still takes the action based on successful evaluation of the terms of the *`condition`*:

```
if condition equals false
  action
```

In other words, when it is true that the *`condition`* is `false`, `if` performs the *`action`*.

EXCURSION

A Stylistic Note…

Programmers commonly put the negative condition before the positive, directing to test for the negated or *`condition equals false`* before testing for the `true`. Although semantically and syntactically correct, it is less readable for anyone who might follow behind.

The `else` Statement

As demonstrated, `if` executes the *`action`* based on a true evaluation of the terms of a *`condition`*. To branch to an action based on an implied false evaluation, use the `else` statement.

The `else` statement must be preceded by an `if` and immediately follow the *`action`* of the `if`:

```
if condition
  action
else
  another action
```

Types of Conditions

Let us focus for a moment on what is referred to as the *`condition`* being evaluated by the `if` statement. Because conditions depend upon the data being evaluated, it will be necessary to look at each type separately.

Numbers

When forming test conditions for decisions, a variable representing a number can be evaluated in many ways. For instance, evaluation can determine if the number is equal to zero:

```
if ( number == 0 )
  action
```

Note

Notice that I've grouped the condition using parentheses. This representation is more readable and helps you to start thinking in terms of the C language syntax. Remember, the focus here is on the constructs; representing them using C syntax will begin the introduction of Chapter 3, "A Word on C," where the focus is the C programming language.

A variable representing a number can also be tested for being less than or greater than zero:

```
if( number < 0 )
  action
```

or

```
if( number > 0 )
  action
```

Further, it can be tested for being within a range:

```
if( number < 20 && number > 10 )
  action
```

This example tests to see whether `number` is less than `20` *and* greater than `10`.

Table 2.1 shows the symbols understood by the C language for forming test conditions on numbers. Again, C syntax is not the current focus but is useful because a means of representing the conditions is needed for our current discussion.

Table 2.1 Symbols for Numeric Conditions

Symbol	*Meaning*	*Example*
>	greater than	if(variable > value)
<	less than	if(variable < value)
>=	greater than or equal to	if(variable >= value)
<=	less than or equal to	if(variable <= value)
==	equal to	if(variable == value)
!=	not equal to	if(variable != value

continues

Table 2.1 Continued

Symbol	*Meaning*	*Example*
&&	and	if(condition && condition)
||	or	if(condition || condition)
!	not	if(!condition) equivalent to if(condition == false)

Characters

Conditions created using variables representing characters follow the same rules and employ the same symbols as numbers. This is true because, as we discussed in Chapter 1 (see the section "The C Compiler," page 18), the computer does not understand the letter *A*. The representation of a letter in a manner that the computer can understand is as a number.

The letter *A*, as seen in Table 1.2, is the same as the number 65 when it is compiled into machine language, allowing the same conventions used with numbers to be applied to characters when forming test conditions.

To illustrate this, consider the following conditions:

```
    if( character > 'A' )
```

or

```
    if( character < 'Z' )
```

or

```
    if( character > 'A'  && character < 'Z' )
```

Note

Treating characters like numbers requires you to know the proper order in which the language places them. This ensures that the semantics of a condition are as intended.

Because the letter *A* is a decimal 65, *Z* is 65+25 or 90 because the letter *Z* is 25 places away from the letter *A*. But where do *a* through *z* fit in? Is *a* less than or greater than *A*? Also, what about other printable characters that are not letters, such as punctuation?

A standard known as *ASCII* defines the representation of characters as numbers for manipulation by computers.

To see the entire ASCII table of characters and where all printable (and non-printable) characters fit in to it, use the `man` command:

```
        bash[20]: man ascii
```

Figure 2.2 shows a portion of the man page for the ASCII table.

EXCURSION

ASCII (American Standard Code for Information Interchange)

ASCII is the most common format for text files in computers and on the Internet.

In an ASCII file, each alphabetic, numeric, or special character is represented with a 7-bit binary number (a string of seven zeros or ones). ASCII defines 128 possible characters.

UNIX- and DOS-based operating systems (except for Windows NT) use ASCII for text files. Windows NT uses a newer code, *Unicode*. IBM's System 390 servers use a proprietary 8-bit code called EBCDIC.

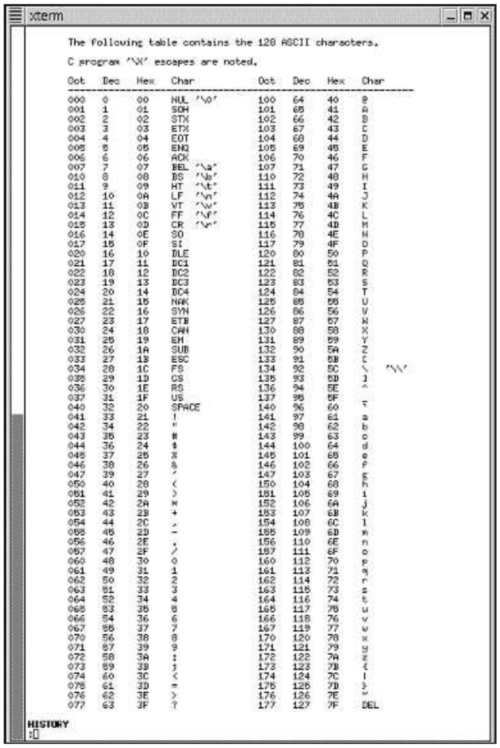
Conversion programs enable different operating systems to change a file from one code to another. The American National Standards Institute (ANSI) developed ASCII.

how tōō prō nouns′ it

The term ASCII is pronounced as if it were spelled *ask-e*.

Figure 2.2

ASCII Table `man ascii`.



```
xterm

The following table contains the 128 ASCII characters.

C program '\X' escapes are noted.

Oct   Dec   Hex   Char               Oct   Dec   Hex   Char
------------------------------------------------------------------
000   0     00    NUL  '\0'          100   64    40    @
001   1     01    SOH                101   65    41    A
002   2     02    STX                102   66    42    B
003   3     03    ETX                103   67    43    C
004   4     04    EOT                104   68    44    D
005   5     05    ENQ                105   69    45    E
006   6     06    ACK                106   70    46    F
007   7     07    BEL  '\a'          107   71    47    G
010   8     08    BS   '\b'          110   72    48    H
011   9     09    HT   '\t'          111   73    49    I
012   10    0A    LF   '\n'          112   74    4A    J
013   11    0B    VT   '\v'          113   75    4B    K
014   12    0C    FF   '\f'          114   76    4C    L
015   13    0D    CR   '\r'          115   77    4D    M
016   14    0E    SO                 116   78    4E    N
017   15    0F    SI                 117   79    4F    O
020   16    10    DLE                120   80    50    P
021   17    11    DC1                121   81    51    Q
022   18    12    DC2                122   82    52    R
023   19    13    DC3                123   83    53    S
024   20    14    DC4                124   84    54    T
025   21    15    NAK                125   85    55    U
026   22    16    SYN                126   86    56    V
027   23    17    ETB                127   87    57    W
030   24    18    CAN                130   88    58    X
031   25    19    EM                 131   89    59    Y
032   26    1A    SUB                132   90    5A    Z
033   27    1B    ESC                133   91    5B    [
034   28    1C    FS                 134   92    5C    \   '\\'
035   29    1D    GS                 135   93    5D    ]
036   30    1E    RS                 136   94    5E    ^
037   31    1F    US                 137   95    5F    _
040   32    20    SPACE              140   96    60    `
041   33    21    !                  141   97    61    a
042   34    22    "                  142   98    62    b
043   35    23    #                  143   99    63    c
044   36    24    $                  144   100   64    d
045   37    25    %                  145   101   65    e
046   38    26    &                  146   102   66    f
047   39    27    '                  147   103   67    g
050   40    28    (                  150   104   68    h
051   41    29    )                  151   105   69    i
052   42    2A    *                  152   106   6A    j
053   43    2B    +                  153   107   6B    k
054   44    2C    ,                  154   108   6C    l
055   45    2D    -                  155   109   6D    m
056   46    2E    .                  156   110   6E    n
057   47    2F    /                  157   111   6F    o
060   48    30    0                  160   112   70    p
061   49    31    1                  161   113   71    q
062   50    32    2                  162   114   72    r
063   51    33    3                  163   115   73    s
064   52    34    4                  164   116   74    t
065   53    35    5                  165   117   75    u
066   54    36    6                  166   118   76    v
067   55    37    7                  167   119   77    w
070   56    38    8                  170   120   78    x
071   57    39    9                  171   121   79    y
072   58    3A    :                  172   122   7A    z
073   59    3B    ;                  173   123   7B    {
074   60    3C    <                  174   124   7C    |
075   61    3D    =                  175   125   7D    }
076   62    3E    >                  176   126   7E    ~
077   63    3F    ?                  177   127   7F    DEL

HISTORY
:
```

Strings

Strings differ from characters in that they are *groupings* of characters. Although this might seem too obvious to point out, manipulation of strings requires a completely new heuristic.

The ASCII table does not apply to groups of characters. If the test condition considers the individual characters comprising a string, it is okay to apply what you've learned. However, when evaluating the group of characters as an entity, you must use other methods.

Note

Strings do not employ the same symbols for comparison as numbers or characters. To do so requires that a decimal value be acquired for a string; however, very different strings could result in the same numeric value. For instance, when adding the ASCII values for each of the letters in the following strings, the same value results:

```
FIG = 70 + 73 + 71 = 214
```

and

```
SAB = 83 + 65 + 66 = 214
```

The means or mechanisms used to evaluate strings are specific to a computer language. In the C programming language, an entire library is dedicated to string manipulation and testing. The C string library is covered in Chapter 3; however, here we need an understanding of test conditions for strings and specifically the concept of *lexical analysis*.

The lexical analysis of strings results in one string being evaluated as less than, greater than, equal to, or not equal to another.

Everyone, at some point, has used a dictionary to look up a word he didn't know how to spell; we are all comfortable with the idea that strings are ordered. The lexical analysis of two strings seeks to determine whether one string comes before or after another in the dictionary. If one string precedes another, it is said to be less than the second string. If a string appears after another in the dictionary, it is considered greater than. The concepts of strings being equal or not equal should be immediately apparent.

As mentioned, string manipulation is language specific; therefore, examples of how to form test conditions for strings are detailed in Chapter 3.

With a clearer understanding of test conditions for different data types, you are ready to form the construct to support multiple test conditions.

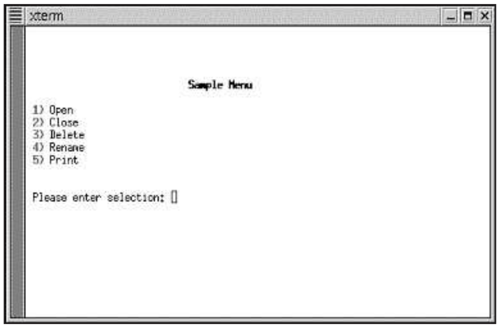
The `if else` Statement

So far, we've only considered a condition of true (`if`) and an implied false (`else`).

What if the condition you are testing has one of many possible values? For instance, what if the user has selected an option from a menu of several choices, as shown in Figure 2.3?

Figure 2.3

Sample menu.



```
                    Sample Menu

1) Open
2) Close
3) Delete
4) Rename
5) Print

Please enter selection: []
```

Testing the user response requires an equal number of tests as choices.

To test for all the possible inputs, you could use a series of `if` statements:

```
if( input == '1' )
  open action
if( input == '2' )
  close action
if( input == '3' )
  delete action
if( input == '4' )
  rename action
if( input == '5' )
  print action
```

The shortcoming of this approach, however, speaks not only to performance but also to the lack of a default condition in the case where the user misses the number `5` key and presses `6`.

Simply adding an `else` to the list as in

```
if( input == '1' )
  open action
if( input == '2' )
  close action
if( input == '3' )
  delete action
if( input == '4' )
  rename action
```

```
    if( input == '5' )
      print action
    else
      error action
```

is not valid because the *error action* is taken not only when the user enters an invalid entry, but also when he enters any valid response other than 5. This, of course, ignores the fact that every if statement is evaluated regardless of the input or when a match is found.

Use of the else if statement resolves both problems:

```
if( input == '1' )
  open action
else if( input == '2' )
  close action
else if( input == '3' )
  delete action
else if( input == '4' )
  rename action
else if( input == '5' )
  print action
else
  error action
```

After an evaluation results in true, no further if statements are considered. Also, when no match to input is found, then and only then is the *error action* performed.

Notice in the if else example, placement of the if else statement follows rules similar to use of the else statement. The first use of an if else must be preceded by an if statement and must immediately follow the action associated with the if. Following the first use of an if else, further if else statements may follow in the chain immediately after the action of a preceding if else.

The if statement always starts a new chain and must always be first. The else statement always ends the chain; therefore, it must be the last statement in the series (chain) of if, if else statements.

The if else statement is well suited for evaluating conditions with potentially many values. However, continuing with the discussion of decision constructs, the case statement provides an alternate solution to this problem.

The case Statement

Another way to solve the problem of evaluating the input shown in Figure 2.3 is to employ a case statement. The case statement provides a graceful and flexible alternative to the if else for this type of problem.

A case statement operates by switching on the value of a numeric or character variable for each of the conditions you want to account for:

```
switch( input ) {
  case '1':
    open action
   break
  case '2':
    close action
   break
  case '3':
    delete action
   break
  case '4':
    rename action
   break
  case '5':
    print action
   break
  default:
    error action
}
```

For each of the conditions in the switch, the case statement equates to an entry point for the condition and the break defines where to exit the structure. The flexibility this provides is that case statements can be grouped to share actions.

Note

Notice that the examples are becoming more and more rich with the C programming language syntax. The switch statement requires that start- and end-of-body markers show where the valid entry points begin and end as in the following example:

```
switch( var ) {
  case 1:
    break;
  case 2:
    break;
  default:
}
```

The open ({) and close (}) curly braces are used in C to mark the beginning and end of a body of code. We'll see more of this in Chapter 3.

Consider again the if else solution to the menu problem. If you decided to accept either a 1, O, or o as valid input to invoke the *open action* you would have to expand the test of if (*input* == '*1*') to a significantly more complex condition.

The flexibility of the `case` statement requires only that we add a condition (entry point) for the new input:

```
case 'o':
case 'O':
case 1:
  open action
  break
```

The `case` statement provides a decision construct that remains clear, succinct, and easily maintained.

Before leaving the `case` statement, notice the `default` condition in the previous examples. As expected, this is the condition executed when no other match is found.

Loops

Whether you are trying to find a match to an item in a list, count the number of customers in a database, or endlessly display a menu soliciting user input, doing things multiple times is often necessary in a computer program. A construct to make repetition possible is the *loop*.

Four ways to perform looping are available, and which one to choose does, of course, depend on the problem being solved.

The `for` Loop

Faced with the problem of having to look through a fixed length *array* for an empty slot, you should instantly consider the `for` loop in the solution.

EXCURSION

An Array Does Not Live in the Ocean

An *array* is a storage mechanism best thought of as a filing cabinet. A filing cabinet has a fixed number of drawers after it is created. Each *drawer* of an array is a storage location called an *element,* and the number of elements make up the array's *length*.

Like the number of drawers in a cabinet, the number of elements in an array cannot be changed after creation. Indexing into an array always begins with zero: `array[0]` is the first element of any array. Therefore, if an array has four elements, `array[0]` is the first and `array[3]` is the fourth and last element.

how tōō prō nouns′ it

Indexing into `array[0]` is spoken as if written *array sub zero*. Following this pattern, `array[1]` is *array sub one*, and so on.

Although there are several ways to construct a `for` loop, it generally expects that things will be done a finite number of times. The layout of the `for` loop looks like the following:

```
for ( initialize - test  - increment ) {
  action
}
```

EXCURSION

How to Define Bodies Using C Syntax

As seen with the `switch` statement, C syntax uses the curly braces (`{}`) as a means of grouping lines of code into a body. Syntactically, they are not required with the `for` loop as they were with the `switch` statement. However, without the start and end body markers, the `for` action could be only a single line of code.

Consider again the `if` statement where an ***`action`*** was associated with the `true` and `false` conditions. That was an easy way to conceptualize that the successful evaluation of the test results in something being done. The same could be said for looping, as the following shows:

```
for( initialize -- test -- increment )
  take action
```

However, the ***`action`*** taken may not be a single line of code but rather a series of lines.

If the ***`action`*** spans multiple lines, you must identify that they are *all* conditional, either conditional on a test associated with an `if` or conditional on the number of times you want to perform a loop.

Placing the conditional lines within a code body allows for the necessary grouping to identify them as a single conditional action.

To review, the C programming language uses the `{` as the start of a body of code (begin body marker) and the opposing `}` as the end of the body of code.

Consider again the `for` loop with the correct C syntax:

```
for( initialize; test; increment )  {
  some action
}
```

Notice the correct syntax is to have the initialize, test, and increment fields of the `for` loop separated by a semicolon (;).

Having `some` variable, you set its initial value in the `initialize` field of the `for` loop. In the next field, you test it to see whether it has reached some limit or satisfied a condition that will stop the looping. Finally, you increment the variable so that it approaches the value needed to satisfy the test:

```
for( i = 0; i < 10; i++ ) {
  some action
}
```

Note

In the previous example, if you omit the increment field of the `for` loop, you never move the value of `i` closer to the condition of `i` equal to `10`, which ends the loop. If this happens, `i` always equals zero, which is *forever* less than `10`. This is called an *endless loop* because the loop never ends.

Optionally, you can place the increment field in the body of the `for` loop if you leave a placeholder:

```
for( i = 0; i < 10; ) {
      i++;
some action
    }
```

Assuming the length of an array to be searched is 10 (meaning it has 10 elements), the `for` loop in the following example is ideal for the task:

```
for( i = 0; i < 10; i++ ) {
  if( item == array[i] ) {
    item found action
  }
}
```

Of course, the test field of the `for` loop could be adjusted according to the length of any array.

The `while` Loop

A `while` loop is generally employed when you don't know the number of times the loop will need to execute. An instance when a `while` loop is most likely to be employed is finding the end of a list of unknown length, as in the following:

```
while( condition )
  some action
```

or more specifically

```
while( !found ) {
  ...
}
```

how tōō prō nouns′ it

The statement reads *while not found*.

Note

Referring to Table 2.1, the *not symbol* (`!`) literally inverts the value of the variable. In order for the loop to execute, the value of `found` must initially be set to `false` as not false is `true`. As long as the condition of the `while` is `true`, the loop continues to execute.

The condition to end the loop must be created within the body of the loop to avoid the endless looping discussed earlier. For example,

```
while( !found ) {
  found = (item == list_entry );
  ...
}
```

In the previous example, the value of the variable `found` is assigned the result of the test ensuring that when `item` is found, the loop ends.

This would be equivalent to

```
while( !found ) {
  if(item == list_entry ) {
    found = True;
  }
  ...
}
```

Note the `...`, signifying that some critical elements of this fragment are missing to make this example fully functional. Specifically, the code necessary for traversing the list of unknown length is not shown. However, as this type of list management is used heavily to parse the objects within the Graphics Editor, I will visit it again in more detail in Chapter 6, "Components of an X Window Application."

The `do while` Loop

Very similar to the `while` loop is the `do while` loop. It differs in that the test for determining whether you continue executing the loop comes at the end of the loop instead of the beginning. This ensures that the loop is performed at least one time.

Borrowing from the menu example from Figure 2.3, consider that when displaying a menu for user input the menu must be displayed at least once. Further, it continually displays until the user enters some input to signify his intention to exit the program. This is exactly the problem for which the `do while` loop should be used.

```
do {
  print menu action
  get input action
} while( input != EXIT );
```

While the input from the user is not equal to the exit condition, continue displaying the menu.

Certainly the body of the `do while` wants to employ the `case` statement for performing the actions consistent with the user's selection.

Having thus far only generalized about the actions associated with decisions and loops, constructs are required to support branching program execution for valid actions. These constructs, which make branching possible, are called *functions*.

Functions

For the same reasons that you don't store all your files in a single directory, you don't put all your source code into a single routine. This amounts to poor organization, which makes program maintenance, advancement, and support extremely difficult. Further, without grouping code into functions, sharing functionality within a project requires typing it in again and again.

Functions, also known as *modules*, are the constructs that enable you to organize and group your source code based on the tasks it performs. The more precise the task, the more concise the code, the easier it is to maintain, and the greater chance you'll be able to reuse it elsewhere in your project.

Within a project, performing a task more than once should indicate the need to write a function. For instance, the requirement to open files is common to many programs. Instead of placing the code necessary to accomplish this in multiple places within your project, it is appropriate to add a function that accepts a filename, determines whether the file exists, opens the file, performs the necessary test for success, and, finally, returns the handle acquired by the open.

Many aspects are critical to understanding and using functions: declarations, return types, parameters, and definitions. After we've discussed each of these, you'll be prepared to look more closely at the C programming language.

Declarations

To use a function, whether it's one that you've written or one that is provided by the language or environment in which you are working, it must be prefaced with a declaration.

Declaring a function enables the compiler to validate it by looking for obvious violations of syntax, and perhaps warning about semantic errors.

Syntax violations checked by the compiler include ensuring that the proper numbers of arguments (parameters) are passed at invocation. Examples of parameter usage are provided in the section "Parameters."

If a function expects two arguments and you've only specified one, serious and potentially fatal results can occur. This is not fatal to the programmer, luckily, but it could very well cause the program to crash.

Note

The incorrect number of parameters specified when calling a function is a syntax error that causes the compiler to stop and insist that you fix it. Detection of this syntax error, however, is only possible if the compiler knows the number of parameters to expect for a function. This information is provided by the function declaration.

Semantic errors considered by the compiler include comparing the data type of items passed to those required. The type of the data implies an associated size, as discussed in the section "Data Types," later in this chapter. A mismatch in the type of data passed and expected can cause data to be lost or corrupted because size differences force the data to be expanded or truncated to satisfy the expected size.

Note

Type checking compares what the function expects to what is passed. For instance, if the function requires a number (integer), the compiler ensures that you passed a number.

More detail on data types is provided later in this chapter. However, it is important to know that a function's declaration ensures that the function is used properly. Without the function declaration, the compiler must make assumptions about the function, and the compiler always assumes anything unknown is an integer because this is the default type in the C programming language.

Two ways to declare a function for use in your source code are by *prototyping* the function and by *defining* the function.

When a function is defined, it satisfies as a declaration because the compiler learns everything it needs to by the function's definition. If a function definition appears before its first invocation, there need not be a separate declaration.

geek speak

A function's *definition* is, in fact, the function, and *a prototype* is the declaration of a function before its definition in order to provide the compiler with the following necessary information:

```
<return data type> Function Name ( parameter data types );
```

Although I've not discussed the various data types at length, you should be comfortable with the idea that variables must have a type associated with them. Some types available in C are listed in Table 2.2.

Table 2.2 Data Types

Type	*Description*	*Example*
`char`	character	a, b, c, and so forth
`int`	integer	1, 2, 3, and so forth
`float`	number with a decimal point	1.0, 2.02, 3.5, and so forth
`void`	non-type	an undefined or non-type

There is much more to know about data types, but we'll postpone the discussion until we visit them more fully in the section, "Data Types."

Return Type

The first thing to decide when writing a function, or to determine when employing an existing function, is the function's return type.

The function can return an integer specifying some success or failure. Perhaps there is data formed and returned by the function. Determining the return type will enable you to satisfy the first field of your function definition.

The return type of a function will be one of the data types from the first column of Table 2.2.

Note

You usually don't need to prototype an existing function because this is the responsibility of the author of the function. This is true in the C language because the functions provided for use have a corresponding prototype in a header file provided with the library. A challenge of learning C is becoming aware of the functions provided by the language and the correct header file to include when employing them.

Function Name

Following the return type of the function, you must include the function's name in the function's prototype. No matter what you've learned about computer programming, choosing names for variables and functions is the hardest part.

When choosing a function name, try to be as clear as possible without overdoing it. A function name too terse is as bad as a name that isn't sufficiently succinct.

EXCURSION

Naming Functions and Variables Is More Difficult Than Naming a Pet

Valid function and variable names must start with either a character or an underscore (_) followed by more characters and/or numbers. A function name cannot contain spaces or punctuation and should be descriptive of the task it performs.

For instance, `addTwoNum` is an appropriate name for a function that adds two numbers. Because function and variable names are allowed to contain numbers, `add2Nums` is also acceptable.

Parameters

Continuing to form the function prototype, following the function's name, the prototype is completed by providing a comma-separated list of the data types expected as arguments (parameters).

A *parameter list* is the data provided to a function.

Because the return type associated with a function provides the caller with a means of getting data out of a function, the parameter list is a way to pass data into a function.

Consider the example where you have a function that adds two numbers together.

The prototype for this example would be the following:

```
int addTwoNum( int, int );
```

Although I have yet to write the function (that is to define it), you know already what it looks like.

The function adds two numbers together so that we must pass the two numbers to be added into the function. As the purpose of the function is to sum two numbers, the return type should be the result of the addition. The only thing not immediately evident is what to call it (the hardest part). Choose a name as descriptive as possible without needing punctuation.

Notice that in specifying the data types required by the function (the return type and parameter list) you were not concerned with the variable names associated with the types. This is true for a function's declaration: It cares nothing for variable names. However, including them is not a violation of syntax, as in

```
    int addTwoNum( int num1, int num2 );
```

A function's prototype provides a wealth of information to the compiler. Once it is declared, you are free to use the function no matter where it resides in your project structure.

Even if the compiler does not know the contents of the function (its definition), you are still able to use it by merit of its prototype.

Definition

To define a function is to write it. The syntax for writing a function is similar to declaring it with two notable exceptions. Although optional in our prototype, we must include variable names in the parameter list, and we must specify the start- and end-of-body markers.

```
int addTwoNum( int num1, int num2 )
{
    function stuff goes here
}
```

The parameter list is the method of passing data into the function. In this example, you are passing two integers *num1* and *num2* into the function.

Presumably (based on the function name), the two numbers will be added together and the sum returned to you by the `return` statement. The data that the function returns dictates its return type.

The `return` Statement

The `return` statement is not present in the function body of the example, but it is required. Without it, the compiler reports the following error message:

```
Control reaches end of non-void function
```

From this you can infer that if your function does not return anything, its return type must be `void`.

The `return` statement for the function *addTwoNum* has the value of the sum of the numbers passed:

```
return sum;
```

Similarly, if the function does not receive anything, meaning that it accepts no data in the parameter list, the parameter list is `void`.

```
void SomeFunction( void )
```

The necessity of a `return` statement within the body of the function is important to remember. It must return data consistent with the return type of the function.

If the function has a `void` return type, no data is returned and the `return` statement is *optional*. If an optional `return` statement is present in a function, it cannot return data, as in the following:

```
void SomeFunction( void )
{
return;
}
```

Note

All C programs must have a function called `main` because this serves as the entry point into the program. When the program is executed, the function `main` is called. Examples of defining `main` with the required parameter list are covered in Chapter 3.

Having alluded to different data types and their associated size to satisfy elements of our discussion on functions, we are now ready to turn the focus to data.

Data

Perhaps the most important aspect of computer problem-solving is the understanding of data and its representation within the computer. Knowing differences between a byte, an integer, and a float is imperative for a computer programmer.

The understanding of data is even more crucial with the C programming language because C allows the representation of all data by reference.

geek speak

Accessing data *by reference* means referring to its address (location) in memory. A data element used by reference is also called a *pointer* because you are pointing to the data instead of pointing to the value of the data.

EXCURSION

Declaring Variables in the C Language

In the C programming language a variable is declared as follows:

```
int sum;
```

which follows the form

```
datatype variable name;
```

Any reference to the variable `sum` after this declaration is a reference to its *value*.

```
{
   int sum = 0;
   sum = sum + 1;
}
```

In the previous example, you are adding `0` (the current value of `sum`) and the constant `1` together, storing the result in `sum`, which is an integer (`int`).

The C programming language permits referencing data (variables) by their addresses. This is used extensively throughout the Graphics Editor project. Employing variables by reference is a very useful and powerful feature of the C language.

A clear benefit of variable pointers (variables by reference) is realized when passing data to a function's parameter list. When data is passed as a parameter to a function, it must be copied to a temporary storage location called the *stack* in order for the function to see it. As will be evident when we define the data structures used in the Graphics Editor project, the data being copied to this temporary location can be very large. The larger the data being copied, the larger the stack must be, and the longer it takes for the computer to copy the data.

However, if the data passed to a function is a pointer to where the data already resides, only the pointer is copied to the stack. Relatively speaking, pointers are small and quickly copied. Further, passing the address of where the function can find the data enables the function to modify the data with the results visible throughout the program.

Declaration of the variable `sum` as a pointer would appear as the following:

```
int * sum;
```

Notice the asterisk (`*`), which wasn't in the declaration of `sum` in the previous example. In this declaration, `sum` is declared as a pointer to an `int` and not itself an `int`.

A *pointer* refers to an address in memory.

Demonstrating the danger of pointer manipulation, consider that the variable `sum` is a pointer (reference to an address of where an `int` can be stored), and therefore, an assignment such as

```
sum = 0;
```

sets the *address* where an `int` would be stored to `0x0000`, which is invalid. The next *correct* assignment to `sum` would result in data being placed at address `0x0000` in memory and a crash (segmentation violation) would soon follow because the assignment was beyond the bounds (segment) allowed.

The correct method of accessing and assigning values of pointer variables is discussed later in this chapter; however, it is important to appreciate the flexibility of the C programming language in its representation of data.

Pointer manipulation is one of the most powerful (as well as one of the most dangerous) features of the C language. The authors of Java thought so ill of it that the Java programming language does not allow pointer references.

The goal of this section is to gain an understanding of the data types available in the C programming language. After discussing data types, focus will shift to understanding data by reference, as our Graphics Editor project uses this feature of C extensively.

Data Types

The first consideration when deciding what data type to assign when declaring a variable is the information to be stored by the variable. A second consideration is the precision (or space) required for the anticipated maximum and minimum values.

Deciding the type of information stored by a variable is an easy first step because whether the variable will be used for whole numbers, characters, or floating point digits should be clear from the context the variable is used.

Refer back to Table 2.2 for the data types available for representing information held in a variable.

When selecting a data type from Table 2.2 based on the criteria of the type of information being represented (stored), you must appreciate that the data type has an associated size.

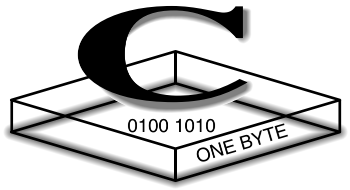
The size of a variable determines the limit of information that can be stored in it. Trying to fit more data into the storage space allotted by the implicit size of the data type will either result in corrupted data, or worse, a segmentation violation. (See Chapter 3, Figure 3.2 for an explanation of a segmentation violation.)

The `char` Data Type

The character data type (`char`) is represented in computer memory as one byte of data. As you might recall, a byte consists of eight bits, as shown in Figure 2.4.

Figure 2.4

Size of `char`.



In an earlier discussion, I represented the capital letter *A* as a decimal 65 or binary 0100 0001. Because it is the computer representation that matters and not the form that we are most comfortable viewing it in, let us consider the eight bits (byte) that the machine needs for representing a character. The binary representation of the letter A (0100 0001) has eight digits, each corresponding to one of the eight bits allowed for a character in computer memory.

Remember that these eight bits are bound by the binary numbering system in which there are only two digits, 0 and 1.

Note

Although there are eight bits within a byte, the most significant bit (furthest left in our representation) is used as a *sign bit* (+/-) and does not count in the value. Instead, the sign bit determines whether the data value represented is negative (bit set to 1) or positive (bit set to 0).

To determine the maximum value able to be represented by a variable of type `char`, set all bits available for representing data to 1 and add up the value.

With all available data bits in a `char` set to their maximum value, you get `s111 1111` where the `s` reminds you that there is a sign bit. Calculating the value requires a review of the binary numbering system.

As in the decimal numbering system, each column is weighted. When you balance your checking account you are comfortable with the fact that the weight of each column is the number of digits available within the numbering system raised by the column number.

In Figure 2.5, column weights for the decimal system are demonstrated (the numbering system used to balance your checkbook and pay your speeding tickets).

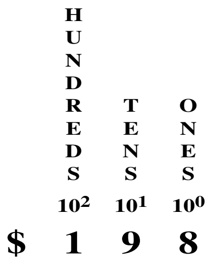


Figure 2.5

Decimal column weights.

Reading from right to left, number the columns starting with zero. For each column, raise the number of digits available for the numbering system by the column number.

Notice above the number 8 in Figure 2.5 that 10 is raised to a power of 0. There are 10 digits available to the decimal numbering system and 0 is the column number, making the weight of the first column 1 (any number to the power of zero is 1).

After you've determined the weight of each column, multiply the number in that column by the weight of the column and add it to its neighbor:

$$(\mathbf{1}\times100) + (\mathbf{9}\times10) + (\mathbf{8}\times1) = 198.$$

We are so used to the decimal numbering system that we might have forgotten how it works.

Apply the example of the decimal numbering system to the binary column weights shown in Figure 2.6.

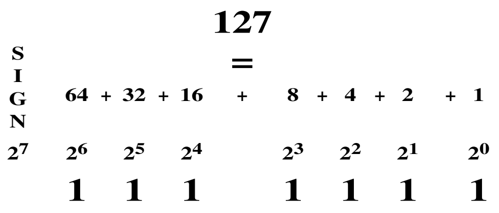


Figure 2.6

Binary column weights.

The maximum value a variable of type `char` can represent is 127.

Referring again to the ASCII table (see Note in previous section "Characters," page 54), you see that a value of 127 is sufficient for representing any character found in the ASCII table.

To determine the minimum value that can be assigned to a variable of type `char`, set all bits available for data to 1 and set the sign bit as well.

Setting the *sign bit* simply makes the entire value negative. Therefore, a variable of type `char` can represent any value from –127 to 127. However, what happens if you assign a value greater than 127 to a variable of type `char`?

If you've done the math, you've discovered 2^7 (the column weight of the sign bit) is 128. Therefore, representing any value greater than 127 using the data type `char` requires setting the sign bit. Although you think you are assigning, for instance, 129 to a variable of type `char`, the value it holds after the assignment is really –127 because you've exceeded its maximum value and data has *overflowed* to the sign bit column.

Note

The overflow condition is evidence of the necessity for type checking as discussed previously. If a character (`char`) variable is expected as a function parameter, but a data type with a greater maximum value is passed, the likelihood of exceeding the maximum value of the `char` is high.

How do you store a value greater than the 127 allowed by a `char` data type?

One way to exceed the maximum value of the `char` data type is to modify the `char` with the C keyword `unsigned`.

The *`unsigned` modifier* preceding a data type instructs the compiler to use the most significant bit for data instead of the sign bit. Clearly, a variable declared as `unsigned` is not capable of representing negative numbers. However, depending on the use of the variable this might not be an issue.

An `unsigned char` has a maximum value of 255 (128+64+32+16+8+4+2+1) and a minimum value of 0.

A second way to store values greater than the maximum of a `char` is to declare the variable as data type `int`.

The `int` Data Type

The data type `int` (integer) employs four bytes in computer memory for storing data, as indicated by Figure 2.7.

Figure 2.7

Size of `int`*.*

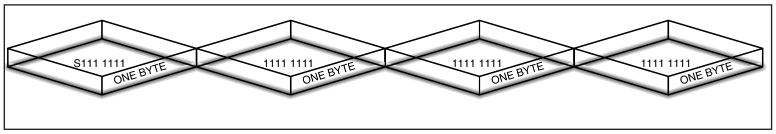


Figure 2.7 gives instant appreciation that the maximum value able to be represented by an `int` is significantly larger than that of a `char`.

Note

The size difference between a `char` and an `int` is not linear because the column weights in the binary numbering system increase exponentially.

How many data bits are available within the four bytes of the `int`?

4 (bytes) × 8 (bits per byte) – 1 (sign bit) = 31

There are 31 data bits in the data type `int` (integer). Table 2.3 shows the column weights of each of the 31 bits.

Table 2.3 Maximum Value of an `int`

Column Weight	*Value*
2^{30}	1,073,741,824
2^{29}	536,870,912
2^{28}	268,435,456
2^{27}	134,217,728
2^{26}	67,108,864
2^{25}	33,554,432
2^{24}	16,777,216
2^{23}	8,388,608
2^{22}	4,194,304
2^{21}	2,097,152

2^{20}	1,048,576
2^{19}	524,288
2^{18}	262,144
2^{17}	131,072
2^{16}	65,535
2^{15}	32,768
2^{14}	16,384
2^{13}	8,192
2^{12}	4,096
2^{11}	2,048
2^{10}	1,024
2^{9}	512
2^{8}	256
2^{7}	128
2^{6}	64
2^{5}	32
2^{4}	16
2^{3}	8
2^{2}	4
2^{1}	2
2^{0}	1
Total	2,147,483,647

geek speak

A variable of type `int` is capable of representing any number between –2,147,483,647 and 2,147,483,647.

As with the `char` data type, the modifier `unsigned` can be applied to the `int`. An `unsigned int` has a range of values from 0 through 4,294,967,295.

Because the data type `int` is able to represent much larger values than the `char`, why select the `char`?

Just as the size of the data type enables it to represent larger data values, it also requires more space in memory. Based on the data requirements, a programmer must decide the appropriate type to assign a variable. It is poor design to always use the largest data size available.

A programmer must give careful thought to how the data will be employed, correctly anticipate its maximum and minimum values, and choose a data type to represent it best.

Having covered the data types `int` and `char` and the modifier `unsigned`, let's continue by looking at other data types for representing numbers in the C programming language.

The `short` Data Type

Like the `int`, the `short` data type is used to represent numeric values. However, the `short` data type only employs two bytes for its data representation, making it smaller than the `int` and therefore capable of representing a smaller range of values.

geek speak

The 2 bytes (16 bits) available in a `short` allow it to represent values in the range –32,767 to 32,767. Applying the `unsigned` modifier to the `short` (`unsigned short`) changes the range to between 0 and 65,535.

The `long` Data Type

Another data type for numeric representation is the `long`. With 8 bytes (64 bits) of storage capacity, the `long` is the largest numeric data type with values between –9,223,372,036,854,775,807 and 9,223,372,036,854,775,807. As an `unsigned`, the `long` can represent between 0 and 18,446,744,073,709,551,615.

EXCURSION

A Caveat to the `long` *Data Type…*

The 64 bits required by the `long` must be supported by the platform. As reviewed in Chapter 1, "UNIX for Developers," PC platforms (Intel 80x86 architecture) are 32-bit machines. When only 32 bits are available to the `long`, it mirrors the `int` data type.

Other architectures exist that support the full 64 bits (for example, DEC Alpha).

A third data type provided by the C language allows for representation of real numbers.

The `float` Data Type

The `float` data type is used to represent numbers containing a decimal point or fractional value. Less important than size when choosing the `float` is the precision.

A larger data type, and therefore one significantly more precise, is the `double`. The `double` maintains up to 8 bytes of data for representing floating point numbers.

Consistent with other data types, choosing between the `float` and `double` depends on data requirements of the application and what is being represented.

In an application that, for instance, maintains data representing currency, a `float` can be adequate for representing fractions of dollars. However, if the task is to calculate world or geospatial coordinates, a `double` can be necessary for pinpoint accuracy.

Having covered the basic data types of the C programming language, we are ready to create our own.

Defining Structures

Structures are groupings of data types used to create a new entity.

The syntax for defining a structure requires use of the keyword `struct` followed by the name to be assigned to the new entity. Begin (`{`) and end (`}`) body markers enclose the fields of the structure.

```
struct mystruct {
  int  count;
  char flag;
};
```

The example structure has two fields: an integer named ***`count`*** and a character ***`flag`***. Separating the name of the structure and the name of a field with a period is one way to access fields internal to the structure.

```
mystruct.count = 1;
```

Defining `mystruct` this way makes it available for use in your program as well as for declaring other occurrences of it.

Declaring another occurrence of `mystruct` is accomplished with the syntax

```
struct mystruct newStruct;
```

where `struct mystruct` is treated as a data type and ***`newStruct`*** as the variable name of the new occurrence.

Note

Remember that `struct mystruct` must be defined before a declaration that employs it.

In the following example, `newStruct` is declared to be a `struct mystruct` and its fields are referenced as with `mystruct`:

```
newStruct.flag = 'a';
```

Optionally, C provides the keyword `typedef` for defining new data types.

Defining Data Types

To define the structure `mystruct` as a new data type requires the `typedef` keyword as a modifier to `struct`:

```
typedef struct {
  int count;
  char field;
} mystruct;
```

The modifier `typedef` instructing the compiler to define a new data type is different for this definition of `mystruct`. Also, the name of the structure follows the structure definition.

Following the `typedef` of `mystruct`, variables are declared in the following form:

```
mystruct newStruct;
```

Notice the keyword `struct` is no longer necessary in the declaration of *`newStruct`*. Further, following the `typedef` of `mystruct`, `mystruct` is not available for use in any other capacity except as a data type.

The last thing to consider before focusing on the syntax and convention of the C programming language is the correct method of referring to pointer variables.

Data by Reference

As discussed in the introduction to data types, the C language has the capability of referring to variables by their location in memory.

A variable, which refers to an address where the data is stored, is called a *pointer*. A variable of any data type can be declared a pointer.

Syntax for declaring pointers uses the asterisk (`*`) to indicate that the variable is an address.

```
int *intPtr;
```

In the previous example, `intPtr` is a pointer to where an `int` can be stored.

Consider a declaration of integer in the following manner:

```
int i;
```

At the moment of this declaration the value of `i` is unknown and considered *garbage*; therefore, the variable must be initialized.

The same is true for the value of `intPtr`. The address where an `int` may be stored is, at the moment of `intPtr`'s declaration, unknown and `intPtr` must be given a known value.

EXCURSION

Automatic Variables Employ Volatile Memory

Variables declared within a function are called *automatic* variables. The memory an automatic variable uses for storing its value is provided from the stack.

The stack was said earlier to be temporary storage used by the program. In addition to being temporary, the memory associated with the stack is very volatile: It is changing constantly.

Every time a function is called, a new *frame* is placed on the stack. A frame is a structure containing fields the program needs to store information about the function call. This information includes the function's parameter list, its automatic variables, and the point to return to when the function completes.

When a function finishes, the frame is removed from the stack and replaced with a frame for the next function called.

The contents of the memory where a stack frame is placed to support a function call is uninitialized except for the values assigned by the system to the fields that it controls. These fields include the return point and parameter data (assuming that the correct number of arguments was passed). Although memory is reserved for automatic variables, the content of this memory is entirely unknown and referred to as garbage.

An assignment into `intPtr` before its initialization would be a segmentation violation and a fatal error.

Note

An assignment can be made to any variable, but only pointers can be assigned into, or have a value placed in the location (address) pointed to by the variable.

Before an assignment can be made into `intPtr`, a valid address must be assigned to it.

Consider the following syntax for correctly initializing the `intPtr`:

```
{
  int i;
  int *intPtr;

  i = 0;
  intPtr = &i;
}
```

In this example, a variable of type `int` (`i`) and a pointer to an `int` (`*intPtr`) are declared. The variable `i` is initialized to `0` and the variable `intPtr` is assigned the address of `i`. Notice the use of the ampersand (`&`) for obtaining the address of `i`.

Following this example, `intPtr` is assigned a valid address for storing values (the address of `i`). It is now possible to store values into `intPtr`. To do so requires the following syntax:

```
*intPtr = 1;
```

geek speak

`intPtr` points to a memory address for storing an integer. *De-referencing* the variable with an asterisk (`*`) indicates the *contents* of the storage location or, literally, the value stored there.

Finally, consider how pointer manipulation applies to structures.

Modeling an example similar to `intPtr` and borrowing the structure definition `mystruct` results in the following code fragment:

```
1:    {
2:      typedef struct {
3:        int count;
4:        char flag;
5:      } mystruct, *mystructPtr;
6:
7:      mystruct newStruct;
8:      mystructPtr newStructPtr;
9:
10:     newStruct.count = 0;
11:     newStruct.flag = 'a';
12:     newStructPtr = &newStruct;
13:   }
```

Varying from the previous example of `typedef`, this example declares two new data types:

```
5:      } mystruct, *mystructPtr;
```

Applying what has been learned, you should realize that the variable `newStruct` is an occurrence of the structure `mystruct`. However, `newStructPtr` is a pointer to where a structure can be stored.

Just as the contents of `newStruct` are unknown at its declaration, meaning the value of the fields `newStruct.count` and `newStruct.flag` are considered garbage, the address pointed to by `newStructPtr` is also unknown at its declaration:

```
7:      mystruct newStruct;
8:      mystructPtr newStructPtr;
```

Therefore, initializations must be performed:

```
10:     newStruct.count = 0;
11:     newStruct.flag = 'a';
12:     newStructPtr = &newStruct;
```

Because `newStruct` is an occurrence of the structure, it has memory associated to it. The address of the memory is obtained by using the ampersand (`&`) before the variable name. This address can be used in an assignment to `newStructPtr`:

```
12:     newStructPtr = &newStruct;
```

To reference the fields of a pointer to a structure, such as `newStructPtr`, requires syntax different from referencing fields of an occurrence of a structure. With `intPtr` it was necessary to de-reference the address the variable pointed to. The same must be done to `newStructPtr`; however, the symbol to de-reference the address is different:

```
    newStructPtr->count = '1';
    newStructPtr->flag = 'b';
```

The dash followed by an arrow (`->`) is the correct syntax for referencing the fields pointed to by `newStructPtr`.

Next Steps

The discussion of programming conventions and data types throughout this chapter employed many elements of the C language syntax; however, we have not seen many subtleties and nuances of the language. Further, it is important that the requirements of the language be specified. The next chapter, "A Word on C," provides a thorough look at the syntax and conventions of C.

Chapter 3

In this chapter

- *Hello World*
- *Conclusion*
- *Next Steps*

A Word on C

Having been exposed to the C language syntax in Chapter 2, "Programming Constructs," we now focus full attention on the structure and convention of the language.

Every year the C community runs a contest to see which participant can write the most obfuscated C code.

obfuscate: *-cated, -cating, -cates*. 1. a. To render obscure. b. To darken. 2. To confuse.

The point of the contest is to show the importance of programming style in an ironic way and to emphasize subtleties of the C programming language. Although other structured programming languages have intricacies of their own, C inherently lends itself to the contest by offering features that are often confusing.

This chapter will lead you through a review of the syntax seen in the examples of Chapter 1, "UNIX for Developers," and Chapter 2, "Programming Constructs," and expand your exposure to the C language.

If you are already comfortable with your understanding of the C programming language, feel free to proceed to the discussion in Chapter 4, "Windowing Concepts."

The C programming language has a significant evolutionary history in the world of computer science. It was derived from the B language written by Ken Thompson in the late 1960s. (The predecessor of B was BCPL written by Martin Richards.)

The importance of this history is to recognize that C did not appear as a new language but was adapted from an existing language.

C's predecessor, B, was a language without types wherein it was up to the programmer to ensure that variables were used in a valid context.

The previous chapter discussed data types and the challenge they bring to computer programming. On the heels of this, you have to wonder how confusing the B language could have been.

Dennis Ritchie wrote C in the early 1970s, keeping most of B's syntax. Two elements that are characteristic of C, and often cause the most confusion with the language, are the relationship between pointers and arrays and the similarities between declaration syntax and expression syntax.

Developing the necessary awareness of C pitfalls while learning correct programming structure and C syntax is a goal of this chapter.

Hello World

A constant when teaching C is that at the end of the exercise the student is able to print *Hello World*. Using this as our structured example and bringing together lessons from previous discussions, let's begin with a file to hold the C source code.

A file can be created using the `vi` command (refer to "The `vi` Editor" in Chapter 1, page 22 for a review of the command):

```
bash[1]: vi first.c
```

Because every C program must define a function called `main`, begin by inserting the following code into the file:

```
0: /* the start of the program */
1: int main( int argc, char *argv[] )
2: {
3:    printf("Hello World");
4:
5:    return( 0 );
6: }
```

Comment Tokens

Use of the comment token recognized by the C language starts our example:

```
0: /* the start of the program */
```

The slash asterisk (`/*`) combination begins the comment and an asterisk slash (`*/`) ends it. Anything contained within the start and end comment tokens is ignored by the compiler and does not get placed in the object file. The GNU C Compiler also accepts the token slash slash (`//`) for support of a comment contained on a single line:

```
// everything after this comment token is ignored
```

The Function `main`

Continuing the discussion of the code sample, look closely at the definition of the function `main`:

```
1: int main( int argc, char *argv[] )
```

Because the compiler requires the function `main`, it does not need a prototype (declaration) before its definition. The compiler expects the function and therefore dictates its parameter list and return type.

EXCURSION

A Syntax for Representing Arrays of Strings

The declaration of the variable `argv` uses a syntax that you've only seen in part.

```
char *argv[];
```

Use of the `char` data type as a pointer (`char *`) is for storing a string or group of characters.

```
char *str = "abcdefghij";
```

Further, the syntax of following a variable name with square brackets indicates that the variable is an array (`argv[]`) (see the section "Data Types" in Chapter 2, page 70 for a review of arrays).

Defining a variable as

```
char str[10];
```

requests the compiler create an array with enough space in memory to hold 10 characters.

```
str[0] = 'a';
str[1] = 'b';
.
.
.
str[9] = 'j';
```

Combining the two in a single declaration as with `argv` from the example, we see

```
char *strArray[3];
```

which indicates that `strArray` is an array of strings as in

```
strArray[0] = "abc";
strArray[1] = "def";
strArray[2] = "ghi";
```

Reviewing the definition of main, you see that many elements have already been discussed.

Identifying the data types in the function definition

```
1: int main( int argc, char *argv[] )
```

reveals the return type of the function main is the data type int. Further, main expects two parameters, an int named argc and an array of char * (character pointer) called argv.

EXCURSION

Compiler Differences Affecting the Declaration of the Function main

The C compiler used imposes the return type and parameter list of the function main. The arguments argc and argv are consistent with all C compilers. These parameters enable the program to access the command-line parameters the user passes.

For instance, in Chapter 1 we used man to illustrate passing flags and parameters to UNIX commands:

```
bash[1]: man ascii
```

With the execution of the man command, the operating system responds by calling the function main defined by the program's author. When main is entered, the first parameter, argc, specifies the *number* of arguments placed on the command line when the user invokes the program and argv holds the *value* of each of them.

Invoking man ascii results in the value of argc being set to 2 where argv[0] contains man and argv[1] ascii. A way of representing this is

```
char *argv[] = { "man", "ascii" };
```

Notice that in this example, argv contains two elements, which correspond to the value of argc.

The return type expected from main is not consistent between compilers. Unlike Linux, most SYSV versions of UNIX allow a return type of void when defining main.

A return type is beneficial because it gives you the ability to test for the program's success or failure. By convention, a program exits with a value of 0 to indicate that no errors occurred. If the program fails to complete, a value greater than 0 is generally returned.

Think of argc as the *argument count* and argv as the *argument values*.

Code Bodies

Now that we've evaluated the definition of the parameter list and return type for main, let us consider the function's body:

```
{
    printf("Hello World");

    return( 0 );
 }
```

With any body of code, you must inform the compiler where the body begins (`{`) and ends (`}`).

In the previous example, no automatic variables are defined in the body of `main`; however, when automatic variables are defined, they must be placed immediately after the open brace marking the beginning of the code body.

EXCURSION

Where to Define Code Bodies in the C Language

Code bodies can be placed anywhere within a function and they can be nested. Code bodies define functions and associate a body of code with a loop or decision; however, code bodies can also be unconditional within a function. For instance, it is valid to have code bodies within bodies:

```
int main( int argc, char *argv[] )
{ // start of function body
    int done = 0;
    printf( "Hello World" );
    { // start of unconditional body
        while( !done ) { // conditional body
            /* do something */
        } // end while
    } // end of unconditional body
    return( 0 );
} //end function body
```

Notice the automatic variable `done` in this example. The declaration of `done` occurs immediately after the start of a code body, specifically, the function's body; however, variables can be declared after any start-of-body marker.

EXCURSION

A Stylistic Note…

When code bodies are added to functions (as in the preceding code), it is a courtesy for those who follow behind you, and it also increases the readability of the code and adds a level of indentation to the code contained in a body. For instance, the code in the first body would be tabbed once, the code in the second body twice, and so on, with the associated begin and end body markers always in the same column. Consider the previous example without using any indentation and without the benefit of comments:

```
int main( int argc, char *argv[] )
{
int done = 0;
printf( "Hello World" );
{
while( !done ) {
/* do something */
}
}
return( 0 );
}
```

Only by carefully studying the sample can you determine the relationships between the various code bodies illustrating the utility of proper indentation.

When automatic variables are declared within a new body of code, they are governed by rules of visibility known as *scope*.

Variable Scope

Automatic variables declared in a body of code are visible from the moment of the declaration until program execution reaches the end of the body.

```
int main( int argc, char *argv[] )
{
    int cnt;
    /* sum is not visible here as it has not been declared */
    {
        int sum;
        /* cnt is still visible here as
           execution has not reached the body
           in which it was declared */
    }
    /* sum is no longer visible as the body
       in which it was declared has ended */
}
```

A variable is said to be *in scope* when it is visible and *out of scope* when it is not.

Variables defined outside a function are called *global variables*.

If a variable declaration occurs outside a function, its scope is extended to all functions and code bodies that are defined after the point of the declaration.

```
int sum; // a global variable visible
         // until the end of file

int main( int argc, char *argv[] )
{
    sum = add2Nums( 1, 2 );
7:  }
8:
9:  int add2Nums( int num1, int num2 )
10: {
11:     return( num1 + num2 );
12: }
```

This example takes into account nearly everything discussed so far. As required by the GNU C compiler, the function `main` is defined as returning `int` and expecting the parameters of `argc` and `argv`. When `main` is called, it assigns the value of the global variable `sum` the result returned by the function `add2Nums`.

The function `add2Nums` expects two arguments of type `int`, which `main` provides by passing `1` and `2`.

Notice that `add2Nums` does not have to return anything but could make the assignment directly to `sum`.

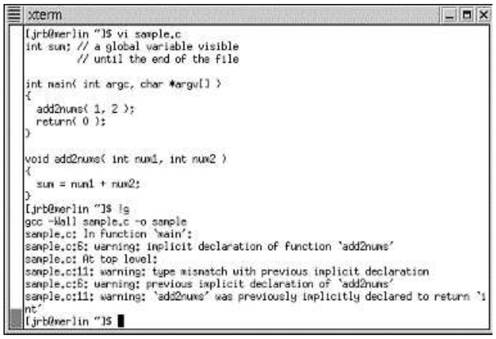
```
9:  void add2Nums( int num1, int num2 )
10: {
11:     sum = num1 + num2;
12: }
```

As the variable `sum` is defined before the function `main`, it is visible to `main` as well as to `add2Nums`. Notice that as `add2Nums` now has a `void` return type the `return` statement may be omitted. Further, `main` would simply invoke `add2Nums` as

```
6:  add2Nums( 1, 2 );
```

A significant detail has been overlooked in this example; there is no declaration of `add2Nums` prior to its invocation by `main`.

Figure 3.1 shows the error issued when an attempt is made to compile this sample.



```
[jrb@merlin ~]$ vi sample.c
int sum; // a global variable visible
         // until the end of the file

int main( int argc, char *argv[] )
{
  add2nums( 1, 2 );
  return( 0 );
}

void add2nums( int num1, int num2 )
{
  sum = num1 + num2;
}
[jrb@merlin ~]$ !g
gcc -Wall sample.c -o sample
sample.c: In function `main':
sample.c:6: warning: implicit declaration of function `add2nums'
sample.c: At top level:
sample.c:11: warning: type mismatch with previous implicit declaration
sample.c:6: warning: previous implicit declaration of `add2nums'
sample.c:11: warning: `add2nums' was previously implicitly declared to return `i
nt'
[jrb@merlin ~]$ █
```

Figure 3.1

An example of a compile error resulting from a lack of function prototype.

The GNU C compiler is relatively clear on what is wrong with the sample code.

The compiler makes up an *implicit declaration* when no actual declaration is found before a function is called. Implicit declarations are avoided by explicitly defining functions.

Notice the portion of the error output in Figure 3.1 that reads

```
sample.c:11: warning: 'add2Nums' was previously implicitly declared to return 'int'
```

This is consistent with the description in Chapter 2, section "Forward Declarations," page 64 of the compiler's behavior. Anything the compiler doesn't recognize is assumed to be an `int`, which is the default data type in C.

At line 6, according to the error output, the compiler found an implicit declaration. This line corresponds to the first use of the function. Later, at line 11, it found the actual definition of the function that wasn't consistent with what was implied (the compiler's assumption).

To resolve the error, simply add a new line at the beginning of the file declaring the function prior to its use:

```
0:  void add2Nums( int, int );
```

Knowing how to define, declare, and employ functions and variables correctly is critical in advancing our knowledge of the C programming language. Building on this knowledge, it is important to become familiar with the functions that C provides us.

Built-In Functions

You will not have to author all the functions that your application employs because the C programming language provides many functions for you. When you become aware of functions inherent to a language, you increase your proficiency in it.

The availability of functions provided by an environment such as X Window or a third-party package adds to this challenge.

Third-party packages are ones that you add to your system and are generally not part of the standard installation process. These packages are usually purchased separately and selected for the functionality they add to your development or runtime environment. The OSF/Motif Widget Set is an example of this; it compliments the X Window System because it provides elegant three-dimensional features to X Window-based application development.

Focusing on some of the functions provided by the C programming language, look again at the code sample from the Hello World example that started the chapter:

```
3:       printf("Hello World");
```

The `printf` Command

The `printf` command is a function the C language provides for printing formatted output to the terminal. The use of `printf` is extremely flexible because it supports every data type recognized by the C language.

EXCURSION

Terminals Aren't Just Where Planes Depart

A terminal may be your screen or a terminal emulator (window) such as the `xterm` depending on whether your system is running in graphics or text mode.

Generally, `printf` places its output on the command line following where the program was executed. This destination is known as *standard out* and represented in C as `stdout`. In addition to standard out, the C language also provides the destination standard error (`stderr`) for output.

Standard out and standard error start as the same destination; however, they can be altered through the use of redirection as discussed in the section "`grep`, Pipes, Redirection, and `more`" in Chapter 1, page 44.

Chapter 4 covers windowing concepts and terminal emulators in detail.

As with the function `add2Nums` defined earlier, there must be a forward declaration (prototype) for the function `printf`. Because the C language provides the function, it also provides the prototype.

The `printf` function's declaration (and many other functions performing input and output) is found in the file `stdio.h`, which is part of the standard C header files.

EXCURSION

Another Look at the Use of Header Files

Header files, as discussed in the Note found in the section "Definition" in Chapter 2, page 68, are source files often containing prototypes and other declarations shared by multiple files within a project.

You include a header file by using the `include` *compiler directive*. As the name implies, a compiler directive directs the compiler to perform a task or make a decision.

In C, all compiler directives are prefaced with the pound sign (`#`).

```
#include <stdio.h>
```

Used in the same manner as the `include` statement from `Makefile` syntax as discussed in Chapter 1, section "Include" on page 37, the compiler directive to include a header file performs the inclusion at the point of the directive. The effect is that the compiler sees the contents of the file as if replicated by the directive.

The syntax when employing the `include` compiler directive offers a hint to where the file can reside. Specifically, when the header is part of the standard C library the < and > are used to enclose the filename. However, when the header file is part of a third-party package or one that you have authored as part of the project, the filename is enclosed by double quotes.

```
#include "gxGraphics.h".
```

The C compiler first looks in a standard directory such as `/usr/include` for files enclosed with the < > symbols and then it considers the paths specified by the `-I` flag passed when the compiler was invoked.

To review the `gcc` command and use of the `-I` flag, refer to Chapter 1, section "`gcc -I`", page 28.

To properly declare the `printf` function prior to its invocation, add the following line to the `"Hello World"` code sample:

```
0a: #include <stdio.h>
```

The syntax of the `printf` function follows the form

```
printf( "format string" [, arg][, arg][, ...] );
```

To accomplish the formatting recognized by the `printf` function, the *format string* ranges from a constant such as `"Hello World"` to accepting a variety of *formatting tokens*.

Note

Notice in the syntax of the `printf` that the `args` are enclosed in square braces ([]), indicating that they are optional. An argument (`arg`) is only required if a format token is nested in the format string.

Table 3.1 shows some common formatting tokens recognized by `printf` that can be embedded in the format string.

Table 3.1 `Printf` Formatting Tokens

Token	*Type*	*Sample*	*Output*
`%s`	`char *`	`printf("String: %s", "Hello" );`	`String: Hello`
`%c`	`char, unsigned char`	`printf( " Character: %c", 'A' );`	`Character: A`

continues

Table 3.1 Continued

`%d`	`int`, `long`, `short`, `unsigned int`, `unsigned long`, `unsigned short`	`printf( "Number: %d", 198 );`	`Number: 198`
`%f`	`float`, `double`	`printf( "Real Number: %f", 3.14 );`	`Real Number: 3.14`

All formatting tokens are prefaced with the percent sign (%), indicating to `printf` that what follows is for argument substitution. Further, for each token in the format string there must be a corresponding argument to satisfy the substitution.

Multiple arguments are comma separated and substituted in the order they are placed.

```
printf( "String %s and Char: %c", "Hello", 'A' );
```

Note

As demonstrated in the previous example, C uses double quotes (" ") to represent multiple characters (strings) and single quotes to represent a single character (' ').

Note

`printf's` capability to perform type checking is limited. A token nested in the format string expects a complimenting argument of a specific type. If an argument of a differing type is placed in the argument list, the results cannot be predicted. The `printf` statement will attempt to *cast* the argument, but because this is done at run-time (while the program executes) there is no recovery if the types are not compatible.

EXCURSION

Promoting Variable Data Types to Satisfy Type Checking

Casting from one data type to another is a way to promote variables to satisfy type requirements and avoid compiler warnings. For instance, a character (`char`) is easily promoted to an integer (`int`), which has a larger storage capacity.

The code fragment

```
    char chr = 'A';
    int num = (int)chr;
```

assigns the letter A to the variable chr. Because the letter A has a decimal equivalence of 65, casting chr to an int would assign the value 65 to the variable num.

Caution must be used when casting from a larger data size to a smaller, however, because data could be lost. Consider the fragment

```
int bigNum = 999;
char chr = (char)bigNum;
```

Because the maximum value of a character is 256, the assignment of bigNum to chr, although made legal by the cast, results in the new value of chr being –25. Clearly, this is not the expected result.

See the section "Data Types" in Chapter 2, page 70 for a review of valid value ranges and the implicit size of recognized data types.

Looking again at the format string accepted by the printf function, it should be clear why printf is said to perform formatted output. Additional formatting tokens and token modifiers are available to printf for outputting data types in varying forms as well as controlling field widths, alignment, and more.

Review the printf man page for a full description of its capabilities.

The printf is one of several functions that C provides for performing formatted output. Similar to printf are the functions fprintf and sprintf.

Both fprintf and sprintf enable the output to be directed some place other than standard out. For instance, fprintf may be used to send the formatted output to standard error.

```
fprintf( stderr, "%s %s", "hello", "world" );
```

Differing from the syntax of printf, fprintf requires as its first parameter the destination designator for the output. This destination can be one that C provides, such as stderr used in the example, or it can be one created by using the fopen (file open) function.

Before looking at the use of fopen, consider sprintf as it relates to the printf and fprintf functions. Use of sprintf enables a programmer to format output for placement in a *buffer*.

geek speak

A *buffer* is a character array used for intermediate storage during input or output operations. For instance, in

```
char message[25];
```

message is declared as an array of 25 characters and could be used to satisfy the first parameter required by sprintf.

```
sprintf( message, "Error occurred at line %d", lineno );
```

EXCURSION

Apply Great Caution when Determining the Correct Size of an Array

In the `sprintf` example, `message` was declared with a length of 25. Count the characters in the format string passed to the function:

```
"Error occurred at line %d"
```

Including spaces, there are 23 characters in the format string, excluding the value of the formatting token `%d`.

The C function `sprintf` *null terminates* the output that it formats: It inserts a null character (`\0`) at the end of the string. This termination is important for other C functions that can act on the string and it consumes one place in the buffer.

With the 23 characters in the format string and one character for null termination, a total of 24 characters are placed in the buffer `message` before the argument substitution for the token `%d`. If the value of `lineno` substituted in the `message` is only one digit (0–9), you have exactly filled the 25 character spaces available to `message`. However, if the value of `lineno` is greater than 9 (two or more digits), you will exceed the length of `message` because the null termination will be placed outside the valid memory associated with the buffer.

Figure 3.2 illustrates the effect on memory when the boundary of an array is exceeded, a condition known as a *segmentation violation*.

The value 69 placed in the buffer exceeds the allowed space for `message`. The owner of the memory that the `\0` (null) overwrites is unknown. The memory can be unused or it can be a critical part of the program. Depending on the importance of the unknown space, the program might crash instantly, or it might only corrupt the value of the neighboring space, leading to a crash much later in program execution.

Note

Improper use of arrays is one of the most common causes of program bugs.

geek speak

A *program bug* is anything that causes a program to behave unexpectedly, often resulting in a crash.

Now that we've reviewed the built-in functions of `printf`, `fprintf`, and `sprintf` for formatting and outputting data, we can return to the use of the `fopen` function for creating destination designators to be passed to `fprintf`.

The `fopen` Function

The `fopen` function opens a file named in the parameter list returning a handle to the file known as a *file pointer*.

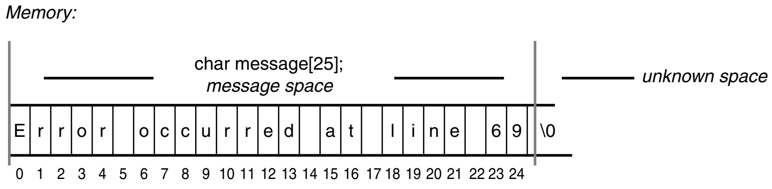
Incorporating everything discussed thus far, consider the following code sample illustrating the use of the `fopen` function:

```
1:  #include <stdio.h> // for printf and fopen function prototypes
1a:                    // and FILE structure definition
2:  FILE *openFile( char * filename )
3:  {
4:       FILE *fp;
5:
6:       fp = fopen( filename, "w+" );
7:
8:       if( fp == NULL ) {
9:           fprintf( stderr, "Unable to open file %s", filename);
10:
11:     }
12:     return( fp );
13: }
```

The code sample defines a function named `openFile` that accepts a single parameter `filename` and a character pointer, and returns a pointer to `FILE`.

Figure 3.2

An illustration of the error that results in a segmentation violation.



Note

The `FILE` data type is a structure that C provides for referencing files opened with the `fopen` function. The structure is considered *opaque*, meaning that the fields defined within the structure are not to be accessed. The structure exists only to serve as a handle for manipulating files.

The standard out (`stdout`) and standard error (`stderr`) references provided by C are pointers to the `FILE` structure (`FILE *`).

In the code sample demonstrating use of the `fopen` function, notice the test on the value returned by the `fopen`.

```
8:      if( fp == NULL ) {
```

If `fopen` is unable to open the file specified, it returns `NULL`.

C provides `NULL` as a means of representing *nothing*. Literally defined as 0 (zero) and cast to a void pointer (`(void *)0`), it is returned in cases of failure to create a valid reference to a return value by functions such as `fopen`.

The `fopen` command expects two parameters. The first, evident by the example, is the name of the file to open:

```
fp = fopen( filename, "w+" );
```

The second parameter is the *mode* in which `fopen` should open the file.

Table 3.2 shows the valid modes that can be passed to the `fopen` function and describes their effect.

Table 3.2 File Modes Understood by `fopen`

Mode	*Description*
`r`	Open the file for reading
`r+`	Open the file for reading or writing
`w, w+`	Truncate to zero length or create; file is opened for writing
`a, a+`	Append, open, or create file for update, writing at the end of the file

Upon successfully opening a file, `fopen` returns a file pointer (`FILE *`) reference which can be passed to functions requiring a destination designator such as `fprintf`:

```
fprintf( fp, "Line entered into file referenced by fp" );
```

Now that you're comfortable with the code sample illustrating the `fprintf` command, the use of `NULL`, function bodies, rules of scope, and variable and function declarations, we will consider another family of functions provided by C.

The C String Library

The lexical analysis of strings as discussed in Chapter 2, the section "Types of Conditions," page 52, was said to require a different method from integers and characters for forming test conditions. Because string manipulation is language specific, the C string library satisfies this requirement.

C provides many built-in functions for comparing, copying, and creating strings. The first to consider is the function for comparing two strings called `strcmp` (string compare). Passing the two strings for comparison satisfies the parameters required by the `strcmp` function:

```
#include <strings.h>
{
    char *str1 = "tag",
         *str2 = "day";

    if( strcmp( str1, str2 ) == 0 ) {
        // the strings match
    } else if( strcmp( str1, str2 ) < 0 ) {
        // str1 appears before str2 in a dictionary
    } else if( strcmp( str1, str2 ) > 0 ) {
        // str1 comes after str2 in a dictionary
    }
}
```

Notice that this code sample uses the `include` compiler directive to include the file `strings.h`:

```
#include <strings.h>
```

This is the header file that C provides to satisfy the built-in string functions' forward declarations.

EXCURSION

Combining Declarations and Assignments Using C Syntax

A variation on variable declaration appears in this code sample as the declaration of `str1` and `str2` are combined sharing the data type `char`.

```
    char *str1 = "tag",
         *str2 = "day";
```

Syntactically correct, multiple variables of the same type can be comma separated at their declaration.

Seen previously but not explicitly noted is the combination of a variable's declaration and its initialization as shown in the previous example.

A sound programming habit is to initialize a variable prior to the variable's use. It saves you from having to scour code later looking for the obscure bug that lack of initialization might cause if the variable is employed in an expression before being assigned an initial value.

As implied by using `strcmp` in the previous code sample, the comparison of the two strings returns 0 if the strings are equal. The `strcmp` function returns a value greater than (or less than) 0, depending on the lexical analysis of the two strings.

EXCURSION

The Lexical Analysis of Two Strings of Varying Length

If the two strings being compared by `strcmp` are not the same length, as in

```
char *s1 = "act", *s2 = "abra";

if( strcmp( s1, s2 ) == 0 ) {
```

then the function compares the strings up to the point of finding the `NULL` termination or end-of-string marker. (See the Excursion in the section about `sprintf` for a description of string termination performed by C).

It is possible to inform C how many characters of the two strings to compare by use of the `strncmp` function.

The `strncmp` function accepts one more parameter than its sister function `strcmp`. Specifically, the third parameter informs `strncmp` of the number of characters to consider in the comparison.

```
char *s1 = "act", *s2 = "abra";
if( strncmp( s1, s2, 3 ) == 0 ) {
```

In this example, you've explicitly said to stop comparing after three characters.

how tōō prō nouns′ it

The function `strncmp` is read like it is written *stir-n-compare*.

Table 3.3 shows several string functions that C provides; they are common in programs employing the language.

Table 3.3 C String Functions

Function	*Example*	*Description*
`strcmp`, `strncmp`	`if(strcmp(str1, str2) == 0) {` or `if(strncmp(str1, str2, 4) == 0){`	Compare two strings (optionally specifying number of characters to compare)
`strcpy`, `strncpy`	`char str1[10]; strcpy( str1, "Initialize" ); strncpy(str1, "Initialize" , 10);`	Copy one string into another (optionally specifying number copy); Caution: the destination string, `str1`, must have space available for the number of characters being copied to avoid a segmentation violation
`strcat`, `strncat`	`char str1[10]; strcat( str1, "Warn" ); strcat( str1, "ing " ); strncat( str1, "at line", 2 );`	Concatenate two strings; the second string specified is added to the end of `str1` (optionally specify number of characters to add to the end of `str1`); Caution: the destination string,`str1`, must have space available to hold the contents of the second string or a segmentation violation will occur

continues

Table 3.3 continued

Function	*Example*	*Description*
`strchr`	`char *token, *buf = "No error"; token = strchr ( buf, 'e' );`	Find the first occurrence of a character within a string; function returns either a pointer to the position of the character within the first parameter, or `NULL` if no match was found
`strstr`	`char *token, *buf = "No error"; token = strstr ( buf, "err" );`	Find the first occurrence of a sub-string within a string; function returns a pointer to the start of the sub-string within the first parameter or `NULL` if no match was found
`strdup`	`char *newStr = strdup( " error_log");`	Duplicate a string, returns a new copy of the string specified to the function
`strlen`	`int len = strlen ( newStr );`	Return the length (number of characters) comprising the string

Table 3.3 shows that dangers are associated with string manipulation because the space available when copying strings must be sufficient to hold the new contents. The idea of space in a computer program always translates to memory.

The caution extended in the introduction to pointer manipulation, variables by reference, (see the section "Data Types" in Chapter 2, page 70) must be applied to strings as well. In fact, it is the same warning described in the Excursion in the section "`sprintf`" earlier in this chapter. The common thread is a necessity for proper memory management.

Memory Management

Proper memory management is critical in application development. Careful attention has been paid in previous code samples to ensure that the compiler always *implicitly* provided the memory space necessary.

Memory management can be implicit or explicit. C provides built-in functions enabling a programmer to allocate and free memory explicitly. It is also possible that the necessary memory is implied by the manner in which a variable is declared.

Samples of associating memory implicitly with a variable include

```
char str1[10];
```

where the variable array `str1`, upon declaration, has sufficient space for storing 10 characters.

Consider also

```
char *buf = "No error";
```

where the variable `buf` is declared with space enough to hold 8 characters as implied by the initialization combined with the declaration.

Memory associated with variables can be taken from one of several areas available to an application. Variables declared after a start-of-body marker reside on the stack. (See Chapter 1 for a review of automatic variables and the stack.) The stack is volatile memory changing constantly during program execution.

Note

A programmer affects the stack through the manner in which variables are declared. In other words, the programmer can never explicitly allocate or free stack memory.

The memory associated with the variable `buf` in the previous code fragment resides on the stack until it is out of scope. After it is no longer visible, the stack frame holding the reference to `buf` is removed from the stack and a new frame uses the memory.

Contrast the declaration of the two character pointers in the following example:

```
char *buf = "No error",
     *token;
```

The variables `buf` and `token` both point to characters with the contents of `buf` being initialized at the moment of its declaration. Based on `buf`'s initialization, you know both its size (length) and contents. Because no memory is associated with `token`, it has neither a length nor valid contents. `Token` is a character pointer, but as of yet points to nothing (garbage).

The cumbersome task of *memory management* thus begins.

A valid statement is to assign the memory associated with `buf` to `token`.

```
token = buf;
```

Because the variables are compatible (both character pointers) the assignment is legal and logical; now, `token` refers to valid memory (the implicit memory given to `buf` from the stack). Employing `token` follows the same rule as using `buf`, namely, the size of the memory provided cannot be exceeded (eight characters).

Furthermore, when the variables go out of scope, their contents are no longer valid because a new stack frame will begin using the memory that was once reserved for them.

Consider this incorrect code sample in which the memory associated with an automatic variable is returned to the calling function:

```
// forward declaration
char *someFunc( void );

// entry point of the program
int main( int argc, char *argv[] )
{
    char *badString = someFunc();
    printf("The value of str is: " );
    sleep( 1 ); // new stack frame
    printf( "'%s'\n", badString );
    return( 0 );
}

char *someFunc( void )
{
    char buf[15];
    strcpy( buf, "Error ahead" );
    // return the memory provided from the
    // the stack for the variable buf
    return( buf );
}
```

As described in the previous example, when the function `main` calls `someFunc`, a new frame is placed on the stack to manage all aspects of the function call. Specifics such as the function arguments, the return address of the calling function, the return value, and the local (automatic) variables, as well as temporary storage needed during function evaluation, are part of the stack frame.

Therefore, the memory used to store the value of `buf` in the function `someFunc` resides in the stack frame managing the function call. When the function returns, the frame is removed and all associated memory is made available for subsequent function calls.

Note

A distinction must be made between returning *pointers* (addresses) from a function and returning *values.*

The caution being issued in the previous example is for returning addresses to data values whose location is on the stack and therefore in volatile memory.

The variable `buf` is a pointer to a value and not the value itself. Returning the value presents no danger because a copy of the data is made before returning it to the calling function.

Returning a value is not always practical, however, because the size of a data structure can adversely affect program performance.

Warning

Returning and employing memory from the stack creates memory errors that will likely result in a program crash when the stack frame is no longer active.

In the previous example, if `main` did not exit immediately but instead invoked another function, a new frame would be placed on the stack and the memory previously associated with `buf` (still pointed to by `badString`) would be reused. The new function owning the memory location would store a new value, possibly a value that is not character data. This would ensure that the value of `badString` is not as expected and a program crash could soon occur.

To avoid the problem of returning memory contained on the stack, you must explicitly allocate memory associated with pointer variables or reserve them in non-stack memory if their use extends beyond a single function.

The `static` Keyword

The keyword `static` serves several purposes in the C programming language. One of its uses is to force the memory associated with a variable to come from non-stack memory. This is beneficial because the memory will be persistent through the life of the program and not just the life of the function, as we saw with the previous example.

Use of the `static` keyword follows the form

```
static data type variable name;
```

Applying this to the previous example to correct the imminent program crash would require that the declaration of the variable `buf` change from

```
15:     char buf[15];
```

to

```
15:     static char buf[15];
```

As stated previously, the `static` keyword informs the compiler that the memory associated with the variable must not be taken from the stack. Instead, persistent memory is used and therefore the contents of `buf` are maintained during the life of the program. In addition to the memory address being safely returned to the calling function with the line

```
19:     return( buf );
```

the contents of the variable will be maintained in consecutive calls to the function. More examples of static automatic variables will be seen in the Graphics Editor project.

Note

Memory associated with `static` variables can never be returned or unreserved by a program. Therefore, memory reserved as `static` permanently increases the size of a program.

Note

You can also use the `static` keyword beyond making a variable's memory persistent to limit the variable's scope.

➔ As described earlier in this chapter, in the section "Variable Scope" on page 88, variables declared outside of a function body are global variables.

Global variables are visible (in scope) from the moment of their declaration until the end of the file. Global variables can be made visible to other files through use of the keyword `extern`.

The `extern` Keyword

The keyword `extern` can preface any variable declaration as a method of informing the compiler that the variable's *actual* declaration is *external* to the current file.

The declaration

```
extern int sum; // references a previous declaration of sum
```

tells the compiler that use of the variable `sum` is by merit of a previous declaration in another file. By using `extern`, the variable is global (visible) to multiple files.

However, if the original (actual) declaration employed the `static` keyword, visibility would be limited to the file in which it was declared. In other words, when you use the keyword `static` on a global variable, the variable cannot be declared externally through use of the `extern` keyword in another file.

Similarly, when functions are prefaced with the `static` keyword, their visibility, too, is limited to the file in which they are declared even if a prototype for the function exists elsewhere.

A function defined as

```
static int add2Nums( int num1, int num2 )
{
    return( num1 + num2);
}
```

can only be invoked by functions in the same file, which contains this definition because the `static` keyword limits its visibility.

Using the `static` keyword in the context of ensuring that a variable's memory is persistent during program execution is one method of managing memory that doesn't employ the stack. Another method of managing memory is to dynamically request it from an area known as the *heap*.

Dynamic Memory Allocation

The area of memory known as the heap that is available to applications is managed by using built-in functions provided by C for dynamic memory allocation.

geek speak

The *heap* is an area of memory, separate from the stack, which is available for program use. Nothing automatically happens to heap memory as it does to the stack with frames added and removed constantly; this makes the heap a preferred memory location for data and structures with a life span greater than a single function.

Only heap memory can be dynamically allocated and freed using the built-in functions for memory management.

To dynamically allocate memory, C provides several functions. The first to consider is the function `malloc`.

`malloc` (Memory Allocation)

The allocation function `malloc` reserves a block of uninitialized memory from the heap in the size specified.

```
char *buf;
buf = malloc( sizeof(char) * 10 ); // space for 10 characters
```

If the function `malloc` is not able to reserve the requested memory, it returns `NULL`. Testing the return of the function ensures that a valid memory block exists and diverts a program crash.

```
if( buf == NULL ) {
    printf( "Fatal Error: Failed to malloc memory" );
    exit( 1 );
}
```

The allocation of `buf` in the previous example is equivalent to

```
char buf[10];
```

except that the memory returned from the `malloc` is heap memory and will exist after the function containing the allocation returns. This complicates memory management slightly because any memory allocated from the heap must be explicitly returned to the heap when the program is finished with it.

To return memory to the heap, making it available for subsequent allocations, C provides the function `free`.

`free` (Returning Allocated Memory)

Every block of memory dynamically allocated should have a corresponding `free`.

When you fail to free memory that is dynamically allocated within a program, you have caused a *memory leak*. Memory leaks cause programs to suffer in performance as well as face a potential failure.

The memory associated with a program grows in direct proportion to the size of the program's heap.

A program that never returns memory to the heap continues to grow in size until no further growth is possible; either the system runs out of memory or a maximum program size is reached, at which time further calls to `malloc` fail. Also, the greater the amount of memory consumed by a program the slower its execution.

The syntax for performing a `free` follows the form

```
free( (char *)mem );
```

where *`mem`* is the memory gained by a dynamic allocation function such as `malloc`.

Note

The function `free` expects a parameter of the `char *` type to be passed to it. Because the function is used to return the allocation of any type of data to the heap, it is often necessary to cast the memory being returned to a pointer of the proper type (`char *`).

Table 3.4 shows several functions provided by C for performing dynamic memory allocation.

Table 3.4 Memory Allocation Functions

Function	*Example*	*Description*
`malloc`	`memPtr = malloc( size );`	Returns a pointer to an unitialized block of memory; where *`size`* is the size of the block, for example, `(sizeof(`*`data type`*`) * `*`count`*` )`
`calloc`	`arrayPtr = calloc( size, num_ele );`	Returns a pointer to an array initialized to 0, with `num_ele` (number of elements) of the specified size
`realloc`	`memPtr = realloc( memPtr, new_size );`	Expands an existing block of memory to a `new_size`; current contents of `memPtr` are transferred to the expanded block

Function	Example	Description
strdup	char *newStr = strdup ("Abcde");	Duplicates a string; constitutes a memory allocation function because the results of strdup must be explicitly freed

Let me give you a final word of caution for developing proper memory management skills: Care for the relationship of automatic variables placed on the stack and references to dynamically allocated memory, which at some point must be returned to the heap.

Memory Leaks

Consider the following code fragment:

```
void someFunc( void )
{
    char *buf;

    buf = malloc( sizeof(char) * 256 );

     // do something

    return;
    // returning from the function results
    // in the frame being removed from the stack
}
```

The variable buf is local to someFunc and its value is lost when the function returns. The value, however, is a reference to memory allocated from the heap. If the function returns without an appropriate call to free, the memory can never be returned to the heap and is an illustration of a memory leak.

Finally, in our discussion of C syntax, style and convention are mechanisms for defining constants and macros within a program.

Definitions and Macros

As discussed previously, compiler directives are a means of directing the compiler to make decisions or perform actions.

In addition to the include directive seen earlier (refer to the Excursion in the section "The printf Command," on page 91, for an introduction to compiler directives), directives exist for defining constants and macros within a program.

The `define` Directive

Use the `define` compiler directive to instruct the compiler to perform a Search and Replace for the item being defined.

For instance, defining a constant `GLOBAL` in a header file included by multiple files is one method of managing global variables.

```
100: #ifndef GLOBAL
101: #define GLOBAL
102: #else
103: #define GLOBAL extern
104: #endif
105: GLOBAL int lineno;
```

Recognizing that variables declared external to extend their scope to multiple files must have an actual definition, the macro `GLOBAL` is a graceful way to accomplish it.

The first time the compiler sees this code fragment, `GLOBAL` is not defined, thus the compiler directive `ifndef` results in `true` and the compiler defines `GLOBAL`—albeit with an empty value.

```
101: #define GLOBAL
```

After you've defined `GLOBAL` to an initial empty value, line 105 appears as the *actual* definition for `lineno`.

```
105: int lineno;
```

Subsequent inclusions of this code fragment by other source files result in the test for `GLOBAL` as *not defined* (`ifndef`), resulting in `false` because `GLOBAL` was defined with the initial inclusion as the empty value. Therefore, the compiler directive `else` is performed, setting `GLOBAL` to `extern`. This time, when line 105 is reached, the variable `lineno` is defined as external.

```
105: extern int lineno;
```

The actual definition of the variable `lineno` is effectively in the first source file, which included the header containing this fragment. Subsequent inclusions result in `lineno` being referenced as an external variable.

More examples of macros and constants are seen in the Graphics Editor project.

Conclusion

The C programming language is a vastly complex language, and it literally takes years to master.

This chapter attempted to give you an introduction to the language syntax and style through example. You should now have sufficient exposure to face the more advanced issues that follow in the text, namely, the functions made available through the X Window System environment, and ultimately the challenge of creating the Graphics Editor.

Next Steps

At this point, you should have a level of confidence in the topics discussed so far, either through experimenting with the code samples provided or through independent study.

In the next chapter, focus shifts quickly from the introductory material consisting of use of the Linux operating system, understanding programming constructs, and employing the C programming language, to applying these concepts at an advanced level as we begin looking at the pieces that comprise the X Window System.

Part II

The Pieces of X

Chapter 4

In this chapter

- *Origin of the X Window System*
- *The Pieces of X*
- *Next Steps*

Windowing Concepts

Origins of the X Window System

Students involved in an effort known as *Project Athena* began the X Window System at MIT in the mid 1980s. Since the initial release for commercial use, the X Window System has seen many changes.

how tōō prō nouns′ it

The X Window System is called *X* for short.

Despite the similarity in names between *X Window* and *Microsoft Windows*, there is little consistency. The most notable difference is the separation that X maintains between the windowing environment and the operating system. Microsoft Windows, on the other hand, is a proprietary environment closely tied to the DOS operating system.

Many proponents of the Linux community want to adopt a naming convention that clearly illustrates the separation of the two systems. However, the course has been set, and there is little left to do beyond educating users.

Until recently, driven by a consortium of businesses such as Digital Equipment Corporation, Hewlett-Packard, Sun, IBM, and AT&T, the X Window System was given much focus to further its development and guide its evolution. With the release of revision 6, X has reached a level of maturity that will see it through the next few decades.

Although the products produced at MIT form the core of the X Window System, others in the industry have emulated, ported, and furthered it. Examples include companies such as Metro X, Hummingbird, Xi Graphics, and those involved with the efforts of XFree86, which is the X Window System port used in the Linux operating system.

The Pieces of X

This chapter focuses on gaining an understanding of windowing concepts necessary to program in the X Window System environment. If you are already acquainted with client/server models, immediate graphics, window hierarchy, window clipping, and event propagation and queuing, feel free to proceed to the next chapter.

Client/Server Model

The X Window System follows a client/server model that is unique to any windowing system. The model meets one of the goals set by the authors of the X environment in that it is fully extensible.

geek speak

The X Window Server is responsible for managing all resources available to a *display*.

Note

The term *display* in X Window vernacular does not refer only to the monitor associated with a workstation. A display is everything having to do with input and output for a specific system.

A typical display is a single monitor, keyboard, and mouse. However, it can be simply a touch-type plasma screen, or, it can consist of several monitors, a graphics tablet, and a keyboard.

Whatever the hardware configuration for your workstation, the X Server is responsible for its management.

Due to the relationship of the X Server and the specific hardware comprising a workstation, X Servers are not very portable. To manage the resources available to a particular video driver or monitor properly, the X Server must be very finely tuned at the hardware level.

Clearly, generic servers exist for many devices, but they will not maximize features or capabilities as a device-specific server would.

Note

For major platform support, manufacturers of the hardware generally author the X Server because they hold the details of the devices needed to do it properly. This is certainly the case with product lines produced by Sun Microsystems, Digital Equipment Corporation (Compaq), and Silicon Graphics.

The PC market has been the exception to this rule, because the vendors were not as interested as consumers in having X Servers for the myriad of device configurations available to this class of machine.

The manner in which you start the X Server varies slightly from system to system. Under the SYSV family of UNIX, the X Server is generally added to an *initlevel*.

Initlevel stands for initialization level as defined in the system file `/etc/inittab` (initialization table).

Effectively, the system is set to initialize to a specific level. A corresponding entry in the initialization table instructs the system, which proceeds to start to satisfy the desired level. If configured to start automatically, the X process will be added to an initlevel.

Under the BSD family of UNIX, an entry can be made in the `/etc/rc.local` file to start the server.

When the system is configured to automatically run X, the command for starting the server is `xdm`.

Executing the `xdm` command starts the X Display Manager, which enforces the user login process. After a user has successfully entered a login ID and password, `xdm` launches the X Server.

If the system is not configured to run X automatically, you can, after satisfying the normal UNIX login requirements, issue either the `startx` or `xinit` command to start X manually.

Once started, the X Server for the display hardware is responsible for servicing all requests made by an X client. For instance, a request by a client to draw a line or render a window is communicated to the server, which manages the hardware to satisfy the request. Also, any *event* generated by the hardware is queued for the client.

EXCURSION

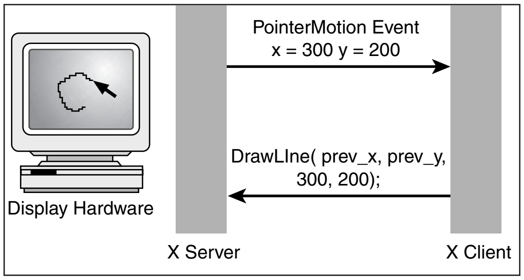
The Flow of Events in the X Client/Server Model

The X Window System is entirely *event driven*. The X Server communicates events generated in the display hardware to the X application (client). Events acted on by a client in turn generate requests of the server. Because the client may need time to process an event sent by the server, event queues are maintained by the server for all clients. The client is responsible for removing the events from the queue and processing them.

Figure 4.1 illustrates the flow of events and requests in the X client/server model.

Figure 4.1

The exchange of event generation and client requests.



As the mouse cursor (pointer) is moved across the screen in Figure 4.1, the X Server communicates the motion event (`PointerMotion`) to the X Client. Based on the client's function, the event can be acted on or ignored. In this illustration, the X Client is drawing a line to connect the previous location of the pointer (implied by the variables `prev_x` and `prev_y`) to the current pointer location. (This is the behavior of the pencil object in the Graphics Editor.)

Notice that the X Server in Figure 4.1 entirely separates the X Client from the display hardware. An X Client that attempts to communicate directly with the display sacrifices portability.

Events exist to represent all actions possible in the display hardware managed by the X Server. Table 4.1 shows some events understood by the X Server that will be processed by the Graphics Editor.

Table 4.1 X Events Needed by the Graphics Editor

X Event	*Is Generated When*
`PointerMotion`	The mouse moves.
`ButtonPress`	The user presses a mouse button. Fields in the event structure indicate which button was pressed.
`ButtonRelease`	The user releases a mouse button being pressed.
`KeyPress`	The user presses a button on the keyboard.
`EnterNotify`	The mouse enters a window.
`LeaveNotify`	The mouse leaves a window.
`Expose`	A window has been obstructed in such a way that the contents of the window need to be redrawn.

Understanding that an X Server must exist for the display in which the X Window System will run, we now introduce the X Client in more detail.

X Clients

A critical step when structuring an X Client application is establishing a connection to the X Server. (The X library call to perform this task is discussed in Chapter 6, "Components of an X Window Application.") The communication between an X Client and the X Server occurs using a standard network protocol such as TCP/IP.

geek speak

An X Client is any X-based application that communicates display requests through an X Server.

Note

Understanding the details of the network over which the client and server communicate is not necessary for creating X clients.

What is important is that a network layer exists on the system on which you are running the server and clients. Because TCP/IP is inherent to UNIX, and specifically Linux, this should never be a problem. If you feel it is, consult the Linux setup to ensure that network services are started by the system.

Note

When running an X Client, the server and the client processes typically execute on the same system. However, because the communication between the two processes is network based, it is possible to run a client on one machine and have it displayed on another. Examples of this will be covered in Chapter 6 where we demonstrate how clients choose to which server to connect.

Table 4.2 shows several X Clients provided by the Linux operating system and a brief description of their functions.

Table 4.2 Linux X Clients

Command	*Description*
`xterm`	Terminal emulator for X windows
`xset`	Enables user to set preferences for the display
`xmessage`	Displays a window containing a message
`xman`	Manual page browser
`xload`	Displays a periodically updated histogram of the system load average
`xfontsel`	Application to display fonts known to the X Server
`xeyes`	Watches what you do and reports to the Boss
`xev`	Creates a window and displays all event information sent by the server
`xedit`	Simple text editor
`xdpyinfo`	Utility for displaying information about an X Server
`xdm`	Manages an X display by prompting for login name and password, authenticating the user, and running a "session"
`xconsole`	Displays messages that are usually sent to `/dev/console`
`xclock`	Displays time in analog or digital
`xcalc`	Scientific calculator desktop accessory

Many X Clients exist to perform a variety of tasks ranging from image manipulation to integrated application development. Further, entire environments are provided by packages such as CDE, KDE, and GNOME. These packages determine the look and feel of a tailored X session and provide embedded clients and utilities specific to the environment. Included in these environments are X Clients to manage files and printers, access the Internet, or manage your Internet service provider dial-up.

EXCURSION

The Critical Role of the Window Manager

An X Client known as a *window manager* is key to any desktop environment. A window manager is a special purpose application that provides the capability of X applications to be moved, resized, minimized, and restored dynamically by the user.

The window manager applies *decorations* to an application that enable the user to access these features. Figure 4.2 shows the window decorations and their purpose.

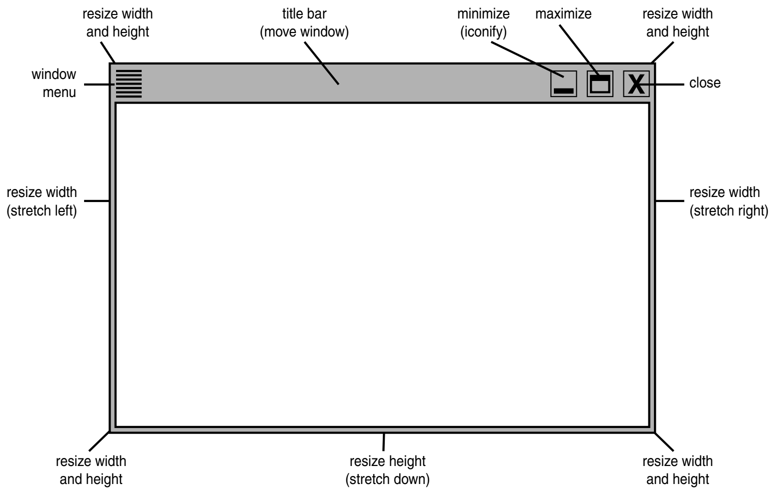


Figure 4.2

Window decorations provided by the Enlightenment window manager.

By using these decorations, a window can be dynamically altered in a number of ways.

Many window managers exist for use with the X Window System. The Enlightenment window manager is shown in Figure 4.2 and is provided as part of the GNOME environment included in Red Hat's distribution of Linux. The Tab Window Manager has been released as part of the standard distribution of the X Window System.

Window managers generally provide similar functionality with common management features applied to windows. The differences lie in the *look and feel* imposed, or the style of the decorations applied to windows.

Note

Many resources exist for becoming acquainted with the clients available under the Linux operating system. With the current focus on the structure of the X Window System and not on specific clients, realize that many clients exist to serve a variety of needs. Any need recognized as yet unanswered is a call for your contribution.

To review, the X Server manages all display resources and relays events that occur in the hardware to the clients it serves. Further, the clients process the events and generate server requests.

Requests instruct the X Server to perform tasks such as creating, positioning, coloring, destroying, rendering, drawing, and clearing windows.

Windows

The task list that the X Server understands is quite extensive. Every request made to the X Server is relative to a window within the application.

Note

In terms of application development, invoking functions contained in the X Library `libX.a` generates an X Server request. (Review the section "`gcc -l`" in Chapter 1, page 28 for a description of function libraries.) The contents of the X Library will be discussed at length in Chapter 6.

A window is the basic element of an X Window application. By arranging multiple windows within the application, the program forms its purpose or usefulness.

Consider the `xcalc` application, as seen in Figure 4.3.

Every component of the `xcalc` application consists of a window. Every one of the buttons forming the keypad of the calculator is a window with another window acting as the display panel.

As the cursor moves through the windows of the application, it generates events and places them in a queue that the application processes. For instance, as `EnterNotify` events occur for the different windows, the application processes it by highlighting (increasing the border width) of the window that was entered. When the `LeaveNotify` event is processed, the window border returns to its original width.

The `ButtonPress` event is important to the `xcalc` application. Many things must occur within the program when a `ButtonPress` event is seen.

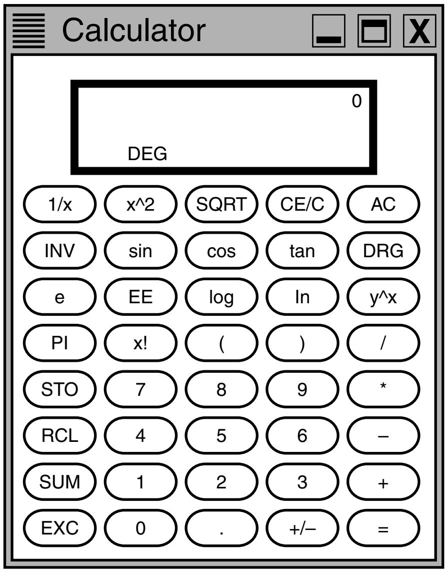


Figure 4.3

The `xcalc` *X Client.*

1. A request is made to the server for a change in the window colors. The application requests that the foreground color be used for the background and the background color of the window be used in the foreground. The inverted colors give the appearance of a button being pushed, as seen in Figure 4.4.

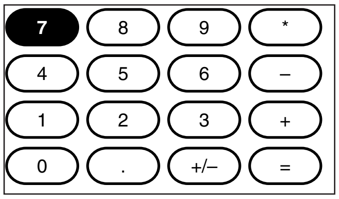


Figure 4.4

`ButtonPress` *event, step 1.*

2. Another request is then made to the X Server to display in the status window the value represented by the window in which the `ButtonPress` occurred, as seen in Figure 4.5.

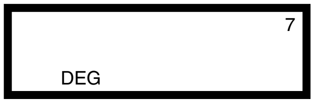


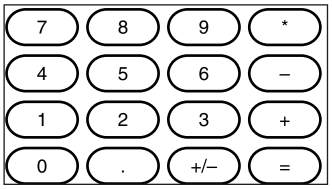
Figure 4.5

`ButtonPress` *event, step 2.*

When the button is released and the `ButtonRelease` event is generated, the colors must be returned to their original values, as seen in Figure 4.6.

Figure 4.6

`ButtonRelease` *event.*



As demonstrated in the previous example, processing the many events that occur in the various windows of an application, and subsequently generating requests from them, is the nature of event-driven programming.

In addition to understanding the purpose of processing events communicated by the X Server, it is important to recognize the relationship between the multiple windows of an application.

Window Hierarchy

The `xcalc` application, as with all X Window applications, consists of a hierarchy of windows. The more sophisticated the application the greater the complexity of the hierarchy.

Window hierarchy is used to group windows based on common function. This is particularly useful when a window grouping is conditionally mapped to the screen. For instance, a file selection dialog box only appears when the user prompts to save or load a file.

Note

The term mapping a window refers to a request for the X Server to display it or make it visible on the screen. To unmap a window means to remove it from view.

The programmer, by the parent specified to the window creation function, determines the window hierarchy of an application.

EXCURSION

The Effect of Window Parenting on Window Visibility

Window creation is covered in Chapter 6. However, every window has a parent and the relationship between a window and its parent imposes a behavior that must be understood to ensure proper management.

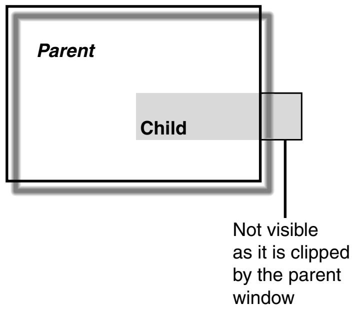
Specifically, if a parent window is not visible on the screen, its children (descendents) will not be visible either.

Further, any portion of a window extending beyond the dimensions of its parent will be clipped by the parent and not appear onscreen.

Figure 4.7 illustrates the concept of window clipping, as the portion of the child not fully contained within the parent window is not visible.

Figure 4.7

Window clipping forced by a child extending beyond a parent's bounds.



You can further understand the relationship between a parent and a child window by describing *window origin*.

The origin of any window is its upper-left corner. When placing a window within its parent, the position (location) specified is always relative to the parent's origin.

Examples of how to manage the placement of windows are provided in Chapter 6.

Figure 4.8 shows conceptually the relationship between the multiple windows of the `xcalc` application.

The toplevel window in Figure 4.8 is the window decorated by the window manager as a means of providing the application with capabilities such as resizing, minimizing, and positioning.

The toplevel window also has the responsibility of managing all the application's child windows. This responsibility includes ensuring that the windows are visible and correctly placed and that the events sent to the program are dispatched to the appropriate window.

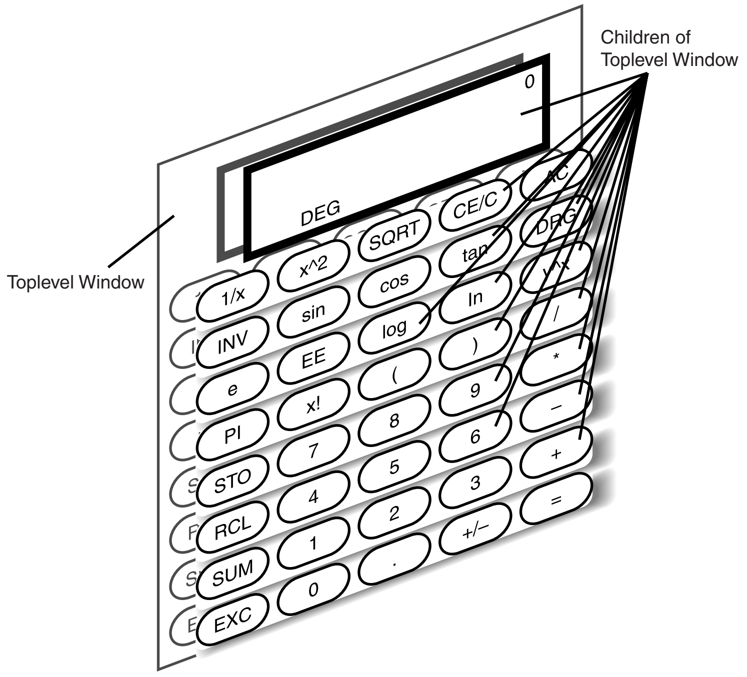
The usefulness of several of the events dispatched was illustrated with discussions of `ButtonPress`, `ButtonRelease`, `EnterNotify`, and `LeaveNotify`. We will now consider the `Expose` event as shown in Table 4.1.

Expose

Processing the `Expose` event is crucial to X-based applications due to the *immediate graphic nature* of X.

Figure 4.8

`xcalc` window hierarchy.



geek speak

The immediate graphic nature of X refers to the absence of an inherent mechanism for redrawing graphics contained in windows.

A primary reason for creating windows in an application is to communicate something to the user. Whether to provide the keypad associated with the `xcalc` application or a canvas for drawing graphical objects as in the Graphics Editor, windows communicate either textually or by employing graphics.

The contents of the windows associated with the keypad of `xcalc` were explicitly placed to provide the functionality of the application. Because of the immediate nature of X, no mechanism is within the environment to replace the contents of these windows if they are erased or destroyed.

Note

The X Server does not retain a window's contents, but places them in the window when the program makes an explicit request. After this they are forgotten. If a window is unmapped or obscured by another window, its contents are lost.

Rather than attempting to retain and replace the contents of all of the windows in an application, the X Server will send an `Expose` event to notify the program that a redraw is necessary. Details within the `Expose` event structure will indicate which window requires updating, the position and dimensions of the region affected, and the number of `Expose` events pending in the event queue.

The immediate graphic nature of X adheres to a sound principle of software development: 90 percent of the effort should not be extended to benefit 10% of the users.

The task of maintaining and replacing the contents of all windows serviced by the X Server is daunting. Each X Client could conceivably have dozens of windows associated with the application. Further, the X Server is servicing dozens of applications, amounting to many windows. The computing power and memory required to accomplish this task by the server could make the environment unusable.

However, the application that owns the window has already assigned the initial window contents. The task of the application updating the window as necessary is trivial in comparison. For that reason, X sends the `Expose` event notifying the program when an update is necessary.

Next Steps

With the understanding of windowing concepts gained in this chapter, you are ready to learn how aspects requiring constant management in an X-based application can be automated through the use of objects known as *widgets*.

Chapter 5, "Widget Sets," will lead you through a discussion of the X Intrinsic Toolkit and the widget objects it manages.

Chapter 5

In this chapter

Widget Sets

Authoring an X Client requires constant attention to many details. The relationship between using windows within the application's hierarchy, processing events in the queue (ensuring critical events are processed quickly), and anticipating and responding to user input are a few of the considerations when writing an application for the X Window environment.

Much of the management necessary within an X-based application is greatly simplified through use of the X Toolkit Intrinsics, known as *Xt*.

As Chapter 4, "Windowing Concepts," led a discussion of X at the lowest level, this chapter demonstrates the relationship between widgets and windows, and the toolkit that manages them.

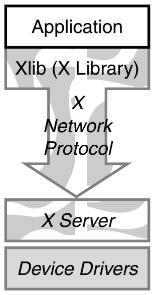
If you are already familiar with the X Toolkit Intrinsics and the Athena widget set, feel free to continue to the next chapter where you will build your first X Window application.

The Power and Convenience of Using Widget Sets

All aspects of creating an X Window application can be accomplished using only the X library known as Xlib, which is the lowest layer of X library functions. Figure 5.1 shows the architecture of an application employing only Xlib.

Figure 5.1

Components and architecture of an Xlib application.



As demonstrated in Figure 5.1, the application employs the X library to form the requests made of the server.

Specifically, the application invokes functions from the X library, which formats the request into a network call for communicating them to the server. The X Server, in turn, is responsible for translating the request to the specific devices it is responsible for managing.

geek speak

By ensuring that all communication to devices targeted by an application is accomplished solely through the X Server, the application maintains maximum *portability*.

Many details accounted for in the application in Figure 5.1 are left to the imagination. As discussed in Chapter 4, elements of event management, refreshing graphics, and responding to user input add complexity to the program.

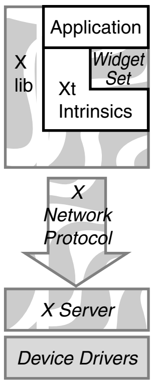
However, through use of a toolkit and a set of *widgets* the tasks common to most application are simplified. Figure 5.2 shows the relationship of an X-based application to the X libraries and toolkit.

geek speak

A *widget* adheres to the principles of object-oriented methodology and is, in fact, an object.

Figure 5.2

Architecture of an Xt application.



In Figure 5.2, side relationships illustrate the association between each component of the architecture. The application in this figure is employing the X library (Xlib) as well as the X Toolkit Intrinsics (Xt) and a *widget set*.

Note

In studying the side relationships of the components in Figure 5.2, notice Intrinsics supports the widget set and Xlib supports Intrinsics. These relationships are critical in the X environment because a widget set would be of little use when writing an X-based program without the use of Xt.

Because Xt is a toolkit, its purpose is to simplify the use of Xlib and to manage a widget set. The relationship between Xlib, Xt, and a widget set are intricate.

Consistent with Figure 5.1, however, is that all communication with the devices displaying the application are managed by the X Server in order to maintain portability.

Chapter 4 discussed X Window concepts demonstrating the hierarchy of windows within the `xcalc` application. Windows are the basic building block of an X application and are the primary management function of the X library.

Adding a toolkit to the X library, as X Toolkit (Xt) does, maintains the window as the basic component of an application. However, as stated earlier, the toolkit simplifies use of the X library (Xlib) and has as its primary management function *widgets*.

geek speak

A *widget set* is a group of components that manage different aspects of a graphical user interface. Elements such as menus, buttons, scrollbars, and text fields are entities provided by a widget set.

As demonstrated with the `xcalc` application in Chapter 4, section "Windows," page 119, processing all the events pertinent to a window within an application is the responsibility of the programmer. An example is acting upon `EnterNotify` and `LeaveNotify` for modifying the border width of the window as the cursor moves through the application. Another example is managing `ButtonPress` and `ButtonRelease` for inverting the colors of the window to give the appearance of a button being pushed. A final example of an area that an Xlib programmer must address is invoking the necessary procedure for responding to the user's selection of a value or function associated with a window.

A *button* component that monitors and processes all the events pertinent to changing colors and border widths or invoking functions is provided by a widget set.

As a programmer, placing the *button* widget provided by the widget set on the interface of the application frees you from the mechanics of making a window look like a button. Because a window is still the basic component of any X application, it should be understood that the widget creates and manages a window.

The actual components provided within a widget set vary based on the set being used. Two common widget sets are *Athena* and *Motif*. Widget sets, in general, provide the programmer with a variety of objects to employ for accomplishing a graphical user interface.

Note

Because the X Toolkit supports any widget set used within an application, migrating from one widget set to another is largely an exercise in editing. In other words, the naming conventions differ between widget sets but the underlying mechanics are similar. To move from one widget set to another can be as simple as editing the names of the widgets and their *resources* to those understood by the new widget set.

Choosing between one set of widgets and another is largely a matter of determining the desired *look and feel* for the application.

Another deciding factor is the sophistication of the components provided by the widget set. Dictated by the purpose of the application, components of certain function or capability may be required within the application, forcing the selection of one widget set over another.

A *look and feel* is the appearance and layout of the application. Consistency in the appearance and position of components in an application is critical for ensuring that users develop a comfort level for using and navigating the product.

Widget *resources* are the mechanism by which the widgets remain extensible by the user.

Resources are the attributes of an application at the component (widget) level. For instance, it *may be possible* through the resources available to a widget for the user to select the font, color, and placement of a button within an application.

The caveat *may be possible* is applied because the user cannot alter any resource specified by the programmer (hard coded into the application).

Examples of specifying resources are provided in the next chapter, "Components of an X Window Application."

The Athena widget set is available as part of the standard release of X and provides basic components for addressing nearly every requirement of a graphical user interface. It lacks the sophistication of other widget sets, but this simplicity makes it an easy first widget set to learn.

Because the Athena widget set is used in the Graphics Editor project, it will be the primary focus of the remainder of this chapter.

The Athena Widget Set

As described, a *widget* is a component used to implement some aspect of a graphical user interface. Widgets account for menu components, scrollbars, text entry, push buttons, frames, and more.

Widgets are objects in the sense that they have internal structures and methods that are opaque to the programmer. Influencing these objects is done exclusively through the X Toolkit Intrinsics, or convenience routines provided by the widget set. Further, following object-oriented methodology, widgets obey a class hierarchy as imposed by their authors. Because every widget belongs to a *class*, all widgets are represented in the hierarchy for the widget set in which they belong.

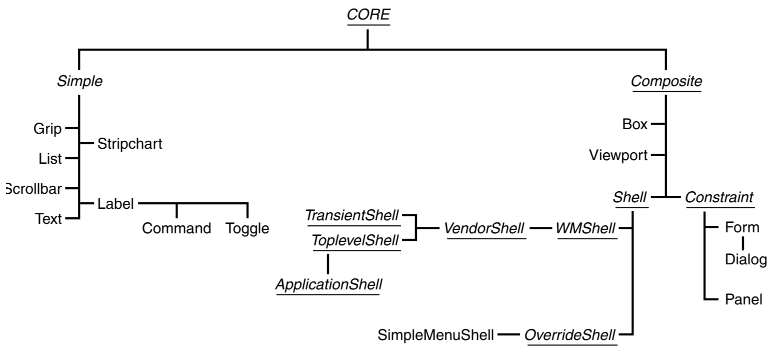
A widget *class* groups widgets based on common appearance and behavior. As a hierarchy implies, there is inheritance from one class in the hierarchy to its descendents.

Widgets inherit resources (attributes) and functionality from their ancestors. Knowing a widget's position in the class hierarchy aids in determining the capabilities of the widget and the appropriate time to employ it.

Figure 5.3 shows the class hierarchy of the Athena widgets.

Figure 5.3

Athena class hierarchy.



Note

The widgets within the class hierarchy shown in Figure 5.3 provided by Intrinsics are underlined, but those included as part of the Athena widget set are plain text.

Intrinsics provides several basic widgets to serve as *super-classes* for vendor specific widgets. Also, some widgets are either so general or so special in purpose that it shouldn't be necessary for them to be repeated by every vendor who wants to create a unique widget set.

Every widget set, despite the vendor, will fold into the widgets provided by Intrinsics.

Note

A *super-class* is any widget designated as the direct ancestor for subsequent widgets. For instance, in Figure 5.3, the super-class of the `Command` widget is the `Label` widget and the super-class of the `Simple` widget and `Composite` widget is the `Core` widget.

Consider again the idea of inheritance as we review the Athena class hierarchy shown in Figure 5.3.

Any resource available to a widget is available to its descendants. For this reason, resources are not repeated in the structure of the descendants. Similarly, the functions or methods available in the ancestor are available for inheritance as well.

Some widgets within the hierarchy exist for no other purpose than to provide a common set of resources or functionality to their descendants. The `Core` widget that heads the hierarchy is an example of a widget not meant to be *instantiated* directly but instead provides features to its descendants (which you will note extends to all other widgets).

geek speak

The term *instantiation* refers to a widget being created within an application. Think of it as creating an *instance* of the widget.

Resources made available to all widgets by merit of inheritance from the `Core` widget include colors, fonts, coordinates, width, and height. Because these attributes are necessary for all widgets, the Core class ensures that they are included at the lowest level of the hierarchy.

Functionality made available by a widget's ancestor is largely the determining factor when selecting the super-class for a new widget. Consider, as an example, the `Command` widget and its `Label` super-class. Everything the `Label` widget can do (namely continuously display a text string or image) is required by the `Command` widget. However, the `Command` widget extends the capabilities of the `Label` widget. The `ButtonPress` and `ButtonRelease` events are ignored by the `Label` widget, whereas the `Command` widget acts upon them and toggles the widget's window colors and invokes a `callback` function.

By inheriting all methods from its ancestors, widgets in the descending layers of the hierarchy grow in complexity.

The following sections provide a description of the widgets used in the Graphics Editor project and the resources available to each. Although there are many more widgets available from the Athena widget set, for the sake of brevity only those needed by the Graphics Editor project are discussed in this text.

The Core Widget

Not explicitly created within the Graphics Editor application, but as the root of the class hierarchy seen in Figure 5.3, the `Core` widget provides many characteristics important to widgets that are instantiated.

Note

The `Core` widget is sometimes used as a drawing area, but otherwise it is rarely instantiated.

Table 5.1 shows the resources defined by the `Core` widget class.

Table 5.1 Core Widget Resources

Name	*Class*	*Type*	*Default Value*
`accelerators`	`Accelerators`	`AcceleratorTable`	`NULL`
`ancestorSensitive`	`AncestorSensitive`	`Boolean`	`True`
`background`	`Background`	`Pixel`	`XtDefaultBackground`
`backgroundPixmap`	`Pixmap`	`Pixmap`	`XtUnspecifiedPixmap`
`borderColor`	`BorderColor`	`Pixel`	`XtDefaultForeground`
`borderPixmap`	`Pixmap`	`Pixmap`	`XtUnspecifiedPixmap`
`borderWidth`	`BorderWidth`	`Dimension`	`1`
`colormap`	`Colormap`	`Colormap`	Parent's `Colormap`
`depth`	`Depth`	`int`	Parent's `Depth`
`destroyCallback`	`Callback`	`XtCallbackList`	`NULL`
`height`	`Height`	`Dimension`	*widget dependent*
`mappedWhenManaged`	`MappedWhenManaged`	`Boolean`	`True`
`screen`	`Screen`	`Screen`	Parent's `Screen`
`sensitive`	`Sensitive`	`Boolean`	`True`
`translations`	`Translations`	`TranslationTable`	*widget dependent*
`width`	`Width`	`Dimension`	*widget dependent*
`x`	`Position`	`Position`	`0`
`y`	`Position`	`Position`	`0`

Table 5.2 shows the descriptions of each of the items in Table 5.1.

Table 5.2 Descriptions of `Core` Widget Resources

Name	*Description*
`accelerators`	List of *event to action* bindings to be executed by this widget when events occur in other widgets—allows for application *shortcuts*
`ancestorSensitive`	The state of sensitivity for a widget's ancestors—a widget is insensitive if either it or any of its ancestors are insensitive. An insensitive widget will appear as grayed-out and will not process events
`background`	A value used to index into the widget's *colormap* to determine the background color of the window associated with the widget
`backgroundPixmap`	The background *pixmap* applied to the widget's window—if this resource is set, the pixmap specified is used instead of the widget's background color.
`borderColor`	A pixel value used to index the widget's colormap to determine the border color of the widget's window
`borderPixmap`	The border pixmap applied to this widget's window—if this resource is set, the pixmap specified is used instead of the border color.
`borderWidth`	The width of this widget's window border
`colormap`	The colormap used by the widget
`depth`	The depth used for this widget's window
`destroyCallback`	A function list called when the widget is destroyed
`height`	Specifies the height of the widget
`width`	Specifies the width of the widget
`mappedWhenManaged`	If `True`, the widget's window will automatically be mapped by the Toolkit when it is managed—otherwise, an explicit request to map the window will be necessary.

EXCURSION

The Vernacular of an X Window Programmer

A number of terms and types referenced in the `Core` widget's resource list have not been previously introduced.

The following list is a mini-glossary to explain these new concepts.

`Boolean` Data type used to represent a `True` (defined as 1) or `False` (defined as 0) value

`Callback` A function registered with the widget by the programmer after instantiation—`callbacks` are specific to a reason such as `ButtonPress`, `EnterNotify`, `CreateNotify`, `DestroyNotify`, and more. The function is invoked by the widget when an event corresponding to the callback reason is received.

Colormap An array of elements defining color values—the colormap associated with a window is used to display the contents of a window; each pixel value is an index into the colormap to produce a red, green, and blue component that drives the guns of a monitor. Depending on hardware limitations, one or more colormaps can be installed at any one time.

Depth The number of bits per pixel of the window or drawable

`Dimension` The `Dimension` data type is defined within the X library for specifying width and height values for widgets.

Drawable Either a window or pixmap used as either the source or destination of a graphic operation

Pixel A value used to index into the colormap of a window for determining the actual color to be displayed

Pixmap A two-dimensional array of pixels used for displaying icons or applying texture in an X application

Screen An X display server can provide several screens, which are typically independent physical monitors. A `Screen` structure is the data type containing the information about the screens controlled by the server.

Translations A *translation* maps an event into an action name understood by the widget. Translations and actions can be added to widgets based on user needs. After a translation is installed in a widget, the named action will be invoked when the specified event occurs in the widget.

XtCallbackList A list of Callbacks registered with a widget for a given reason—all functions in the list are invoked when the corresponding event (reason) occurs in the widget, although the order in which they are invoked cannot be predicted.

EXCURSION

The `ApplicationShell` *Widget*

Returned by the Intrinsics function call that opens a connection to the X display server, the `ApplicationShell` widget serves as the root widget for the *instance hierarchy* of an application.

Note

Differing from the class hierarchy shown in Figure 5.3, an application's *instance hierarchy* is dependent on the order in which the program's author creates the widgets.

An instance hierarchy indicates the relationship of the widgets created in the application and, specifically, the manner of parenting specified by the programmer.

Table 5.3 shows the resources introduced by the `ApplicationShell` widget.

Table 5.3 `ApplicationShell` Widget Resources

Name	*Class*	*Type*	*Default Value*	*Description*
argc	Argc	int	0	Stores the value of `argc` passed to the function `main`
argv	argv	StringArray	NULL	Stores the values of `argv` passed to the function `main`

Note

The utility of X knowing the parameters made available to the program (`argv`) will become clear in Chapter 6 when we take a detailed look at the function used to connect to the X Server.

In brief, a standard set of parameters is accepted by all Xt-based applications. The presence of the accepted parameters is made known by providing the variables `argc` and `argv` to the server initialization call.

Table 5.4 shows the resources inherited by the `ApplicationShell` as a descendant of the `ToplevelShell` widget.

Table 5.4 `ToplevelShell` Widget Resources

Name	*Class*	*Type*	*Default Value*	*Description*
iconName	IconName	String	NULL	Specifies the name to be applied to the icon when the application is minimized.
iconNameEncoding	IconName Encoding	Atom	NULL	As X supports internationalization, it is necessary to specify how the name is encoded.
iconic	Iconic	Boolean	False	Determines whether the program is started in the iconified or minimized state—the default value is that it is not minimized at startup.

Table 5.5 shows a few of the resources inherited by the `ApplicationShell` as a descendant of the `Shell` widget class.

Table 5.5 `Shell` Widget Resources

Name	*Class*	*Type*	*Default Value*	*Description*
`allowShellResize`	`AllowShellResize`	`Boolean`	`False`	Specifies whether the user will be able to use the decorations applied by the window manager to dynamically change the width and height of the application.
`geometry`	`Geometry`	`String`	`NULL`	An alternate way to specify the initial placement and dimensions of the application. The geometry string follows the form `widthxheight+/-xoffset+/-yoffset` and is interpreted relative the root window (desktop) .

Referring again to the Athena class hierarchy shown in Figure 5.3, the `ApplicationShell` is a descendant of the `ToplevelShell` widget class. As is true for all shell widgets, it is only able to parent a single child. For this reason, a class of widgets known as `Composite` widgets exists in order to serve as *managers* of other widgets.

Widgets That Manage Other Widgets

Two forms of manager widgets are employed in the Graphics Editor project—the `Box` and `Form` widget. Each widget has a unique approach for managing the widgets that it parents.

The `Box` Widget

The `Box` widget, the simpler of the two, provides management of an arbitrary number of widgets by containing them in a box of specified dimensions.

The children of the `Box` widget are rearranged when the parent is resized or when children contained by the box are managed, unmanaged, or resized. The `Box` widget always attempts to pack its children as tightly as possible within the geometry allowed by the parent.

Box widgets are most commonly used to manage a related set of buttons, but their children are not strictly limited to being only buttons.

Table 5.6 shows the resources introduced by the Box widget.

Table 5.6 Box Widget Resources

Name	*Class*	*Type*	*Default Value*	*Description*
hSpace	HSpace	Dimension	4	The amount of horizontal space left between each child of the Box widget
vSpace	VSpace	Dimension	4	The amount of vertical space left between each child of the Box widget
orientation	Orientation	Orientation	XtOrientVertical	Determines whether the children are oriented vertically (XtorientVertical) or horizontally (XtorientHorizontal)

The Form Widget

Like the Box widget, the Form widget can contain an arbitrary number of children; however, the Form widget provides strict geometry management of its children.

The Form's management style is to control the position of each of the children by asking the programmer to specify the relative placement of all widgets managed by the Form. When a Form widget is resized, it computes new positions and sizes of its children while attempting to maintain the relative positions specified when each child was added to the form.

The default width of the Form widget is the minimum required for holding all its children after computing their initial layout. If a width and height are assigned to the Form widget that are insufficient to house all its children, window clipping can occur.

Table 5.7 shows the resources introduced by the Form widget.

Table 5.7 Form Widget Resources

Name	*Class*	*Type*	*Default Value*	*Description*
defaultDistance	Thickness	int	4	The default distance to use as spacing between the Form widget's children

Table 5.8 shows the resources inherited by the `Form` widget as a descendent of the `Composite` widget. These resources enable children of the `Form` to specify individual layout requirements.

Table 5.8 Composite Widget Resources

Name	*Class*	*Type*	*Default Value*	*Description*
`bottom,` `left,` `right,` `top`	`Edge`	`XawEdgeType`	`XawRubber`	Where to place the corresponding edge of the widget when the `Form` widget is resized
`fromHoriz,` `fromVert`	`Widget`	`Widget`	`NULL`	Specifies the widget this child should be placed underneath or to the right of respectively—the default value of `NULL` indicates that the widget should be positioned relative to the corresponding edge of the parent
`horizDist` `vertDistance`	`Thickness`	`int`	`defaultDistance` `(Form resource)`	Sets the amount of space between this child and its horizontal or vertical neighbor respectively
`resizeable`	`Boolean`	`Boolean`	`False`	Determines whether the `Form` widget will ignore geometry requests made by the child—note that this doesn't prevent the `Form` from resizing the child.

Allowable values for the bottom, left, top, and right resources include

`XawChainBottom`	Ensures the corresponding edge of the widget remains a fixed distance from the bottom of the `Form` widget
`XawChainLeft`	Ensures the edge of the widget remains a fixed distance from the left edge of the `Form` widget
`XawChainRight`	Ensures that the edge of the widget remains a fixed distance from the right edge of the `Form` widget
`XawChainTop`	Ensures the edge remains a fixed distance from the top of the `Form` widget
`XawRubber`	Enables the corresponding edge to move a proportional distance relative to the new size of the `Form` widget

Examples of the semantics for employing the constraint resources of the `Form` widget are provided in Chapter 13, "Application Structure."

The final two Athena widgets to consider are the `Label` widget and the `Command` widget.

The `Label` Widget

The `Label` widget is responsible for displaying a text string or `Pixmap` within a rectangular region of the screen.

The label can contain multiple lines of text or a single `Pixmap` image. When displaying a string, the `Label` widget supports text left, right, or center justification. The `Label` widget cannot be directly selected or edited and is provided by Athena for output only.

Table 5.9 shows the resources introduced by the `Label` widget.

Table 5.9 `Label` Widget Resources

Name	*Class*	*Type*	*Default Value*
`bitmap`	`Bitmap`	`Pixmap`	None
`encoding`	`Encoding`	`unsigned char`	`XawTextEncoding8bit`
`font`	`Font`	`XFontStruct *`	`XtDefaultFont`
`foreground`	`Foreground`	`Pixel`	`XtDefaultForeground`
`internalHeight`	`Height`	`Dimension`	`2`
`internalWidth`	`Width`	`Dimension`	`4`
`justify`	`Justify`	`Justify`	`XtJustifyCenter`
`label`	`Label`	`String`	Widget instance name
`leftBitmap`	`LeftBitmap`	`Bitmap`	None
`resize`	`Resize`	`Boolean`	`True`

Table 5.10 shows descriptions of the resources introduced by the `Label` widget.

Table 5.10 Descriptions of `Label` Widget Resources

Name	*Description*
`bitmap`	The `Pixmap` to be displayed instead of the character string—this enables the creation of icons that invoke actions.
`encoding`	The `encoding` method enables internationalization and is used to interpret the value of the `label` resource. The allowed values are `XawTextEncoding8bit` and `XawTextEncodingChar2b`.
`font`	The font to use when displaying the text string specified by the `label` resource

`foreground`	The pixel value used to index into the widget's colormap to determine the foreground color of the widget's window
`internalHeight`	The minimum space left between the `label` or `bitmap` resource value and the vertical edges of the window associated with the widget
`internalWidth`	The minimum space left between the `label` or `bitmap` resource value and the horizontal edges of the window associated with the widget
`justify`	Specifies the justification of text displayed in the `Label` widget—allowable values are `XtJustifyLeft`, `XtJustifyCenter`, and `XtJustifyRight`.
`label`	Sets the character string to be displayed in the window associated with the widget
`resize`	Specifies whether the widget should attempt to resize to its preferred dimensions anytime its resources are modified

Last to consider is the `Command` widget. As described earlier, the `Command` widget extends the capabilities of its super-class, the `Label` widget.

The `Command` Widget

The `Command` widget is a rectangular area containing a character string or `Pixmap` image much like the `Label` widget; however, it accepts input for selection.

how too prō nouns′ it

Although spelled *c-o-m-m-a-n-d*, it is pronounced as if it were written *button widget*.

When the mouse cursor enters a `Command` widget, the widget highlights by increasing the width of its window border. This highlighting indicates that the widget is ready for selection. When the left mouse button is pressed, the `Command` widget responds by reversing its foreground and background colors. Upon the release of the left mouse button, the colors are returned to the normal values and a `notify` action is invoked that will call all functions registered on the widget's callback list. If the mouse cursor leaves the widget's window before the button is released, the button reverts to its normal colors and does not invoke the `notify` action.

Table 5.11 shows the resources introduced by the `Command` widget. Remember that all resources associated with the `Command` widget's super-class are available as well.

Table 5.11 `Command` Widget Resources

Name	*Class*	*Type*	*Default Value*
`callback`	`Callback`	`XtCallbackList`	`NULL`
`cornerRoundPercent`	`CornerRoundPercent`	`Dimension`	`25`
`highlightThickness`	`Thickness`	`Dimension`	`2`
`shapeStyle`	`ShapeStyle`	`ShapeStyle`	`Rectangle`

Table 5.12 shows descriptions of the resources introduced by the `Command` widget.

Table 5.12 Descriptions of `Command` Widget Resources

Name	*Description*
`callback`	List of routines called when the notify action is invoked
`cornerRoundPercent`	When a `shapeStyle` of `roundedRectangle` is specified, `cornerRoundPercent` determines the radius of the rounded corner. This radius is a percentage of the length of the shortest side of the `Command` widget.
`highlightThickness`	Thickness of the highlighted rectangle used to alert when the widget is ready for selection
`shapeStyle`	Used to create non-rectangular widgets

Examples using the Athena widgets will be provided in subsequent chapters. Understand for now the functionality introduced by each of the widgets and the resources associated with them.

Other widget sets exist that are significantly more complex than the Athena. The following section introduces the Motif widget set.

The Motif Widgets

The Motif widget set is the product of the Open Software Foundation (OSF) and despite implications of the name, the widget set is not free.

Motif is an industry standard widget set that has nearly become synonymous with the X Window System. Its use is widespread in the professional development community because it is considered a serious widget set for commercial software development.

Some in the Linux community hope to eventually have available a free Motif-*like* widget set by merit of the efforts of those working on a Motif clone known as LessTif (as opposed to *Mo*-tif).

The Motif widget set is significantly more complex than Athena and is therefore more difficult to learn. Some parallels exist between the widget sets; however, Motif offers many more widgets with greater capabilities. Further complicating the Motif widget set is that many of Motif's features are controlled through *indirect* resources.

geek speak

An *indirect resource* refers to a resource of one widget affecting the behavior of another.

Mastering Motif resources requires sifting through volumes of information.

To complete this brief introduction of Motif, the following section shows the similarities and differences between Motif and what you now know about the Athena widget set.

The `Core` and `ApplicationShell` Widgets

Refer to the class hierarchy shown in Figure 5.3 and notice that the `Core` and `ApplicationShell` widgets are provided by the X Toolkit Intrinsics. Because the Athena widget set class hierarchy folds into those widgets provided by Xt, the `Core` and `ApplicationShell` widgets also appear in the Motif widget set.

The other widgets introduced so far, however, are provided by Athena and will require a corresponding widget in the Motif widget set.

The `XmRowColumn` Widget

The `XmRowColumn` widget provided by Motif is closely compared to the Athena `Box` widget. The management styles of the two widgets are similar except that the `XmRowColumn` enables the programmer to specify preferences not available to the `Box` widget.

The `XmRowColumn` is used widely as a super-class within the Motif widget set, providing for menu bars, menu panes, and radio or check box buttons. Some indirect resources specific to the `XmRowColumn` for supporting these rolls include

> `adjustLast` and `adjustMargin` override the margin resources of `Label` and `Button` widget children.
>
> `isAligned` and `entryAlignment` override the alignment resources of any `Label` or `Button` widget children.

Like the Athena `Box` widget, the Motif `XmRowColumn` widget provides direct resources for controlling its layout, orientation, borders, and spacing. However, making it more extensible than the `Box` widget, the `XmRowColumn` enables the programmer to specify the number of columns for vertical orientation or number of rows for horizontal orientation in which to arrange its children. As was mentioned of the `Box` widget, it always packs its children as tightly as possible; the Motif `XmRowColumn` widget, however, allows the packing style to be set.

Consider now the Athena `Form` widget and the Motif equivalent.

The `XmForm` Widget

Both the Athena and Motif widget sets provide a `Form` widget for strict management of widget children. However, as was true with the Motif `XmRowColumn` widget, the Motif `XmForm` widget is significantly more complex than the Athena equivalent.

The `XmForm` widget provided by Motif, like the Athena `Form` widget, enables the children to specify attachments for the edges of its widgets, but it also enables edges to be positioned to occupy a percentage of the parent window.

The `XmLabel` Widget

The `XmLabel` widget provided by Motif is equivalent to the Athena `Label` widget, differing only in the naming convention of its resources.

The `XmPushButton` Widget

The Motif `XmPushButton` widget, though closely related to the Athena `Command` widget, demonstrates the sophistication of the Motif widget set above Athena.

For instance, like the `Command` widget, the `XmPushButton` widget can contain a `Pixmap` image in lieu of a string; however, it can contain as many as three `Pixmap` images that are displayed automatically by the widget based on the appearance of certain events.

The `XmPushButton` understands that when the right mouse button is clicked, the widget is *armed* and therefore will display the `armedPixmap` if specified by the programmer. This means that a different image will display when the mouse button is clicked and when the mouse button is released. Further, a third image called the `insensitivePixmap` will be displayed when the widget resource `sensitive` is set to `False`.

This discussion is meant only to serve as an introduction to the existence of the Motif widget set and illustrate that it is significantly more sophisticated and therefore more complex than its neighbor Athena.

Because Motif is not freely available, and the Motif clone alternatives are not at a sufficient state to build even novel applications around them, the Graphics Editor will employ the Athena widget set for use with the X Toolkit Intrinsics.

Next Steps

With an understanding of widget concepts, resources, and the Athena class hierarchy, the discussion will advance to a step-by-step approach for creating the necessary elements of an X Window application.

Applying everything discussed thus far in the text, Chapter 6 will demonstrate the required elements for connecting to an X Server, creating widgets, and modifying resources.

Chapter 6

In this chapter

Components of an X Window Application

Having established a sound foundation in preceding chapters, the aim now is to apply everything discussed concerning programming constructs, C language syntax, windowing concepts, X Toolkit Intrinsics, and widget sets to create an X-based application.

This chapter demonstrates the required elements of an X Window application as it leads you through connecting to the X Server and creating and modifying widgets to form a graphical user interface. If you are already acquainted with the components required to program an X Window application, feel free to proceed to the discussion of "Xlib Graphic Primitives" in Chapter 7.

The steps to create an X Window-based application read like a recipe. The following sections describe the code fragments that accomplish each step involved, with final assembly of these fragments at the end of the chapter to create a functional X Window application.

The necessary components in the creation of an X-based application include

1. Connecting to the X Server
2. Creating the application interface
    - Creating buttons
    - Creating pixmap icons
    - Assigning actions
3. Managing windows
4. Processing events

Starting at the beginning, you must establish a connection to the X Server.

Connecting to the X Server

As demonstrated in Figures 5.2 and 5.3 from Chapter 5, "Widget Sets," an X application communicates requests to the X Server through a network connection. This provides the client with absolute separation from the X Server, ensuring its portability and enabling processes to be distributed over a network.

A preliminary step in writing an X application is to establish the lines of communication with the server that will respond to the application's requests.

Several functions are available in the Xt library for establishing a connection to the X Server.

Note

Of course, anything the X Toolkit Intrinsics (Xt) provides functions for can be accomplished by using only the Xlib or basic X library.

However, employing the Intrinsics layer of X for the conveniences it provides is common to software development. For this reason, focus throughout this chapter is on using the X Toolkit and the Athena widget set.

Chapter 7, "Xlib Graphic Primitives," demonstrates when it is unavoidable to employ only the lowest layers of X.

Two of the most common functions used to establish a connection with an X Server are `XtAppInitialize` and `XtVaAppInitialize`.

The functions are closely related except that the first requires that a preformed *resource list* is passed as a parameter and the second enables a *variable argument list*, accounting for any number of resources.

Employing Widget Resources Using Variable Argument Lists

geek speak

A *resource list* is a list of widget resources and their corresponding values. The `Arg` data type is defined in the X Toolkit to support specifying resources and values.

```
typedef struct {
    String name;
    XtArgVal value;
} Arg;
```

The structure of the `Arg` data type contains two fields: one for specifying the resource name and the second for specifying the resource value.

The data type `XtArgVal` differs depending on the architecture where X is installed. It makes the specifications of the value field as generic as possible for portability between platforms of differing architectures.

Generally, an array of `Arg` elements is defined along with a counter for use with the Xt-provided macro `XtSetArg`. This macro is used for filling elements of the array to form the predefined resource list required by functions in the Xt library.

Listing 6.1 illustrates use of the `Arg` structure and the macro `XtSetArg`.

Listing 6.1 `Arg` and `XtSetArg` Usage

```
Arg args[10];
int argCnt;
argCnt = 0;

XtSetArg( args[argCnt], XtNwidth,  220 ); argCnt++;
XtSetArg( args[argCnt], XtNheight, 250 ); argCnt++;

/* args is not a predefined resource list able to
* be passed to functions like XtAppInitialize
*/
```

When reading this code fragment, it is important to note use of the `++` operator on the variable `argCnt`. The `++` operator increments the variable by one and is equivalent to explicitly performing the following statement:

```
argCnt = argCnt + 1;
```

Also important to the use of the increment operator on the variable `argCnt` is that it is performed *external* to the macro `XtSetArg`.

As discussed in Chapter 3, "A Word on C," in the section "The `define` Directive" on page 108, macros operate by substitution by the compiler. Because this substitution can be quite complex, surprising results can occur if you are not careful. Consider the following example:

```
#define SQR(x) x*x
```

It is expected that using `SQR` in a body of code will double any number passed to the macro. For instance,

```
{
    val = SQR(5);
}
```

would be expanded by 5×5 and the value 25 assigned to the variable `val`. However, what would the result of the following be?

```
{
    val = SQR(5+1);
}
```

The expansion of the macro in this example is not as expected. Instead of 5+1×5+1 or 36, which is intended, it would be seen as 5+(1×5)+1 or 11 because the multiplication operator has a higher precedence than the addition operator and is therefore evaluated first.

The macro `SQR` in the previous example is called an *unsafe* macro. To make it *safe* would require use of parentheses to force the intended evaluation, as in

```
#define SQR(x) ((x)*(x))
```

Any macro you did not author should be treated as unsafe, and operators should not be nested in such macros. The result might not be simply an unexpected evaluation of the expression, but potentially incrementing a variable more times than expected. Improper control of variables tends to lead to segmentation violations or other equally unwelcome side effects.

EXCURSION

Understanding Argument Lists of Varying Sizes

A variable argument list is, as the name implies, an argument list of unknown or varying length. You've already witnessed use of variable argument lists in the discussion of the `printf` command in Chapter 3.

The `printf` function expects a varying number of arguments as determined by the number of substitution tokens nested in the format string passed to the function.

Similarly, Xt provides a variety of variable argument functions. These functions are consistently used for specifying resource names and resource value pairs.

As is true with the format string passed to `printf`, functions allowing a variable argument list can have a number of required arguments.

For the discussions in this chapter, if functions exist in the Xt library that accept a variable argument list then they will be used over functions that don't. Variable argument functions eliminate some overhead within the application and provide an elegant programming solution.

Listing 6.2 illustrates use of the `XtVaAppInitialize` function.

Listing 6.2 The `XtVaAppInitialize` Function

```
#include <stdlib.h>
#include <stdio.h>3:  #include <X11/Intrinsic.h> /* for creation routines  */
#include <X11/StringDefs.h>  /* for resource names            */
#include <X11/Xaw/Form.h>    /* to define the formWidgetClass */
{
       XtAppContext appContext;
       Widget toplevel, form;

    toplevel = XtVaAppInitialize( &appContext,
                                  "2D-Editor", /* Application class name   */
                             NULL,0,   /* option list (not used)          */
```

```
                         &argc, argv, /* command line parameters         */
                                NULL,        /* fallback resources (not
                                             used)                       */
                                NULL         /* end of the variable
                                             argument list               */
                              );
```

The function performs several actions that are critical to the execution of the application.

The first parameter passes the address (&) of an `XtAppContext` structure I've called `appContext`. The `XtAppContext` is a structure Xt uses to maintain information associated with the application. Use of the `appContext` variable, once filled by the `XtVaAppInitialize` function, satisfies parameter requirements of subsequent `Xt` function calls.

From the second parameter, `XtVaAppInitialize` reads the class name of the application and queries the server's resource database for any values that may need to be applied to components of the application.

EXCURSION

Tailoring Widgets in an Application

To provide the extensibility required for meeting the varying needs of a large user community, widgets are highly configurable.

Configuring widgets entails changing the value of the resources available to them. As discussed in Chapter 5, resources are introduced by the individual widget and inherited based on their position in the class hierarchy.

A widget's configuration can be *hard-coded* by the programmer, in which case it is absolutely unalterable by users of the application.

Optionally, a widget's configuration can be entirely determined by the user. This requires that the user know some information, namely the instance hierarchy and the class name of the application. Remember that the source code is not always provided for determining this information. A user is usually given clues of the instance hierarchy by the presence of a default resource file provided by the author of the application.

In the absence of a default resource file, a user can employ the X tool `editres` for viewing the structure of Xt-based applications. The tool is useful for finding and setting resources with instant capability to view the results. Further, `editres` is invaluable for creating an application resource file when necessary information is not otherwise provided by the programmer.

The widget configuration (resource file) for an application can exist in several places in the X environment.

Linux commonly places a file corresponding to the class name of the application in the `/usr/X11/app-defaults` directory. Each line of the file contains a widget-specific resource/value pair to specify an element of the desired application configuration.

In the previous example illustrating the use of the `XtVaAppInitialize` function, the arbitrary class name assigned to the application is `2D-Editor`. Therefore, a file of the same name could be placed in the `app-defaults` directory for specifying the widget configuration for the application. Because this file is read and loaded by the `XtVaAppInitialize` function at run-time, the configuration of the resource values is entirely at the discretion of the person who maintains the file.

Optionally, the contents of the file `.Xdefaults` placed in a user's home directory will override entries in the class name files found in the `app-defaults` directory for any X applications.

Finally, the resource values specified on the command line and passed to the `XtVaAppInitialize` through the `argc` and `argv` parameters are applied to the application.

All resources and their values expressed externally to an application (meaning other than the ones hard-coded) are maintained in a database within the X server known as the X Resource Manager Database. To see the values currently contained in the database for your server, use the command

```
bash[40]: xrdb –query
```

The `XtVaAppInitialize` function merges any options specified on the command line with those found in the X Resource Database for this application.

Table 6.1 shows the command-line options that the `XtVaAppInitialize` function understands. All these options are therefore available to any X applications that use `XtVaAppInitialize` to initialize a connection to the X server.

Table 6.1 Standard Xt Command-Line Parameters

Option	*Resource File Syntax*	*Effect*
`-bg` *`colorname`*	`*background:` *`colorname`*	Background color
`-background` *`colorname`*		
`-bd` *`colorname`*	`*borderColor:` *`colorname`*	Border color
`-bordercolor` *`colorname`*		
`-bw` *`number`*	`*borderWidth:` *`number`*	Border width
`-borderwidth` *`number`*		
`-display` *`displayname`*		X Server to use
`-fg` *`colorname`*	`*foreground:` *`colorname`*	Foreground color
`-foreground` *`colorname`*		
`-fn` *`fontname`*	`*font:` *`fontname`*	Font name
`-font` *`fontname`*		
`-geometry` *`string`*	`*geometry:` *`string`*	Size and position
`-iconic`	`*iconic: True`	Starts application iconified
`-name` *`string`*	`*iconName:` *`string`*	Name applied to application's icon

`-reverse`	`*reverseVideo: True`	Reverse video on
`-rv`		
`+reverseVideo`	`*reverseVideo: False`	Reverse video off
`+rv`		
`-synchronous`	`*synchronous: True`	Synchronous mode on
`+synchronous`	`*synchronous: False`	Synchronous mode off
`-title` *`string`*	`*title:` *`string`*	Title applied\Title Bar provided by the window manager

EXCURSION

Uniquely Specifying Widget Paths

The syntax of the resource/value pair eligible for entry into the application's `app-defaults` file or a user's `.Xdefaults` file is in the middle column of Table 6.1.

Notice the wildcard prefacing each of the resource names and the colons separating the names from the resource values. The wildcards are optional syntax but the colons are required.

In lieu of wildcards, the *explicit widget path* referencing the placement of the widget in the instance hierarchy for the application can be used. (This information is obtainable from the `editres` client.)

geek speak

An *explicit widget path* is a period-separated list of the instance names (or optionally class names) of all widgets preceding the widget in the instance hierarchy.

For instance,

```
2D-Editor.Form.Box.Command.foreground: red
```

refers to all `Command` widgets, which are contained by a `Box` widget held on a `Form` widget within the application with the class name `2D-Editor`.

Wildcards and portions of a widget's instance path can be combined as well:

```
*Command.foreground: red
```

The previous command refers to all `Command` widgets within the application.

Finally, as shown in the second column of Table 6.1,

```
*foreground: red
```

affects the foreground color of the entire application, which has the same effect as specifying the command-line parameter:

```
-foreground red
```

The final and most critical action `XtVaAppInitialize` performs is opening a connection with the X Server and creating the `ApplicationShell` widget.

If the function `XtVaAppInitialize` fails to establish a connection with the X Server, it will not create the `ApplicationShell` and returns `NULL` to indicate an error occurred. For this reason, it is important to test the result returned by `XtVaAppInitialize`:

```
toplevel = XtVaAppInitialize( &appContext,  "2D-Editor", NULL,0,
                              &argc, argv, NULL, NULL );
if( toplevel == NULL ) {
    fprintf( stderr, "Failed to connect to X server!" );
    exit( 1 );
}
```

Failing to test for a valid widget returned from `XtVaAppInitialize` results in a program crash in the instance that the connection failed.

How does `XtVaAppInitialize` know with which X server to seek a connection?

The function `XtVaAppInitialize` first considers the presence of the command-line parameter `-display` *`displayname`* for determining with which X server to establish a connection.

If the `-display` parameter is not specified, the presence of an environment variable called `DISPLAY` is consulted. If the variable is set in the environment, its value is used for determining with which server to attempt communication.

Barring the presence of the `DISPLAY` variable in the environment and the `-display` parameter on the command line, a display server on the local host is sought as a last resort.

The syntax for specifying *`displayname`* to the `-display` parameter or `DISPLAY` environment variable follows the syntax

```
machine:server.screen
```

where `machine` can be a valid hostname or IP address followed by a number indicating which `server` on the target machine to connect to. Specifying which `server` is necessary because a single host can run X servers for multiple workstations.

The field `screen` specifies on which monitor (in the case where the X server is supporting multiple monitors for a given display) the application should display.

Often there is only one server running a machine and it is supporting only a single monitor. A typical `displayname` setting is

```
machine:0
```

where the screen number is omitted and `0` refers to the only server present on the machine.

What causes an attempt by `XtVaAppInitialize` to connect with an X server to fail?

The two most common reasons for an attempted connection to fail are either a malformed *displayname* or access being denied, meaning that a server can deny the request because the host attempting the connection is not in the server's *access control list*.

An access control list is managed by the `xhost` command and determines which machines can connect with a given X Server syntax.

Access is given by using the syntax

```
xhost +hostname
```

or denied by

```
xhost -hostname
```

Enabling all connect requests to the server effectively turns off the access control list. Issuing the `xhost` command without any host specified does this.

```
xhost +
```

Similarly, all connect requests will be denied by issuing the command

```
xhost -
```

Having successfully established a connection to the X Server indicated by receiving a valid widget reference returned by `XtVaAppInitialized`, you are ready to continue creating the graphical interface of the application syntax.

Creating the Application Interface

Because one child widget is not sufficient to accomplish a significant graphical user interface, the sole child of the `ApplicationShell` widget is generally a widget from the `Composite` (manager) class such as the `Form` widget.

The `ApplicationShell` widget returned by the call to `XtVaAppInitialize` is the means of tying the application to the window manager. The `ApplicationShell` widget will be given the window manager decorations that enable the application to be resized, moved, and iconified.

Because *shell widgets* are able to parent only one child, the child will be the root of the application's instance tree.

Listing 6.3 demonstrates the `Xt` widget creation routine `XtVaCreateManagedWidget`.

Listing 6.3 The `XtVaCreateManagedWidget` Function

```
form = XtVaCreateManagedWidget( "topForm",          /* instance name */
                                formWidgetClass,    /* widget class  */
                                toplevel,           /* widget parent */
                                NULL );             /* terminate variable
                                                       argument list */
```

You use the function `XtVaCreateManagedWidget` to create a widget of any class. As shown in the previous code snippet, the first argument is the instance name assigned to the widget.

Although *instance names* for widgets in an application need not be unique, it is more difficult to refer to them in a resource file when attempting to use an explicit widget path.

The second parameter to the function `XtVaCreateManagedWidget` is the class name of the widget to create. This variable is defined in the header file included specifically to employ widgets of this class. For instance, refer again to this excerpt from Listing 6.2:

```
#include <X11/Xaw/Form.h>
```

Header files exist to support all widget classes available within the Athena widget set.

The third argument to the function is the parent widget, which in this case is the shell returned from `XtVaAppInitialize`.

The final argument terminates the variable argument list.

Optionally, *hard-coded* resources could be specified for this widget, as shown in Listing 6.4.

Listing 6.4 Hard-Coding Resources

```
18:  form = XtVaCreateManagedWidget( "topForm",          /* instance name */
19:                                  formWidgetClass,    /* widget class  */
20:                                  toplevel,           /* widget parent */
20a:                                 XtNwidth,       220,
20b:                                 XtNheight,      250,
20c:                                 XtNborderWidth, 5,
21:                                  NULL );             /* terminate variable
22:                                                         argument list */
```

Notice the addition of lines 20a–20c. Each line is a combination of a resource name and value. When resource values are specified in a program (hard-coded), changing the value in the code and recompiling is the only way to alter them.

Note

Removing the user's ability to specify resource values for an application is riddled with pros and cons.

The list of cons includes loss of extensibility to suit a wider user base. Having user preferences always honored ensures the best display results on a greater number of platforms.

The pros extend to removing the dependency on user know-how, which can render the application unusable, and minimizing the necessary level of support and documentation to prevent user mistakes.

There is much to say in the way of personal commentary on this issue, but I will limit it to my own rule of what to hard-code and what to leave configurable. Anything I feel strongly about will get hard-coded and I feel strongly about most everything. However, the needs of your users can vary; clearly individual judgement must be applied.

Notice in Listing 6.4, the `NULL` termination is still required to mark the end of the variable argument list.

Note

When specifying widgets internal to an application as shown in Listing 6.4, the naming convention of the resources varies slightly.

Compare the resources used in Listing 6.4 to the resource names specified for the `Core` widget in Table 5.1 of Chapter 5.

When resources are specified external to the program such as through entries in the `app-defaults` or the `.Xdefaults` file, the naming convention used is as seen in Chapter 5. However, internal to the application, you must tell Xt who has named the resource by prefacing the resource name with `XtN`.

Focus a moment on the semantics of the widget creation function name `XtVaCreateManagedWidget`. It is important to discuss the use of the word `Managed` in this context.

➔ As described in Chapter 5, in the section "The Power and Convenience of Using Widget Sets" on page 125 a window is the basic component of X Window programming; a fact that remains true even when employing Xt and a widget set.

geek speak

The term *managing a widget* refers to making the widget eligible for display. The widget will actually become visible when all its ancestors are managed and it is not obscured by other widgets.

The contrast to this is creating a widget using the Xt function `XtVaCreateWidget`, which expects the same parameter list as `XtVaCreateManagedWidget` but will not manage the widget it creates. Instead, Xt must be told explicitly to manage a widget created by `XtVaCreateWidget` through use of the function `XtManageChild`. This is useful for when the widget's appearance in the application is conditional.

For instance, a button to clear the Graphics Editor canvas could be left unmanaged until the user has drawn something and thereby necessitates the clear action.

Continuing in the construction of the application interface, the `Form` widget created by the call to `XtVaCreateManagedWidget` is capable of parenting a virtually indefinite number of child widgets. This enables the assembly of a meaningful graphical user interface.

Begin by adding a place for the drawing of objects in the Graphics Editor application.

Listing 6.5 shows a function employing the `XtVaCreateManageWidget` function to add a *canvas* to the application.

Note

As is discussed in Chapter 7, when employing the drawing functions provided by X, any window will suffice as a *canvas*.

Because all widgets have an associated window, the choice of which widget from the Athena widget class to employ as the drawing canvas is arbitrary.

The Motif widget set, however, provides a widget explicitly for this purpose called the `XmDrawingArea` widget, which has features not available to any widget in the Athena set, as will be demonstrated shortly. Specifically, it will be necessary to inform the Athena `Form` widget to add specific events to its list of events that it monitors.

Listing 6.5 Graphics Editor Canvas Creation

```
void create_canvas( Widget parent )
{
    GxDrawArea = XtVaCreateWidget( "drawingArea",
                                   formWidgetClass, parent,
                                   XtNbackground,
                                     WhitePixelOfScreen(XtScreen(parent)),
                                   XtNtop,            XawChainTop,
                                   XtNleft,           XawChainLeft,
                                   XtNbottom,         XawChainBottom,
                                   XtNright,          XawChainRight,
                                   XtNheight,         220,
                                   XtNwidth,          250,
                                   NULL );
    XtAddEventHandler(
          GxDrawArea,                         /* widget to send events to */
          PointerMotionMask|ButtonPressMask,  /* events to send           */
          False,                              /* non-maskable events      */
          (XtEventHandler)drawAreaEventProc,  /* function to call         */
          (XtPointer)NULL);                   /* data to pass to function */
}
```

The creation of a `Form` widget with a call to `XtVaCreateWidget` is familiar from previous examples; however, in Listing 6.5 a number of resources have been specified in the variable argument list of the function call. In this example, the creation function is setting the background color, several edge *attachments*, and the width and height dimensions. The setting of the size of the `Form` is useful for initial values but will not be retained or enforced by the parent of the `Form` widget (also a `Form`) because the parent's management style is to honor the *relative* placement as indicated by the specified attachments.

Note

The semantics of the `Form` widget edge attachments are perhaps instantly confusing or less than intuitive; however, understanding them is imperative for properly using the `Form`.

Correct use of the `Form` widget ensures that the placement of widgets within an interface is consistently maintained by the application. Other manager widgets will rearrange (pack tightly) the child widgets they maintain, giving a less professional appearance to your interface.

The edge assignments in Listing 6.5 specify that all four edges (top, right, bottom, and left) for the child being created are to be chained (locked) to the corresponding edge of the parent.

```
7:                   XtNtop,              XawChainTop,
8:                   XtNleft,             XawChainLeft,
9:                   XtNbottom,           XawChainBottom,
10:                  XtNright,            XawChainRight,
```

The means of referring to the edge for which an attachment is being specified (`XtNtop`, `XtNleft`, `XtNbottom`, and `XtNright`) is clear from the example.

What varies is the way to specify the attachment for a given edge. An edge can be placed relative to an edge of the parent `Form` using the `XawChain`*`Edge`*, as seen in the previous example where *Edge* specifies what edge of the `Form` the placement is relative to. When chaining to an edge of the parent, the distance separating the child's edge from the parent is maintained when the application is resized, meaning the child's edge will follow the edge of the parent, keeping the relative distance assigned at creation.

When placing a child widget, remember that the resources used to express attachments are provided by the `Constraint` widget. In other words, the child is specifying how it wishes to be *constrained* by the parent. Therefore, the edge specifications refer to the child's edge and the values given for the child's edges are relative the parent.

Optionally, an edge can be relative to another child of the parent by using the `XtNfromHoriz` or `XtNfromVert` resources. These resources expect a reference to an *existing* sibling widget for placing the edge of the widget being created.

When a widget's edge is chained to the edge of a sibling, the relative distance between them is maintained during a resizing of the application.

Finally, an edge can be loosely placed relative to the edge of the parent or sibling using the edge value `XawRubber` where the distance is not maintained, but will vary.

Note

You have not yet seen the specification of *color* with the use of the XtNbackground resource in Listing 6.5.

X provides several macros for specifying black and white colors or Pixel values, two of which are WhitePixelOfScreen and BlackPixelOfScreen. The macros require as a parameter the Screen structure, easily obtained from any existing widget with the macro XtScreen.

Color specification in X beyond black and white can be a tedious task. Therefore, an entire section in Chapter 7 is dedicated to creating and employing colors.

You have seen the creation of the GxDrawArea widget in Listing 6.5 and the widget's placement relative to the parent. Notice that the variable GxDrawArea is not declared in the code listing. As will be demonstrated shortly, the variable is declared globally for use throughout the application.

Continuing with Listing 6.5, focus on the introduction of a new Xt function XtAddEventHandler.

```
    XtAddEventHandler(
            GxDrawArea,                        /* widget to send events to */
            PointerMotionMask|ButtonPressMask, /* events to send           */
            False,                             /* non-maskable events      */
            (XtEventHandler)drawAreaEventProc, /* function to call         */
            (XtPointer)NULL);                  /* data to pass to function */
```

Listing 6.5 uses a call to XtAddEventHandler to register two events PointerMotion and ButtonPress with the GxDrawArea Form widget.

Widgets request only that the X Toolkit notify them of events that satisfy actions inherent to their behavior. PointerMotion and ButtonPress events are not typical for the Form widget because they don't generally respond to user input.

Using the GxDrawArea Form widget as a *canvas* for drawing the objects of our Graphics Editor, however, requires that we know when the user moves the mouse cursor through the canvas or when to click a mouse button to start or end a draw action.

geek speak

The function XtAddEventHandler enables additional events to be added to the notification list of a widget.

The function expects as the first parameter the widget to which the events are to be added. Secondly, the XtAddEventHandler function expects an event mask where each bit represents one or more events. When multiple events are specified, as in this example, a *bitwise ORing* of the event masks to be added to the widget is used.

EXCURSION

Manipulating Variable Values One Bit at a Time

Bit field operations are common in X as an efficient means of combining unique values into a single variable. Elements of bit fields are called *masks* or *flags*.

Effectively, each binary position within the variable represents a unique value. A declaration such as

```
char fields:4;
```

would divide the variables fields into 4 bits. Setting a bit would represent a distinct value in the variable, as in

```
fields = (1<<3); /* sets the value of field to 1000 */
fields = (1<<2); /* sets the value of field to 0100 */
fields = (1<<1); /* sets the value of field to 0010 */
fields = (1<<0); /* sets the value of field to 0001 */
```

The left shift operator (<<) simply says to shift the `1` *n* times to the left (`1<<`*n*).

Using the OR operator (|) enables values to be combined.

```
fields = (1<<3) | (1<<1); /* sets the value of field to 1010 */
```

The values of `PointerMotionMask` and `ButtonPressMask` represent unique positions in the bit field of the second parameter to `XtAddEventHandler` for representing the `PointerMotion` and `ButtonPress` events, respectively.

Table 6.2 shows some of the X-defined event masks and the event types that they select.

Table 6.2 X Event Masks

Event Mask	*Event Type*
`KeyPressMask`	`KeyPress`
`KeyReleaseMask`	`KeyRelease`
`ButtonPressMask`	`ButtonPress`
`OwnerGrabButtonMask`	(none)
`KeymapStateMask`	`KeymapNotify`
`PointerMotionMask`	`MotionNotify`
`ButtonMotionMask`	`MotionNotify`
`Button1MotionMask`	`MotionNotify`
`Button2MotionMask`	`MotionNotify`
`Button3MotionMask`	`MotionNotify`
`Button4MotionMask`	`MotionNotify`
`Button5MotionMask`	`MotionNotify`

continues

Table 6.2 Continued

Event Mask	*Event Type*
`EnterWindowMask`	`EnterNotify`
`LeaveWindowMask`	`LeaveNotify`
`FocusChangeMask`	`FocusIn`
`FocusChangeMask`	`FocusOut`
`ExposureMask`	`Expose`
`ExposureMask`	`GraphicExpose\` `MappingNotify\` `UnmapNotify\`
`DestroyNotify`	

Some event masks in Table 6.2 select only a single event, whereas others select multiple events. Also, some event masks select the same event type. Table 6.2 also provides examples of events that have no corresponding event masks, called *nonmaskable* events.

The third parameter in the call to `XtAddEventHandler` informs Xt whether the event handler should be invoked for *nonmaskable* events. A nonmaskable event cannot be specifically selected because it does not have an associated mask. In Table 6.2, the example passes `False` for this parameter because our event handler is interested only in the events that have been registered.

The fourth parameter, the event handler function, specifies the function that Xt will call when the widget receives the events you've added to the notify list. The actual definition of the `drawAreaEventProc` event handler will appear later in this chapter.

The fifth and final parameter instructs Xt of *client data* to pass to the event handler function when it is invoked.

Note

Client data is a parameter included with every function registered with Xt for invocation.

If there is data to be passed to the function, include it in the client data field and Xt will pass it to your function. If no data is desired, specifying `NULL` will act as a placeholder and no data will be passed to the function.

Having completed the description of creating the `GxDrawArea` widget and registering events that will make it a useful drawing canvas, you are ready to expand the interface to include controls for drawing actions.

Creating Buttons

Creating a place for the user to draw is an important element of the application's interface. However, the buttons used to request a drawing function are equally important.

The first consideration when creating the buttons for invoking drawing actions is where to place them.

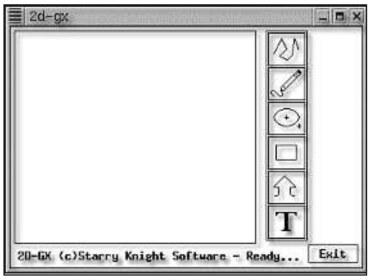
Note

When laying out an application you are authoring from scratch, you should either have a good mental picture for how the application should look or actually doodle a sketch of it.

The mental picture I hold for the Graphics Editor application is shown in Figure 6.1.

Figure 6.1

My mental image of the Graphics Editor layout.



2D-GX (c)Starry Knight Software - Ready...

Supported by the mental image I have of the application layout, a box to contain the drawing buttons will be required. Listing 6.6 shows a function that satisfies this requirement.

Listing 6.6 Button Creation Function

```
#include <X11/Xaw/Command.h> /* for employing the Command widget */
#include <X11/Xaw/Box.h>     /* for use of the Box widget        */

void create_buttons( Widget parent )
{
    Widget butnPanel, exitB;

    /*
     * create a panel for the drawing icons
     */
```

continues

Listing 6.6 Continued

```
    butnPanel = XtVaCreateWidget( "drawButnPanel",
                                 boxWidgetClass, parent,
                                 XtNtop,           XawChainTop,
                                 XtNright,         XawChainRight,
                                 XtNbottom,        XawChainTop,
                                 XtNleft,          XawChainRight,
                                 XtNhorizDistance, 10,
                                 XtNfromHoriz,     GxDrawArea,
                                 XtNhSpace,        1,
                                 XtNvSpace,        1,
                                 NULL );

    create_icons( butnPanel, gxDrawIcons, draw_manager );
    XtManageChild( butnPanel );

    exitB = XtVaCreateManagedWidget( "    Exit    ",
                                   commandWidgetClass, parent,
                                   XtNtop,       XawChainBottom,
                                   XtNbottom,    XawChainBottom,
                                   XtNleft,      XawChainRight,
                                   XtNright,     XawChainRight,
                                   XtNfromVert,  butnPanel,
                                   XtNfromHoriz, GxStatusBar,
                                   NULL );

     XtAddCallback( exitB, XtNcallback, gx_exit, NULL );
}
```

Again, you see the use of the `XtVaCreateWidget`. This time a `Box` widget is instantiated to contain the application's drawing buttons.

The position of the `Box` widget specified by its attachments is for the top and bottom edges of the widget to follow the bottom edge of the parent widget and the right and left edges to maintain a relative distance from the parent's right edge.

This placement ensures that resizing the application by moving the bottom or right resize decorations applied by the window manager will *move* the `Box` widget without *resizing* it because both the top and bottom edges of the widget move in unison, as do the right and left edges.

Both edge pairs (top/bottom and left/right) will retain their relative placement to the parent's edge, preventing the `Box` from resizing, which would ruin the aesthetics of the interface. The only resizing that is desirable is for the canvas (`GxDrawArea`) to grow or shrink with the `ApplicationShell`. Buttons bigger than the icon they hold serve no purpose, whereas a larger canvas means more room for additional drawing.

Before considering the function that actually places `Command` widgets (buttons) into the `Box` widget, consider the creation of the Exit button.

Refer again to the familiar routine for creating a widget in Listing 6.6, lines 26–34. You see some differences to identify; specifically, the Exit button is placed relative to two other widgets.

The use of `XtNfromVert` tells Xt that the top and bottom edge attachments are interpreted relative to the widget specified as the value of the `XtNfromVert` resource or the `butnPanel` `Box` widget. Similarly, the use of `XtNfromHoriz` instructs Xt that the left and right widget edges are relative to a widget called `GxStatusBar`, which you must assume has already been created for it to be used as the value of the `fromHoriz` resource.

Note

The `GxStatusBar` variable refers to a `Label` widget that the application uses to display instructions or statuses to the user. Creation of the widget is reviewed in the final code listing at the end of this chapter. The `GxStatusBar` variable, like `GxDrawArea`, is global because it is used throughout the application.

Comparing Figure 6.1 with the code fragment that creates the Exit button, you will notice that the `Command` widget takes as its default label value the widget's instance name. (If necessary, review `Command` widget resources in Chapter 5, section "The Command Widget," page 139.) In other words, if you do not specifically employ the `XtNlabel` resource assigning a character string value for use in the label field of the `Command` widget, the name assigned to the widget at creation is used.

Line 36 of Listing 6.6 introduces the Xt function `XtAddCallback`.

```
36:     XtAddCallback( exitB, XtNcallback, gx_exit, NULL );
```

The `XtAddCallback` function accepts four parameters. The first specifies the widget to which the callback will be registered. The second parameter instructs Xt the reason for invoking the callback function. The third is the function to invoke for the specified reason. The fourth is the client data to pass when the callback function is invoked.

Referring again to the resources available to the `Command` widget, the callback reason `XtNcallback` occurs when the user activates the widget, or clicks and releases the right mouse button while the mouse cursor is within the window associated with the widget.

We now focus on the function that places the buttons into the `Box` widget created in Listing 6.6.

```
23:    create_icons( butnPanel, gxDrawIcons, draw_manager );
```

The function that creates the buttons for invoking drawing actions expects three parameters: the butnPanel Box widget to hold them, a gxDrawIcons pointer that contains elements necessary for creating the icons, and finally, a callback function to invoke when the button is clicked.

The gxDrawIcons pointer is a pointer to an array of elements of the data type GxIconData, defined as

```
typedef struct _gx_icon_data {
    XbmData *info;
    void (*func)(void);
    char *mesg;
} GxIconData;
```

where XbmData * is a reference to a nested structure that holds the information needed to create the Pixmap icon for each element of the GxIconData.

Two important points here are creating a button to enable the activation of a drawing function and creating an icon to represent the action, (see Figure 6.1).

Creating Pixmap Icons

To create a Pixmap icon, three unique data elements are necessary: the character representation or bits that define the icon, the width of the icon, and the height of the icon. All other parameters to the Pixmap creation function can be common.

An X library call for creating a Pixmap is XcreatePixmapFromBitmapData. It follows the form

```
Pixmap XCreatePixmapFromBitmapData( display,
                                    window,
                                    data bits,
                                    width, height,
                                    foreground color,
                                    background color,
                                    depth );
```

The first parameter, *display*, is a pointer to the structure created when a connection to the X Server was established. It can be obtained from any existing widget using the Xt macro

```
XtDisplay( GxDrawArea )
```

The second parameter is generally the window in which the Pixmap will be placed. However, in our example, the window for the button has not been created (the window associated with the Command widget) so any existing window of the same depth can be used instead.

A macro provided by X enables us to get the root window associated with the display where the application is being displayed, which we are assured will be an existing window; otherwise, the connection request made by `XtVaAppInitialize` would have failed.

```
DefaultRootWindow(XtDisplay(GxDrawArea))
```

Notice that `DefaultRootWindow` expects a pointer to the display structure as its only parameter, which is satisfied with the macro `XtDisplay`.

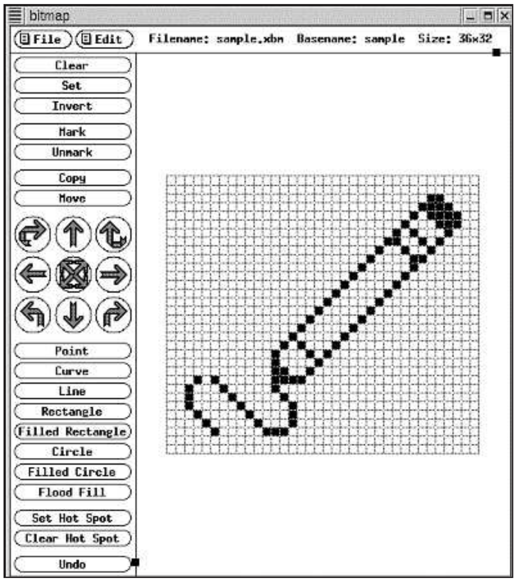
The third parameter passed to the `XCreatePixmapFromBitmapData` is the data bits representing the icon.

X provides a standard client called `bitmap` for creating this data. Other environments can provide superior tools for creating icons. For instance, Sun System's Common Desktop Environment includes a client called `dtpaint`. Similar to this is the `dxpaint` client, which was included with Digital Equipment Corporation's now-defunct Ultrix operating system.

The `bitmap` client is sufficient for creating simple icons if you are a patient individual. Figure 6.2 shows a sample of the bitmap client interface.

Figure 6.2

The `bitmap` *client for creating icons.*



By setting cells within the grid defined to be a specific width and height, the icon is created.

Note

Notice that the `bitmap` client does not enable specifying colors for the icon image's cells. This is because by definition a `bitmap` has a depth of 1. It is a map of *bit*s, as suggested by its name. A single bit can only represent on or off.

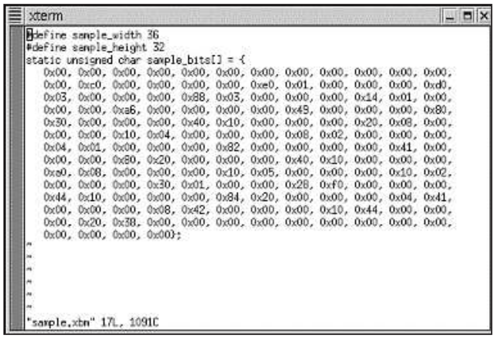
More complex icon images can be created using clients that support the creation of a `Pixmap` directly. Because a `Pixmap` has a depth greater than 1, it is able to represent colors.

A way to understand this is to think of a monochrome monitor where the pixels of the monitor could only represent an on or off condition. Either the pixel is lit or it is black. The monochrome monitor in this context would have a depth of only 1.

Color monitors, however, are comprised of planes of bitmaps where a single cell is the alignment of the planes to express a more complex value than simply on or off.

Using this expression of color representation, the rule for determining the number of colors able to be presented on a monitor is understood. The formula is 2^{depth} (2 raised to the depth of the screen) because the screen's depth represents the number of bit planes and each bit plane is capable of representing 2 values (on and off).

Figure 6.3 shows the output of the file created when the icon in Figure 6.2 is saved.



```
#define sample_width 36
#define sample_height 32
static unsigned char sample_bits[] = {
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0xc0, 0x00, 0x00, 0x00, 0x00, 0xe0, 0x01, 0x00, 0x00, 0x00, 0xd0,
   0x03, 0x00, 0x00, 0x00, 0x88, 0x03, 0x00, 0x00, 0x00, 0x14, 0x01, 0x00,
   0x00, 0x00, 0xa6, 0x00, 0x00, 0x00, 0x00, 0x49, 0x00, 0x00, 0x00, 0x80,
   0x30, 0x00, 0x00, 0x00, 0x40, 0x10, 0x00, 0x00, 0x00, 0x20, 0x08, 0x00,
   0x00, 0x00, 0x10, 0x04, 0x00, 0x00, 0x00, 0x08, 0x02, 0x00, 0x00, 0x00,
   0x04, 0x01, 0x00, 0x00, 0x00, 0x82, 0x00, 0x00, 0x00, 0x00, 0x41, 0x00,
   0x00, 0x00, 0x80, 0x20, 0x00, 0x00, 0x00, 0x40, 0x10, 0x00, 0x00, 0x00,
   0xa0, 0x08, 0x00, 0x00, 0x00, 0x10, 0x05, 0x00, 0x00, 0x00, 0x10, 0x02,
   0x00, 0x00, 0x00, 0x30, 0x01, 0x00, 0x00, 0x28, 0xf0, 0x00, 0x00, 0x00,
   0x44, 0x10, 0x00, 0x00, 0x00, 0x84, 0x20, 0x00, 0x00, 0x00, 0x04, 0x41,
   0x00, 0x00, 0x00, 0x08, 0x42, 0x00, 0x00, 0x00, 0x10, 0x44, 0x00, 0x00,
   0x00, 0x20, 0x38, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00};
~
~
~
~
~
~
"sample.xbm" 17L, 1091C
```

Figure 6.3

`sample.xbm` file contents.

Notice that the file contains the required fields for defining a unique icon.

The `XbmData` data type provides a structure for storing these unique fields and satisfying the requirements for creating an icon within the Graphics Editor application.

```
typedef  struct _xbm_data {
    unsigned char *bits;
    int           w, h;
} XbmData;
```

Listing 6.7 shows the definition of the bitmap data used to represent the various drawing icons.

Listing 6.7 Icon Definitions

```
#define icon_static( name, bits, width, height ) \
static XbmData name = { bits, width, height }
/*
 * drawing icons
 */
static unsigned char line_bits[] = {
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x20,0x00,0x04,0x00,0x00,0x30,0x00,
  0x0c,0x00,0x00,0x50,0x00,0x0c,0x00,0x00,0x48,0x00,0x14,
  0x00,0x00,0x88,0x00,0x14,0x00,0x00,0x84,0x00,0x14,0x00,
  0x00,0x04,0x01,0x22,0x00,0x00,0x02,0x01,0x22,0x00,0x00,
  0x02,0x02,0x22,0x00,0x00,0x01,0x02,0x42,0x00,0x00,0x01,
  0x04,0x42,0x00,0x80,0x00,0x04,0x42,0x00,0x80,0x00,0x02,
  0x01,0x00,0x40,0x00,0x01,0x01,0x00,0x40,0x80,0x00,0x01,
  0x00,0x20,0x40,0x00,0x01,0x00,0x20,0x20,0x00,0x01,0x00,
  0x00,0x10,0x00,0x01,0x00,0x00,0x08,0x80,0x00,0x00,0x00,
  0x30,0x40,0x00,0x00,0x00,0xc0,0x20,0x00,0x00,0x00,0x00,
  0x13,0x00,0x00,0x00,0x00,0x0c,0x00,0x00,0x00,0x00,0x00,
  0x0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0x00};
icon_static( line_icon, line_bits, 36, 32 );

static unsigned char pencil_bits[] = {
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
  0x00,0xc0,0x00,0x00,0x00,0x00,0xe0,0x01,0x00,0x00,0x00,0xd0,
  0x03,0x00,0x00,0x00,0x88,0x03,0x00,0x00,0x00,0x14,0x01,0x00,
  0x00,0x00,0xa6,0x00,0x00,0x00,0x00,0x49,0x00,0x00,0x00,0x80,
  0x30,0x00,0x00,0x00,0x40,0x10,0x00,0x00,0x00,0x20,0x08,0x00,
  0x00,0x00,0x10,0x04,0x00,0x00,0x00,0x08,0x02,0x00,0x00,0x00,
  0x04,0x01,0x00,0x00,0x00,0x82,0x00,0x00,0x00,0x00,0x41,0x00,
  0x00,0x00,0x80,0x20,0x00,0x00,0x00,0x40,0x10,0x00,0x00,0x00,
  0xa0,0x08,0x00,0x00,0x00,0x10,0x05,0x00,0x00,0x00,0x10,0x02,
  0x00,0x00,0x00,0x30,0x01,0x00,0x00,0x28,0xf0,0x00,0x00,0x00,
  0x44,0x10,0x00,0x00,0x00,0x84,0x20,0x00,0x00,0x00,0x04,0x41,
  0x00,0x00,0x00,0x08,0x42,0x00,0x00,0x00,0x10,0x44,0x00,0x00,
  0x00,0x20,0x38,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00};
icon_static( pen_icon, pencil_bits, 36, 32 );

static unsigned char arc_bits[] = {
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0x3f,0x00,0x00,0x00,
  0x70,0xc0,0x01,0x00,0x00,0x0c,0x00,0x06,0x00,0x00,0x02,0x00,
  0x08,0x00,0x00,0x01,0x00,0x10,0x00,0x80,0x00,0x00,0x20,0x00,
```

continues

Listing 6.7 Continued

```
  0x40,0x00,0x00,0x40,0x00,0x40,0x00,0x04,0x40,0x00,0x20,0x00,
  0x04,0x80,0x00,0x20,0x00,0x1f,0x80,0x00,0x20,0x00,0x04,0x80,
  0x00,0x40,0x00,0x04,0x40,0x00,0x40,0x00,0x00,0x40,0x00,0x80,
  0x00,0x00,0x20,0x00,0x00,0x01,0x00,0x10,0x00,0x00,0x02,0x00,
  0x08,0x00,0x00,0x0c,0x00,0x86,0x00,0x00,0x70,0xc0,0x81,0x00,
  0x00,0x80,0x3f,0xe0,0x03,0x00,0x00,0x00,0x80,0x00,0x00,0x00,
  0x00,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00};
icon_static( arc_icon, arc_bits, 36, 32 );

static unsigned char box_bits[] = {
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,0x1f,
  0x00,0x80,0x00,0x00,0x10,0x00,0x80,0x00,0x00,0x10,0x00,
  0x80,0x00,0x00,0x10,0x00,0x80,0x00,0x00,0x10,0x00,0x80,
  0x00,0x00,0x10,0x00,0x80,0x00,0x00,0x10,0x00,0x80,0x00,
  0x00,0x10,0x00,0x80,0x00,0x00,0x10,0x00,0x80,0x00,0x00,
  0x10,0x00,0x80,0x00,0x00,0x10,0x00,0x80,0x00,0x00,0x10,
  0x00,0x80,0x00,0x00,0x10,0x00,0x80,0x00,0x00,0x10,0x00,
  0x80,0x00,0x00,0x10,0x00,0x80,0xff,0xff,0x1f,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0x00};
icon_static( box_icon, box_bits, 36, 32 );

static unsigned char arrow_bits[] = {
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x00,
  0x00,0x00,0x80,0x02,0x00,0x00,0x00,0x40,0x04,0x00,0x00,0x00,
  0x20,0x08,0x00,0x00,0x00,0x10,0x10,0x00,0x00,0x00,0x08,0x20,
  0x00,0x00,0x00,0x04,0x40,0x00,0x00,0x00,0x02,0x80,0x00,0x00,
  0x00,0x01,0x00,0x01,0x00,0x80,0x00,0x00,0x02,0x00,0x40,0x00,
  0x00,0x04,0x00,0x20,0x00,0x00,0x08,0x00,0xe0,0x07,0xc0,0x0f,
  0x00,0x00,0x04,0x40,0x00,0x00,0x00,0x04,0x40,0x00,0x00,0x00,
  0x04,0x40,0x00,0x00,0x00,0x04,0x40,0x00,0x00,0x00,0x04,0x40,
  0x00,0x00,0x00,0x04,0x40,0x00,0x00,0x00,0x04,0x40,0x00,0x00,
  0x00,0x02,0x80,0x00,0x00,0xc0,0x01,0x00,0x07,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00};
icon_static( arr_icon, arrow_bits, 36, 32 );

static unsigned char text_bits[] = {
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
  0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,
  0x1f,0x00,0x80,0x83,0x1f,0x1c,0x00,0x80,0x01,0x0f,0x18,
  0x00,0x80,0x00,0x0f,0x10,0x00,0x00,0x00,0x0f,0x00,0x00,
  0x00,0x00,0x0f,0x00,0x00,0x00,0x00,0x0f,0x00,0x00,0x00,
   0x00,0x0f,0x00,0x00,0x00,0x00,0x0f,0x00,0x00,0x00,0x00,
   0x0f,0x00,0x00,0x00,0x00,0x0f,0x00,0x00,0x00,0x00,0x0f,
   0x00,0x00,0x00,0x00,0x0f,0x00,0x00,0x00,0x00,0x0f,0x00,
   0x00,0x00,0x00,0x0f,0x00,0x00,0x00,0x00,0x0f,0x00,0x00,
   0x00,0x00,0x0f,0x00,0x00,0x00,0x00,0x0f,0x00,0x00,0x00,
   0x80,0x1f,0x00,0x00,0x00,0xe0,0x7f,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00};
     icon_static( text_icon, text_bits, 36, 32 );
```

The bulk of Listing 6.7 is the definition of the

```
static unsigned char bits[] = {
```

which defines each of the icons used to represent the `draw` functions.

EXCURSION

Understanding Bit-Mapped Data

The characters in the bit data represent the cells of the bitmap grouped into 16-bit entities and converted to hexadecimal representation using Binary Converted Decimal (BCD) notation, where groupings of 8 bits are converted at a time.

Consider the following example of an 8×2 bitmap where a 1 indicates that the cell is set (on).

```
0    0    0    0    0    0    0    0
0    0    0    0    1    1    1    1
```

The hexadecimal representation of this would be

```
0x        0

0    0    0    0    0    0    0    0
0    0    0    0    1    1    1    1

0x        4
```

or

```
unsigned char bits[] = {0x04};
```

Focus for a moment on the following macro defined in Listing 6.7:

```
#define icon_static( name, bits, width, height ) \
static XbmData name = { bits, width, height }
```

With every call to the macro a new variable is declared.

The first field passed to the macro (`name`) is substituted on the second line of the macro to be a variable of type `XbmData`.

The remaining fields of the macro are then used to fill the `XbmData` structure fields.

For instance,

```
109: icon_static( text_icon, text_bits, 36, 32 );
```

is equivalent to

```
XbmData text_icon = { text_bits, 36, 32 };
```

Therefore, Listing 6.7 declares and fills the following `XbmData` structures:

```
line_icon
pen_icon
arc_icon
box_icon
arr_icon
text_icon
```

These variables are capable of satisfying the first field of the `GxIconData` structure defined earlier.

```
typedef struct _gx_icon_data {
    XbmData *info;
    void (*func)(void);
    char *mesg;
} GxIconData;
```

Listing 6.8 shows the use of the `XbmData` variables to define the `gxDrawIcons` array needed for creating the buttons that will fill the Button box of Listing 6.6.

Listing 6.8 `gxDrawIcons` Array Declaration

```

GxIconData gxDrawIcons[] = {
{ &line_icon, gx_line,   "Draw an elastic line..."       },
{ &pen_icon,  gx_pencil, "Draw a freestyle line..."      },
{ &arc_icon,  gx_arc,    "Draw a circle..."              },
{ &box_icon,  gx_box,    "Draw a square or rectangle..." },
{ &arr_icon,  gx_arrow,  "Draw an arrow..."              },
{ &text_icon, gx_text,   "Draw dynamic text..."          },
/*------------------------------------*/
/* this list MUST be NULL terminated */
/*------------------------------------*/
{ NULL },
};
```

Looking closely at the definition of the `gxDrawIcons` array, you see that the array elements hold a `GxIconData` structure. Further, there are seven array elements defined, with the last one being set to `NULL`.

The first six elements, however, are set as the value of the `info` field, a reference (`&`) to a `XbmData` structure, which satisfies the type requirement of `XbmData *`.

Assigning Actions

The elements then each assign as the value of the `func` field a function appropriate to the action implied by the icon. Shortly, I'll introduce the header file where these functions are prototyped and therefore eligible for referencing.

Note

A new syntax is introduced with the `func` field of the `GxIconData` structure.

```
void (*func)(void);
```

A function is much like a variable in that it is a reference to a value. What differs is the manner in which the value is evaluated. Clearly, with a function, the compiler will go to the point of entry for the function and begin evaluating its contents.

This does not preclude a program from referencing the entry point (function name) as it would any other variable.

The syntax of the `func` field in the `GxIconData` structure supports this because it defines a function pointer `(*func)` that has no return type and accepts no parameters. The program is then able to assign function entry points to this variable as it does with the second field of the `gxDrawIcons` array element's initialization.

The third and final field being set in the initialization of each of the array elements is the `GxIconData mesg` field. The `mesg` field is a character pointer, which is used in conjunction with the `GxStatusBar` described earlier for providing statuses and messages to the user.

Having understood the definition of the structures that form the `gxDrawIcons` array passed as the second parameter to the function `create_icons` in Listing 6.6, we are now able to introduce that function.

It was invoked on line 23 of Listing 6.6

```
23:  create_icons( butnPanel, gxDrawIcons, draw_manager );
```

and its definition appears in Listing 6.9.

Listing 6.9 `create_icons` **Function Definition**

```
void create_icons( Widget parent, GxIconData *iconData,
                   void (*callback)( Widget, XtPointer, XtPointer ))
{
```

continues

Listing 6.9 Continued

```
   Widget btn;
   Pixmap pix;

   while( iconData->info != NULL ) {
        if( iconData->info->bits != NULL ) {
            pix = create_pixmap( parent, iconData->info );

            btn = XtVaCreateManagedWidget( "",
                                       commandWidgetClass, parent,
                                       XtNwidth,  iconData->info->w + 1,
                                       XtNheight, iconData->info->h + 1,
                                       XtNbackgroundPixmap,     pix,
                                       XtNhighlightThickness,   1,
                                       NULL );

            XtAddEventHandler( btn, EnterWindowMask, False,
                              (XtEventHandler)statusProc,
                              (XtPointer)iconData->mesg);
            XtAddEventHandler( btn, LeaveWindowMask, False,
                             (XtEventHandler)statusProc, (XtPointer)NULL );

            XtAddCallback( btn, XtNcallback,
                           callback, (XtPointer)iconData->func );
        }
       /*
        * go to the next element
        */
        iconData++;
    }
}
```

Listing 6.9 presents a weighty example, reviewing everything we've discussed so far.

The function `create_icons`, as discussed previously, expects three parameters. The sample of Listing 6.6 satisfied them by passing the `Box` widget as the first parameter, the `gxDrawIcons` array as the second, and the function `draw_manager` as the third.

Note

Although I've not introduced the function `draw_manager`, you should be comfortable with referring to functions as variables in the sense that they hold an entry point for interpretation.

The `create_icons` function parses the `iconData` array (passed as `gxDrawIcons`), looping until it discovers the `NULL` termination included in the array definition.

For each element found in the `iconData` array, a `Pixmap` is created based on the contents of `info` field stored in the array element. (Remember that the `info` field corresponds to the bit data, width, and height information for each of the icons created by the `bitmap` client.)

The `create_pixmap` function is shown in Listing 6.10.

Listing 6.10 `create_pixmap` Function Definition

```
Pixmap create_pixmap( Widget w, XbmData *data )
{
    return( XCreatePixmapFromBitmapData( XtDisplay(w),
                                         DefaultRootWindow(XtDisplay(w)),
                                        (char *)data->bits,
                                         data->w, data->h,
                                         BlackPixelOfScreen(XtScreen(w)),
                                         WhitePixelOfScreen(XtScreen(w)),
                                         DefaultDepthOfScreen(XtScreen(w))));
}
```

Note

Use of the `XCreatePixmapFromBitmapData` was demonstrated earlier. Unique to this use, however, is that the specifics of the Graphics Editor application satisfy the parameter requirements. Specifically, note the use of the `XbmData *` for providing the icon-specific data required for creating this icon.

With a valid `Pixmap` named `pix` created from the `info` field of the `iconData` array, the `create_icons` function creates a `Command` widget setting the value of the resource `XtNbackgroundPixmap` to the `Pixmap pix`.

When the resource `XtNbackgroundPixmap` is employed for a `Label` or `Command` widget, the `XtNlabel` resource is ignored.

Note

For this reason, no instance name is assigned to the buttons created by `create_icons`.

Before incrementing the array to reference the next element, seek the `NULL` termination with the following statement:

```
29:     iconData++;
```

The `create_icons` function installs two event handlers and a callback function to the newly created `Command` widget.

EXCURSION

Referencing Array Elements with Pointer Arithmetic

The ++ operator was described earlier as incrementing a variable by 1. How, then, does it work to advance an array to the next element?

When the variable being advanced is a pointer (flagging again the dangers of pointer manipulation in C), the variable is advanced by the size of the *thing* to which it points.

In the example of `iconData++`, the statement is equivalent to

```
    iconData = iconData + sizeof(iconData[0]);
```

The address to which `iconData` points is moved forward the distance in memory equal to the size of one element of the array.

The first element (0) is used because 1 is the minimum length of a valid array.

The first event handler monitors for the cursor entering the window associated with the widget. When the `EnterNotify` event occurs, the event handler `statusProc` function will be invoked and passed as client data to the `mesg` field of the `iconData` array.

The purpose of `statusProc` is to update the `GxStatusBar Label` widget with the message string passed. In the case of the `LeaveNotify` event where the client data is `NULL`, the content of the `GxStatusBar` is cleared.

The callback assigned to the `Command` widgets created by the `create_icons` routine is the same for all, namely, the `draw_manager` function passed to the function pointer `callback`.

Evaluating the declaration of the function pointer `callback` provides insight into how to declare a `XtCallback` function.

```
 2:                    void (*callback)( Widget, XtPointer, XtPointer ))
```

The function has no return type and expects three parameters.

The first parameter will be the widget for which the callback is invoked.

The second parameter is of the data type `XtPointer` and is the client data specified as the last parameter to the `XtAddCallback` function.

The actual type of the third parameter, though passed as an `XtPointer`, is determined by the *reason* the callback was invoked.

For instance, the `Command` widget has a unique data structure for providing information relating to a `ButtonPress` event and another for the information pertinent to the `DestroyNotify` event. These structures are always passed by reference to the third parameter of a callback function.

Note

The data type `XtPointer` is defined in the X environment to be something similar to

```
typedef void *XtPointer.
```

because anything can legally (syntactically speaking) be referenced as a pointer to `void`.

It is necessary to cast the variable defined as `XtPointer` to a more appropriate data type before employing it.

Listing 6.11 shows the `draw_manager` function passed to the `create_icons` function as the function pointer `callback`.

Listing 6.11 `draw_manager` and `drawAreaEventProc` Function Definitions

```
void draw_manager( Widget w, XtPointer cd, XtPointer cbs )
{
    void (*draw_func)( XEvent * ) = (void (*)(XEvent *))cd;

    if( draw_func != NULL ) (*draw_func)( NULL );
        draw_mgr_func = draw_func;
    }

void drawAreaEventProc( Widget w,
                        XtPointer cd, XEvent *event, Boolean flag )
{
    if( draw_mgr_func != NULL ) (*draw_mgr_func)( event );
}
```

These functions will grow in complexity as more features and capabilities are added to the Graphics Editor application.

The `draw_manager` function demonstrates that when a `Command` widget is activated, the `draw` function is specified in the `iconData` array (the second field of the `gxDrawIcons` array) and is assigned as client data to the `draw_manager` callback function.

```
      XtAddCallback( btn, XtNcallback,
                     callback, (XtPointer)iconData->func );
```

The previous command is cast from being an `XtPointer` to a function pointer and then invoked, thus enabling the draw action.

The `draw_manager` function then assigns the `draw_func` function pointer to a global variable `draw_mgr_func`, which is visible to the `drawEventProc`. When the `drawEventProc` event handler is invoked due to a `PointerMotion` or `ButtonPress` event occurring in the window of the canvas, it is able to invoke the last `draw_func` registered by the `draw_manager` callback function. The purpose of the `drawAreaEventProc` function is to continue processing the draw action instigated by the user when he activated a draw button icon.

More details will be added in future examples, but you should be able to follow the program flow to this point.

Having postponed until now the creation of the `GxStatusBar Label` widget and definition of the function `statusProc`, consider Listing 6.12.

Listing 6.12 `GxStatusBar` Creation and Management

```
1:  #include <X11/Xaw/Label.h>
2:  /*
3:   * create_status
4:   */
5:  void create_status( Widget parent, Widget fvert )
6:  {
7:    GxStatusBar = XtVaCreateManagedWidget( "statusBar",
8:                                           labelWidgetClass, parent,
9:                                           XtNtop,         XawChainBottom,
10:                                          XtNleft,        XawChainLeft,
11:                                          XtNbottom,      XawChainBottom,
12:                                          XtNright,       XawChainRight,
13:                                          XtNfromVert,    fvert,
14:                                          XtNborderWidth, 0,
15:                                          NULL );
16:   setStatus( "2D-GX (c)Starry Knight Software - Ready..." );
17: }
18: /*
19:  * setStatus
20   */
21: void setStatus( char *message )
22: {
23:     XtVaSetValues( GxStatusBar, XtNlabel, message, NULL );
24: }
25: /*
25a: * statusProc
26:  */
27: void statusProc( Widget w, XtPointer msg, XEvent *xe, Boolean *flag )
28: {
29:     if( msg == NULL )
30:         setStatus( "\0" );
31:     else
32:         setStatus( msg );
33: }
```

Listing 6.12 introduces three functions:

```
5:  void create_status( Widget parent, Widget fvert )
```

which creates the `GxStatusBar`.

```
21: void setStatus( char *message )
```

which employs the Xt function `XtVaSetValues` for changing the resource values of widgets that have already been created. In the `setStatus` routine, the `XtVaSetValues` is used to alter the `XtNlabel` resource value of the `GxStatusBar` to be the status message passed to the function.

Finally, Listing 6.12 shows the definition of the event handler.

```
void statusProc( Widget w, XtPointer msg, XEvent *xe, Boolean *flag )
```

Like `XtCallback` functions, event handlers are invoked by Xt and therefore must have a predictable parameter list. Effectively, the functions you define as callbacks or event handlers will honor the parameter list expected by Xt.

Because we've already reviewed the parameter list associated with the `XtCallback`, consider now the event handler `statusProc`.

The first parameter is the widget in which the event occurred.

The second parameter is the client data specified to the function call `XtAddEventHandler` that registered the event handler.

The third argument is the actual `XEvent` structure definition for the event that triggered the invocation of the event handler.

Finally is a pointer to a Boolean variable that the event handler could set to `False` in order to prevent this event from being dispatched to other widgets that might have registered to receive it. Setting this flag to `False` is not recommended.

Having completed the steps of creating the application interface, adding buttons with `Pixmap` icons, and assigning actions to widgets, you are ready to *realize* the `ApplicationShell` to have it displayed on the screen and, lastly, process events.

Managing Windows

You have seen that all widgets have an associated window. A window follows rules for being displayed that are identical to displaying widgets. Specifically, for a window to be displayed, all its ancestors must be displayed, and it must not be obscured by another window.

In widget terms, the widget must be managed, as must its ancestors, and the windows associated with the widgets must be realized or *produced*. The act of creating a widget does not necessarily create its associated window. Xt must be instructed to do so for all widgets in the application.

A function exists within Xt for the purpose of realizing all the windows associated with the widgets parented by the `ApplicationShell` returned by the call to `XtVaAppInitialize`.

The function is `XtRealizeWidget`, which takes as its only parameter the `ApplicationShell` widget.

```
XtRealizeWidget(toplevel);
```

It is acceptable to create widgets after the call to `XtRealizeWidget`, which is usually done when you don't know in advance when the widget will be needed.

The act of realizing all the widget's windows also causes them to be *mapped*. Mapping a window means to make it eligible for display, dependent, of course, on the rules of display provided earlier.

With the windows realized and mapped, the final step is to process the events sent to the application by the X server.

Processing Events

The function for processing events in an application is `XtAppMainLoop`. It is invoked by specifying the `XtAppContext` variable filled in by the call to `XtVaAppInitialize` as illustrated in the following.

```
XtAppMainLoop( appContext );
```

The `XtAppMainLoop` function contains an infinite loop that intentionally never returns.

Invoking the function `XtAppMainLoop` transfers control of the application to Xt.

From this point on, Xt will be responsible for sending events to the widgets that have requested them. The widgets, in turn, process the events by invoking the callback functions, event handlers, or the widget's internal methods.

Only through dispatching events and responding to the generated requests will Xt communicate with the application.

Summary

A good deal of material was introduced in this chapter. For clarity and brevity, the code samples provided are not entirely complete. A point that should be evident to you is that many of the widget creation examples are contained in functions that were never invoked. Other details have been intentionally omitted to eliminate redundancy.

Appendix B, "Application Layout Code Listing," provides a complete code listing for laying out this phase of the Graphics Editor Project. Building the code in Appendix B will yield the image found in Figure 6.1.

The listing is arranged in logical functions contained within appropriate source code files. Further, the creation routines are linked to the function `main`, making this phase of the project functional. Issues such as function prototypes have been addressed as well, with the hope of making the code a solid example of good programming practice.

You are encouraged to dwell on the code listing. In doing so, you will notice that there are some slight differences (improvements) to what was introduced in this chapter.

Next Steps

The next chapter introduces the only time the lowest level of X (Xlib) *must* be invoked. The X Toolkit Intrinsics provide simpler methods for accomplishing everything X is capable of except for performing X graphic primitives.

As introduced in Chapter 7, Xlib graphic primitives are the basic drawing functions provided by X.

Chapter 7

In this chapter

Xlib Graphic Primitives

The thorough discussion of programming with the X Toolkit Intrinsics in Chapter 6, "Components of an X Window Application," omitted one essential element necessary for approaching the Graphics Editor project. The missing piece is not addressed by any of the functions provided with Xt.

To perform the basic graphic functions of drawing lines, arcs, points, and rectangles, the X library, `Xlib`, must be used. These basic functions, along with others for copying and for creating the graphic contexts needed to define the attributes of the operations, are known as *X Graphic Primitives*.

A *graphic primitive* is an Xlib call that sends a request to the X Server to draw a specific shape at a specific place in a window.

What's important in this statement is that graphic primitives are relative windows and only available from Xlib.

For efficiency, Xlib functions for performing graphic primitives do not include much information. Instead, attributes of the draw request such as foreground and background color, fill styles, line widths, textures, and more, are stored in a Graphics Context structure known as the `GC`.

The Graphics Context

A Graphics Context (`GC`) structure reference is included in the parameter list of *every* Xlib graphic primitive function call. It is as important to the request as the window in which the operation will be performed.

A Graphics Context can be created using either Xlib or Xt and serves the purpose of defining all the attributes of the draw request.

Focusing on the Xlib call for creating a `GC`, consider the following function prototype:

```
GC XCreateGC( Display *, Window, XtGCMask, XGCValues )
```

Note

The Xt function for creating a `GC` is very similar to the Xlib call except that the Xt call ensures that `GC`s are shared within the application.

Depending on the role of the application and the number of `GC`s being created, this may be important, as there is a certain amount of overhead associated with the `GC`. Creating too many `GC`s in an application may degrade the program's performance. What qualifies as *too many*, of course, is system dependent.

Because the Graphics Editor application employs only one GC at a time and the attributes of that GC vary with every draw request, it is better suited to use the Xlib call for creating GCs.

The `XCreateGC` function that Xlib provides creates a unique `GC` for defining a draw request's attributes.

The first parameter is a pointer to the `Display` structure created by the call to `XtVaAppInitialize` (or equivalent). The `Display` pointer can be obtained from any existing widget using the macro

```
XtDisplay( widget )
```

The second parameter refers to a window. Because every widget has a window, the macro

```
XtWindow( widget )
```

can be used to obtain a widget's associated window.

Warning

A word of caution…

When specifying a window value for the creation of a `GC`, the `GC` returned by the creation can only be used on that window or on windows of the same depth.

The caveat only applies to display servers using multiple monitors, because it is impossible to create windows of varying depth on the same screen.

The third parameter is the *value mask* of the fields you've set in the `XGCValues` structure reference passed as the fourth and last parameter.

EXCURSION

Streamlining Field References by the Use of a Value Mask

The use of a *value mask* in X is very common. As shown in Chapter 6, a *mask* is a method of using a single variable to represent multiple values by setting specific bits within bit fields. (See Chapter 6, in the section "Creating the Application Interface," page 151.)

For the third parameter of the `XCreateGC` function, the mask informs the X Server which fields to use from the `XGCValues` structure and which can be ignored.

Therefore, for fields in which an assignment is made in the `XGCValues` structure, the corresponding value mask must be ORed (|) together and passed as the value mask to the creation routine. Values not set in the `XGCValues` structure will be set to their default value.

Listing 7.1 shows the definition of the `XGCValues` structure and the corresponding value mask to signify when value is set as indicated in the comment adjacent to each field.

Listing 7.1 `XGCValues` Structure

```
typedef struct {                       /* value mask         */
  int function;                       /* GCFunction        */
  unsigned long plane_mask;           /* GCPlaneMask       */
  unsigned long foreground;           /* GCForeground      */
  unsigned long background;           /* GCBackground      */
  int line_width;                     /* GCLineWidth       */
  int line_style;                     /* GCLineStyle       */
  int cap_style;                      /* GCCapStyle        */
  int join_style;                     /* GCJoinStyle       */
  int fill_style;                     /* GCFillStyle       */
  int fill_rule;                      /* GCFillRule        */
  int arc_mode;                       /* GCArcMode         */
  Pixmap tile;                        /* GCTile            */
  Pixmap stipple;                     /* GCStipple         */
  int ts_x_origin;                    /* GCTileStipXOrigin */
  int ts_y_origin;                    /* GCTileStipYOrigin */
  Font font;                          /* GCFont            */
  int subwindow_mode;                 /* GCSubwindowMode   */
  Bool graphics_exposures;            /* GCGraphicExposures */
  int clip_x_origin;                  /* GCClipXOrigin     */
  int clip_y_origin;                  /* GCClipYOrigin     */
  Pixmap clip_mask;                   /* GCClipMask        */
  int dash_offset;                    /* GCDashOffset      */
  char dashes;                        /* GCDasheList       */
} XGCValues;
```

In the following sections, the fields of the `XGCValues` structure used in the Graphics Editor will be discussed with illustrations of valid field assignments.

The GC Function

The XGCValues function field has several possible values defined by Xlib. This field specifies how the draw request is accomplished, or how the pixels are drawn in the destination window.

Note

Although I keep using the term *window* when referring to graphic primitive functions, the correct term is *drawable*.

geek speak

A *drawable* is either a Window or a Pixmap.

The utility of drawing in a Window should be immediately evident; however, equally important is the ability to perform graphic operations against a Pixmap.

Because Pixmaps differ from Windows only in that they aren't capable of requesting and receiving events, they are frequently used as off-screen buffers for graphic operations. Anything drawn in an off-screen Pixmap cannot be affected by Expose events or other Windows obscuring their contents, as is the case with Windows.

Therefore, the contents of the Pixmap can be transferred to a Window in response to the Window receiving an Expose event indicating that a graphic refresh is necessary. This illustrates that both Windows and Pixmaps qualify as a Drawable, which is the required second parameter of Xlib graphic primitive function calls.

Table 7.1 shows the possible XGCValues function field values and a description of the type of operation each performs.

Note

All the GC functions are specified in terms of the value of the source foreground color and the destination pixel value.

In other words, if GXand is specified as the GC function, a request to draw a line sets the pixels (points) in the source covered by the line to the foreground color of the GC. A Boolean AND operation is then applied between the pixels in the source and the current value of the corresponding pixels in the destination to determine the new state of the destination pixels.

Table 7.1 GC Function Types

Function	*Description*
GXclear	Clears destination pixels where corresponding pixels in the source are set.
GXand	If source pixels and destination pixels are set, the destination pixels remain set.

Function	Description
`GXandReverse`	If source pixels are set and not the destination pixels then set destination pixels.
`GXcopy`	Any pixels set in the source will be set in the destination.
`GXandInvert`	Pixels not set in the source and set in the destination result in destination pixels being set.
`GXnoop`	Destination pixels left unchanged.
`GXxor`	A Boolean `EXCLUSIVE OR` operation is performed between source and destination pixels. This function is used to create a rubber banding effect.
`GXor`	Source pixels or destination pixels result in the destination pixels being set.
`GXnor`	If the source pixels are not set and the destination pixels are not set, the pixels will be set in the destination.
`GXequiv`	The source pixels are inverted and `EXCLUSIVE OR`ed with the destination pixels to determine the state of the destination pixels.
`GXinvert`	The destination pixels are inverted.
`GXorReverse`	The source pixels or the inverted destination pixels result in pixels being set in the destination.
`GXcopyInverted`	The source pixels are inverted and copied to the destination.
`GXorInverted`	The source pixels are inverted and `OR`ed with the destination pixels.
`GXnand`	If the source pixels are not set or the destination pixels are not set, the destination pixels will be set.
`GXset`	All corresponding pixels from the source are set in the destination.

The default value for the `GC` function is `GXcopy`. To create a `GC` using a value other than the default would require the declaration and manipulation of a `XGCValues` structure as shown in the following code fragment:

```
{
    GC gc;

    XGCValues values;
    XtGCMask  value_mask = 0; /* clear the mask */

    values.function = GXxor; /* to peform rubber banding */
    value_mask |= GCFunction;

    gc = XCreateGC( XtDisplay(GxDrawArea),  /* pointer to Display           */
                    XtWindow(GxDrawArea),   /* window reference             */
                    value_mask,             /* indicate fields set in values */
                    &values );              /* value structure              */
}
```

Note

The assignment to `value_mask` in this previous example uses a syntax not yet introduced.

```
    value_mask |= GCFunction;
```

Combining an operator such as `OR` (`|`) with the assignment operator (`=`) is common in C syntax. These combined operations are semantically equivalent to

```
    value_mask = value_mask | GCFunction;
```

Because the `GCFunction` mask is the only flag assigned (`OR`ed) to the `value_mask` variable, the X Server will create a unique `GC` using default values for all fields of the `XGCValues` structure except the `function` field, which is set to the `GXxor` function.

The next elements of the `XGCValues` structure important to the Graphics Editor project are the `foreground` and `background` fields.

`GCForeground` and `GCBackground`

Creating and managing colors in X should be an independent science, or at least require a license in wizardry.

To specify a color in X, a pixel value is used. The pixel value is effectively (simply stated) an index into a *colormap*.

geek speak

A *colormap* is an array of color cells actively being used by the X Server. When a color is no longer referenced by any X Client, the color cell in the colormap is made available for allocation by other clients.

Figure 7.1 shows a sample colormap obtained using the Linux X client `gimp`.

Figure 7.1
Sample colormap.

The size of the colormap array, as alluded to in the Note following Figure 6.2 in Chapter 6, in the section “Creating the Application Interface,” page 151, equates to the number of colors available to the display as calculated by the function

`max colors =` $2^{\text{depth of screen}}$.

Xlib provides several functions for allocating a color cell in a colormap. Color allocation can be performed by specifying a color *name* or an RGB (red, green, and blue) color component.

EXCURSION

A Closer Look at Color Management

A closer look at color management in the XA display server automatically creates a default colormap for use by all X clients it serves. However, based on the requirements of the client, there might not be a sufficient number of color cells available in the shared colormap. When this happens, an application is free to create a private colormap.

geek speak

Only one *colormap* can be active in a server at a time.

Window focus then determines which colormap is active in the server at any given moment. A client with a private colormap gaining focus will have its colormap *installed* by the server.

This causes other applications running on the desktop to display in colors that might not be aesthetically pleasing or even make sense, because the pixel values referenced by a client using the default colormap are then interpreted relative to the private colormap.

For instance, the default colormap shown in Figure 7.1 has at index 0 the color red. An application creating a private colormap can allocate in its first cell the color black.

When the private colormap is active in the server, all applications employing the pixel value of 0 will display black because that is the color corresponding to index 0 in the *active* colormap.

Returning focus to a client using the default colormap returns the default colormap as active in the server and index 0 again refers to red.

Performing color allocation by specifying *named* colors is *generally* the best way to ensure that your X application obtains a valid color value.

Note

The caveat *generally* is applied because the internal methods for color allocation are server dependent.

If all the color cells in the default colormap are filled by the applications using it, the method of satisfying subsequent allocation requests or performing a closest match algorithm differs from server to server.

The Xlib function for allocating a color by specifying a color name is `XAllocNamedColor`.

```
int XAllocNamedColor(display, colormap, color name, exact_color, closest_color)
```

The function expects as its first parameter a pointer to the `Display` structure created when a connection to the X Server was established.

The second parameter refers to the colormap where the color will be searched for to determine whether it has already been loaded. If it is not found in the colormap, and if there are empty color cells available for allocation, the color will be loaded.

The third parameter is the name of the color to be allocated.

Note

Color names are not arbitrary. A file called `rgb.txt` exists on every system for specifying the named colors available to the X Server. The file also contains the RGB values that are used to create the color when it is requested.

The location of the `rgb.txt` file varies between systems, but is found in the directory `/usr/lib/X11` under the Linux operating system.

The last two parameters required by `XAllocNamedColor` are references to `XColor` structures, which the function will fill to indicate the *exact* match for the color requested and the *closest* match.

The `XColor` structure is defined as

```
typedef struct {
 unsigned long pixel;        /* index into colormap where color resides       */
 unsigned short red, green, blue; /* RGB components                           */
 char flags; /* specifies which components are set when allocating by RGB value */
 char pad;   /* used to satisfy some compiler alignment requirements          */
} XColor;
```

The function returns a non-zero value to indicate that the named color was found in the `rgb.txt` file, and zero to indicate that it wasn't.

The X Server cannot allocate a *named color* not found in the `rgb.txt` file because the RGB components will be unknown.

If the color is found in the `rgb.txt` file, the *`exact_color`* `XColor` structure filled by `XAllocNamedColor` will hold the RGB values found in the file.

The *`closest_color`* `XColor` structure is the one safely employed by the X application.

Consider the following code fragment for demonstrating the allocation and use of a named color:

```
{
    XColor closest, exact;
    int status;

     GC gc;

    XGCValues values;
    XtGCMask  value_mask = 0; /* clear the mask */

    status = XAllocNamedColor( XtDisplay(GxDrawArea),
                  DefaultColormap(XtDisplay(GxDrawArea),
                              DefaultScreen(XtDisplay(GxDrawArea))),
                                              "lightblue",
             &exact, &closest );
    if( status == 0 ) {
        printf( "failed to alloc color" );
        return;  /* nothing more can be done */
   }
    values.foreground = closest.pixel; /* index where match was found/created */
    values_mask = GCForeground;
    gc = XCreateGC( XtDisplay(GxDrawArea),  /* pointer to Display            */
                    XtWindow(GxDrawArea),   /* window reference              */
                    value_mask,             /* indicate fields set in values */
                   &values );               /* value structure               */
}
```

Notice in the previous example that the default colormap can be obtained using the macro

`DefaultColormap(XtDisplay(` *`widget`* `), DefaultScreen(XtDisplay(` *`widget`* `)))`

which expects two parameters, the `Display` pointer and a screen number obtained with the macro `DefaultScreen`.

Also, note that the `pixel` field of the *`closest`* `XColor` structure is the valid reference to the color allocated by the function call.

The next field from the `XGCValues` structure to consider is the attribute for setting the line width of a `draw` function.

GCLineWidth

The simplest `XGCValues` field to discuss is the `line_width` field.

As implied by the name, this field sets the width of lines drawn by the graphic primitive. If the value of `line_width` is 0 (the default), the fastest method known by the server for drawing a line of a pixel width of 1 is used.

The following example illustrates setting the line width for a graphics context:

```
{
    GC gc;

    XGCValues values;
    XtGCMask  value_mask = 0; /* clear the mask */
    values.line_width = 3;
    value_mask |= GCLineWidth;

    gc = XCreateGC( XtDisplay(GxDrawArea),  /* pointer to Display  */
                    XtWindow(GxDrawArea),   /* window reference    */
                    value_mask,             /* indicate fields set in values */
                    &values );              /* value structure     */
}
```

The final `XGCValues` field used in the Graphics Editor project is the `tile` field.

GCTile

The tile field of the XGCValues structure assigns a Pixmap for use by the GC function as the source Drawable for graphic operations.

When a background pixmap of the destination Drawable is used as the tile value, the operation effectively performs an erase.

The effect of tiling is to use the bits set in the tile pixmap that correspond to a specified graphic function for placement in the destination.

Because this is the method of removing objects from the canvas in the Graphics Editor application, the subject will be visited again in more detail in Chapter 16, "Object Manipulation," in the section "Deleting an Object" page 321.

A complete discussion of the Graphics Context value structure involves much more than the elements used in the Graphics Editor. Table 7.2 shows a summary of the fields and their allowable values for reference.

Table 7.2 XGCValues Field Summary

Field	*Values*	*Description*
function	see Table 7.1	Logical operation for performing graphic primitive requests
plane_mask	unsigned long	Specifies in which planes the primitive should be performed
foreground	Pixel	Index into the colormap
background	Pixel	Index into the colormap
line_width	int	Thickness of lines
line_style	LineSolid, LineOnOffDash, LineDoubleDash	Style of lines
cap_style	CapNotLast, CapButt, CapRound, CapProjecting	Manner in which the corners of line and rectangle primitives are drawn
join_style	JoinMiter, JoinRound, JoinBevel	Method of joining two lines
fill_style	FillSolid, FillTiled, FillStippled, FillOpaqueStippled	Method of performing fill primitive request
fill_rule	EvenOddRule, WindingRule	Rule for performing fill primitive request when the points of object being drawn are not ordered
arc_mode	ArcChord, ArcPieSlice	Describes whether the end points of an arc connect to each other (ArcChord) or to the center of the arc that defines them (ArcPieSlice) .

Field	Values	Description
tile	Pixmap	Pixmap for tiling operations
stipple	Pixmap (depth of 1)	Applies texture to primitive requests
ts_x_origin		
tx_y_origin	int	Offset of tile or stipple pixmap in the destination Drawable
font	Font	Specifies the font to use in XDrawString primitive requests
subwindow_mode	ClipByChildren, IncludeInferiors	Controls whether subwindows obscure their parent
graphic _exposures	True, False	Specifies whether an Expose event should be generated by the primitive request
clip_x_origin		
clip_y_origin	int	Sets offset of clip region in the destination Drawable
clip_mask	Pixmap	Applies alternative clipping Drawable for limiting effect of primitive requests
dash_offset	int	Offset of dashes field in the destination Drawable
dashes	char	Pattern to apply to dashed lines

A number of *convenience routines* exist in Xlib for altering the values of a `GC` after its creation. Some of these routines include `XSetForeground`, `XSetBackground`, and `XSetFunction`. They will be introduced in the context of the Graphics Editor application in later chapters.

Knowing how to specify the values desired for a graphics context and the Xlib method of creating them, let us shift our focus to the functions that employ `GC`s.

Graphic Primitive Functions

The graphic primitives provided by X for requesting the drawing of geometric shapes (using the attributes specified in the Graphics Context) all operate in an integer pixel coordinate system with the origin at the upper-left corner of the `Drawable`.

No transformations are performed by X beyond the clipping performed at the edge of the `Drawable`, and optionally requested by the `clip_mask` being defined in the `GC`.

The smallest entity that can be drawn using an X graphic primitive is a *point*.

XDrawPoint

The Xlib graphic primitive for drawing a single point to the screen is `XDrawPoint`.

The first three parameters the `XDrawPoint` function expects should be familiar to you already.

They are

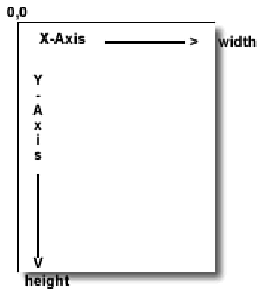
```
XDrawPoint(display, drawable, gc, x, y)
```

a pointer to the `Display` structure, the destination `Drawable`, and a Graphics Context.

Unique to the `XDrawPoint` are the final two parameters used to specify a coordinate. Remember that the origin of a coordinate system is the upper-left corner of the `Drawable` with the X-Axis increasing to the right and the Y-Axis increasing downward as demonstrated in Figure 7.2.

Figure 7.2

Pixel coordinate system.



The `XDrawPoint` graphic primitive draws a *single* point at the specified x, y location using the foreground color of the Graphics Context.

To draw multiple points, an array of `XPoint` structures is defined and passed to the function `XDrawPoints`.

An `XPoint` structure is defined as

```
typedef struct {
     short     x, y;
   } Xpoint;
```

which is simply used to hold a single (x, y) coordinate.

The function `XDrawPoints` expects, in conjunction with the array of `XPoint` structures, an integer specifying the length of the array and a *mode* for interpreting the points as seen in the following:

```
XDrawPoints(display, drawable, gc, points, npoints, mode)
```

The value of the `mode` parameter can be either `CoordModeOrigin` or `CoordModePrevious`.

If `CoordModeOrigin` is used, each point of the array is interpreted as relative to the origin or the `Drawable`. `CoordModePrevious` instructs `XDrawPoints` to treat each consecutive point in the array as a *distance* from the previous point. This means that the first point is placed at the specified (x, y) relative to the upper-left corner of the window and consecutive (x, y) values are used as a distance from the previous point.

The graphic primitive for connecting two points to form a line is `XDrawLine`.

XDrawLine

The `XDrawLine` function, like all other graphic primitives, expects as the first three parameters the `Display` pointer, `Drawable`, and Graphics Context to be used for the operation.

In addition to these parameters, the `XDrawLine` function needs the endpoints that define the line to be drawn.

```
XDrawLine(display, drawable, gc, x1, y1, x2, y2)
```

The `XDrawLine` function uses the foreground of the `GC` to draw a line connecting the two points specified.

To draw a *polyline* (multiple-line segments), X provides the graphic primitive `XDrawLines`.

The `XDrawLines` function parameter list is identical to the `XDrawPoints` function; however, the `XDrawLines` function will connect the points specified in the points array.

```
XDrawLines(display, drawable, gc, points, num points, mode)
```

To request that predefined primitives be drawn, X provides the functions `XDrawRectangle` and `XDrawArc`.

XDrawRectangle

The Xlib graphic primitive `XDrawRectangle` has as its unique parameters (parameters beyond the `Display` pointer, `Drawable`, and `GC`) an x, y location specifying the upper-left corner of the object and the width and height to apply to the rectangle being drawn.

```
XDrawRectangle(Display, drawable, gc, x, y, width, height)
```

As with other graphic primitives, multiple rectangles can be drawn with a single request by creating an array of `XRectangle` structures and employing the function `XDrawRectangles`.

```
XDrawRectangles(display, drawable, gc, rectangles, num rectangles)
```

XDrawArc

The `XDrawArc` function works by specifying a *rectangle* to hold the arc and setting a start and stop angle.

```
XDrawArc(display,drawable,gc, x, y, width, height, angle1, angle2)
```

The rectangle defined is not actually drawn by the request. Instead, it is used to position the arc (x, y) and determine the arc's aspect ratio (proportion of width to height).

The *`width`* value is translated by the `XDrawArc` request as the x-axis diameter of the arc and the `height` value is tsranslated as the y-axis diameter. This enables the drawing of ellipses (width and height values are not equal) as well as circles (values are equal).

The angles passed to `XDrawArc` instruct the function of the starting and ending points of the arc extent. Because the unit of measure X expects angles expressed in is (degrees/64), a full 360-degree arc is requested by setting the value of *`angle1`* to 0×64 and *`angle2`* to 360×64.

Note

Because the value 360 is meaningful to programmers, the angle is generally nested in the `XDrawArc` function as (360×64).

If you actually pull out a calculator and determine that 360×64 equals 23,040 and use this value in the call to `XDrawArc`, those following behind you must reverse the calculation to understand that the intent was for 360×64.

If an arc less than 360 degrees is desired, two decisions must be made: Where will the arc begin and where will it end?

To express the starting point of an arc, *`angle1`* determines (in degrees×64) the distance from the 3 o'clock position of the bounding rectangle `XDrawArc` should begin drawing.

The extent of the arc (end point) is expressed by *`angle2`* as (degrees×64) relative to the starting angle.

Figure 7.3 illustrates the use of *`angle1`* and *`angle2`* to form a draw request using `XDrawArc`.

As shown in Figure 7.3 *`angle1`* determines the start point of the arc and *`angle2`* the distance from *`angle1`*.

Creating arcs and specifying start and end angles will be reviewed again in the context of the Graphics Editor project with the introduction to the `gxArc` object.

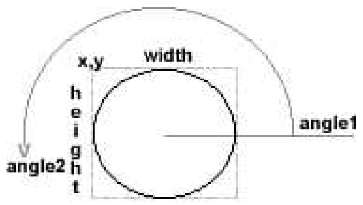


Figure 7.3

`XDrawArc` *parameters.*

The last thing to consider with the introduction of Xlib graphic primitives is the manner in which draw requests are formed for filled objects.

Filled Graphics

For nearly every graphic *`XDraw`* function introduced in this chapter there is a corresponding *`XFill`*, which accepts an identical parameter list but produces a *filled* graphic object.

Table 7.3 shows the match of *`XDraw`* to *`XFill`* functions provided by Xlib.

Table 7.3 Xlib Graphic Primitive Fill Functions

Draw Function	*Fill Function*
`XDrawPoint`, `XDrawPoints`	(none)
`XDrawLine`, `XDrawLines`	`XFillPolygon` (see exception 1)
`XDrawRectangle`, `XDrawRectangles`	`XFillRectangle`, `XFillRectangles`
`XDrawArc`, `XDrawArcs`	`XFillArc`, `XFillArcs`

The exception noted in Table 7.3 refers to the fact that the `XFillPolygon` parameter list does not exactly mirror the `XDrawLines` function.

The prototype for `XFillPolygon` is

```
XFillPolygon(display, drawable, gc, points, npoints, shape, mode)
```

where the added *`shape`* parameter is one of the following:

- **`Complex`**—The polygon can have any *shape* and the path defined by the points can intersect. In other words, the polygon overlaps itself similar to a *bowtie*.
- **`Nonconvex`**—The polygon cans be *concave* and the paths defined by the points do not intersect (overlap) .
- **`Convex`**—The polygon must be convex. This is the least general case, but it is generally the most efficient for the X server to render.

Next Steps

This chapter provided a whirlwind introduction to the Xlib graphic primitives that will be used in later chapters to accomplish the drawing of objects supported by the Graphics Editor application.

Specific examples and more detail are provided when the functions are introduced in the context of authoring the application.

The next important step before employing the graphic primitives within the editor application is to review bitmap versus vector graphics.

The X Window System is a bitmapped-based graphic system. The Graphics Editor, however, is vector based. Reviewing both methods makes it clear how they cans co-exist between the application and the windowing environment.

Part III

Back to School

Chapter 8

In this chapter

- *Vector Graphics*
- *Raster Graphics*
- *Next Steps*

Vector Versus Raster Graphics

Concepts of both raster- and vector-based graphics are important for authoring the Graphics Editor Project and employing the X Window System.

This chapter provides a description of each, showing their similarities, differences, and importance to window and graphics programming.

The X Window System, as introduced over the past few chapters, expects every graphics request to include all the details necessary for performing the task. The required level of detail is obtained by X requiring that the `Window` and `GC` structures be passed to every Xlib Graphic Primitive function call.

The information contained in the graphics request is sent through a process known as the *graphics pipeline*.

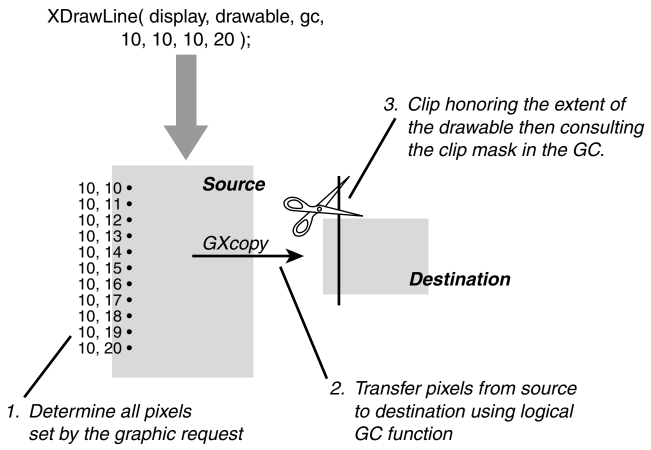
The graphics pipeline determines which pixels in the destination `Drawable` will be set based on the request made and its associated parameters. Elements of a request that influence the determination include the nature of the primitive (line, arc, rectangle, and so forth) and attributes such as fill, stipple, clip mask, line width, line style, foreground, background, and so on.

The X server evaluates the request to determine the state of the pixels in the source, then based on the logical function specified in the `GC`, the source is *applied* to the destination. Finally, clipping is performed based on the dimensions of the destination and any clip region specified in the `GC`.

Figure 8.1 demonstrates the graphics pipeline process for drawing a line.

Figure 8.1

Tracing the X Graphics Pipeline for drawing a line.


XDrawLine(display, drawable, gc,
10, 10, 10, 20);
10, 10 •
10, 11 •
10, 12 •
10, 13 •
10, 14 •
10, 15 •
10, 16 •
10, 17 •
10, 18 •
10, 19 •
10, 20 •
Source
GXcopy
Destination
3. Clip honoring the extent of the drawable then consulting the clip mask in the GC.
2. Transfer pixels from source to destination using logical GC function
1. Determine all pixels set by the graphic request

The request shown in Figure 8.1 is to connect the points between (10,10) and (10,20). The first step in the graphics pipeline is to determine all the source pixels that should be *set* based on the request. This phase of the process incorporates elements of the `GC`, such as the stipple, dashes, line style, and line width. If the line width were 4, for instance, four adjacent pixels would be affected by the request. Similarly, if a dashed line were requested, not all of the consecutive pixels between the points would be set, but only those representing the dash pattern specified.

The actual value of the pixels determined to be set depends on the foreground, background, and possibly tile values specified in the `GC`.

The next step of the pipeline is to apply the source pixels to the destination using the logical function specified in the `GC`.

Finally, any necessary clipping is performed either to bring the graphics request to the bounds of the `Drawable` destination or to honor any clip mask specified within the `GC`.

The purpose of applying the graphics pipeline in response to a graphics request is to determine the final state of the pixels in the destination. These pixel values are used as indexes into a *color look up table* (CLUT) to determine the values needed to drive the monitor (CRT).

The description of Figure 8.2 and the X Server's response to the graphics request leads to the two following concepts:

- Initial requests made of the X server using the Xlib Graphic Primitives covered in Chapter 7, "Xlib Graphic Primitives," are *vector based*.
- The graphics pipeline, however, converts the results of the Xlib primitive functions into a raster (bitmapped) image that can then be mapped to the screen.

Vector Graphics

Vector-based graphics employ a process of creating images using functions based on mathematical statements that place shapes precisely within a given space of the window.

In physics, a *vector* is the representation of both a quantity and a direction.

Functions that create an image in vector-based graphics employ a series of vector specifications such as the Xlib primitive `XDrawLines` function, which specifies a number of vertices to be connected.

A *vertex* (singular of vertices) is a point that marks the junction of two line segments.

Like popular graphic programs such as Adobe Illustrator, the Graphics Editor is entirely vector based, meaning the graphic objects created and altered by the program will consist of *geometric descriptions*.

These descriptions will be retained in terms of origins, vertices, angles, dimensions (width and height), and bounds as is pertinent to a specific shape. In fact, geometric descriptions specific to the shapes supported by the Graphics Editor form the graphic object definitions.

The benefit of maintaining graphics as *vectors* is the flexibility of manipulating them with minimal data loss.

Vector-based images can be scaled, rotated, and transposed without greatly affecting the quality of the image. If you have ever resized a bitmapped image or font, you might have noticed the deterioration in the image quality as the vertices (not explicitly defined) do not continue to connect, giving the image a jagged appearance. Vector-based images, however, retain their original quality during such transformations. Further, vector-based graphics require that less data be stored to represent them because geometric descriptions are all that is required to reproduce them.

Chapters 9–11 review the mathematics necessary for altering vector graphic descriptions to accomplish various forms of object manipulation.

As stated earlier, at some point a vector-based image must be converted into a raster image for mapping to the display.

Raster Graphics

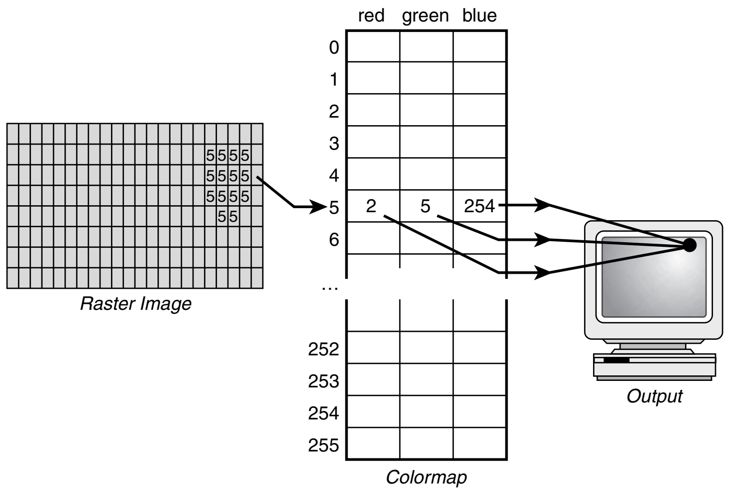
Raster graphics, often called bitmapped graphics, are images described by pixel values that can be directly mapped to the screen.

A *raster* is a grid of x and y coordinates describing either an entire display or some portion of it. A raster-based image identifies which of these coordinates to illuminate based on the pixel value stored in the corresponding cell of the grid (see Figure 8.2).

The amount of information required to represent a raster image is significantly greater than that required to represent a vector image because every cell within the raster image must specify a pixel value for the image mapping.

Figure 8.2

Example of a Raster image.



Modifying a raster image is difficult, slow, and generally results in loss of information that affects the quality of the image. Modifications to raster images are applied to every pixel value comprising the entire image.

Contrast this with modifying vector-based images, which affects only the pixels that connect the geometric description of the objects.

Reversing the alteration of a vector-based image results in an image very near the original, whereas raster images are degraded to some extent.

Note

Clearly, processors and software packages exist that are dedicated to the task of manipulating raster images and ensuring their integrity is retained.

The only thing more astounding than the computing power required for the task is the cost associated with such platforms.

Next Steps

It is crucial to understand the relationship between the vector-based Xlib calls and the bitmapped display nature of the X Window System before proceeding to the next chapter.

Chapter 9, “Object Bounds Checking,” focuses on the best means for determining the bounds of an object based on the graphic descriptions associated with different geometric shapes.

Because an object’s bounds are used later in the Graphics Editor project to determine whether an X `ButtonPress` event has selected the graphic (made it active for manipulation), correctly calculating the occurrence of an event within an object’s bounds is necessary for installing proper object control.

Chapter 9

In this chapter

Object Bounds Checking

Every object of the Graphics Editor application has an associated size, position, width, and height. This chapter discusses methods of using the descriptive information available to the editor objects for determining whether an X `ButtonPress` Event occurred within the bounds of the object. This test is necessary to support the function of *selection* on the objects, making them eligible for manipulation and insuring proper object control.

The list of objects created by the Graphics Editor includes `LatexLine`, `PolyLine`, `Box`, `Arrow`, `Arc`, and `Text`.

Calculating the *bounds* of an object is dependent on the geometric shape represented by the object.

The bounds of an object refer to its dimensions and relative placement within the canvas window.

Because all events that occur within the window used for drawing are relative to the window's origin, the placement and extents of an object must be viewed similarly. Therefore, the calculation of an object's bounds allows events to be compared to objects to determine whether the event occurred within or on the object.

A structure useful for representing the bounds of an object is shown in Listing 9.1.

Listing 9.1 Bounds Structure

```
typedef struct _bounds {
  int x;
  int y;
  Dimension width;
  Dimension height;
} Bounds;
```

Listing 9.1 defines the data type used in the Graphics Editor project for representing the bounds of the object.

Note

The data type `Dimension` used for the `width` and `height` fields of the `Bounds` structure defined in Listing 9.1 is defined in the X Window environment as

```
typedef unsigned int Dimension;
```

Because the `LatexLine`, `PolyLine`, `Box`, `Arrow`, and `Text` objects are all represented with similar internal structures, namely as a series of `XPoints` contained in an array, the method of calculating the bounds of these objects will be the same. Taking into account the `Arc` object, two methods are required for calculating object bounds. One method serves point-array–based objects and one method is explicit to the `Arc` object.

Point-Array–Based Object Bounds

Calculating the bounds of the editor objects represented as an array of points is as simple as assigning the (x,y) origin of an object into the Bounds structure and calculating the minimum and maximum points contained in the array defining the object. This is shown in the code fragment of Listing 9.2.

Listing 9.2 Line Object Bounds Calculation

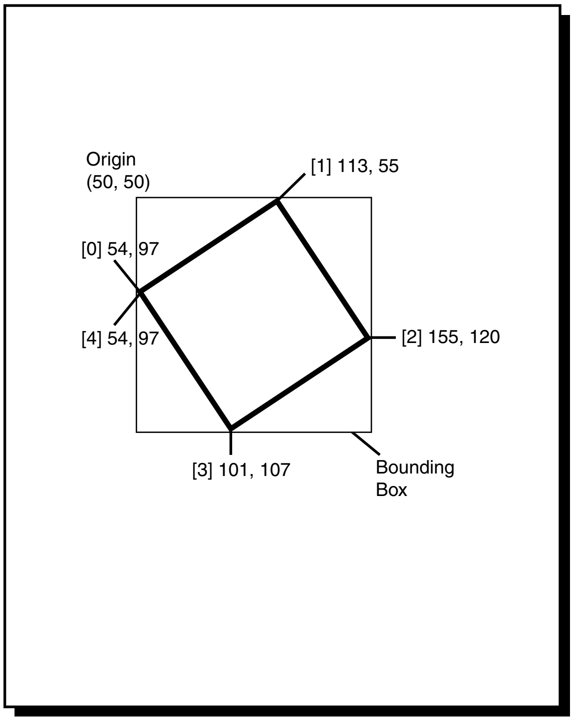
```
// defintion of GXLine Object
typedef struct _gx_line {

  int num_pts; // length of points array
  XPoint pts[]; // array to hold points defining object
} GXLine, *GXLinePtr;

// macros for determining minimum and maximum values
#define GXMIN(a, b) ((a) < (b) ? (a) : (b))
#define GXMAX(a, b) ((a) < (b) ? (b) : (a))

// sample of parsing GXLine points array
Bounds *getLineBounds( GXLinePtr line )
{
  static Bounds bounds; // static makes the variable
                        // persistent after the function
                        // returns
  int i, min_x = 999, min_y = 999, max_x = -999, max_y = -999;

  bounds.x = line.x
  for( i = 0; i < line->num_pts; i++ ) {
    min_x = GXMIN( min_x, line->pts[i].x );
    min_y = GXMIN( min_y, line->pts[i].y );
    max_x = GXMAX( max_x, line->pts[i].x );
    max_y = GXMAX( max_y, line->pts[i].y );
  }
  bounds.width  = (max_x - min_x);
  bounds.height = (max_y - min_y);

  return &bounds; // return a pointer to the structure
}
```

Listing 9.2 shows the definition of the `GXLine` structure used to represent the data specific to the point-array–based objects used in the Graphics Editor.

The difference of the minimum and maximum points representing the line object determines the dimensions of the object as illustrated in Figure 9.1.

Figure 9.1

A demonstration of Line Object Bounds.



The points defined for the line object shown in Figure 9.1 can be defined as

```
XPoint pts[6] = {{54, 97}, {113, 55}, {155, 120}, {101, 107}, {54, 97}};
```

There are five points in the array accounting for the last line that closes the object.

The maximum x point found is at element two (155) and the smallest is at the first and last elements (54). The largest y point is at element three (107) and the smallest is at element two (55) .

Applying the width and height calculations seen in Listing 9.2 reveals

```
width  = 155 - 54 = 99;
height = 107 - 55 = 52;
```

At the return of the function `getLineBounds`, the bounds structure for this object is

```
  bounds.x = 50;
  bounds.y = 50;
  bounds.width  = 99;
  bounds.height = 52;
```

After illustrating the calculation for determining the bounds of an arc object, I will show you how to use the bounds of an object to accomplish object selection.

Arc Object Bounds

The definition of the arc-specific object data, as shown in Listing 9.3, implicitly describes the bounds of the object. Because of this, the function for calculating the arc's bounds is significantly simpler than seen with the line object.

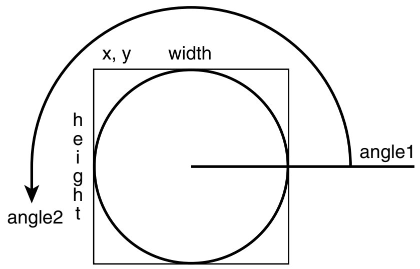
Listing 9.3 Arc Object Bounds Calculation

```
typedef struct _gx _arc {
  int x, y;
  Dimension width, height;
  int angle1, angle2;

} GXArc, *GXArcPtr;

void getArcBounds( GXArcPtr arc )
{
  static Bounds bounds;

  bounds.x = arc->x;
  bounds.y = arc->y;

  bounds.width  = arc->width;
  bounds.height = arc->height;
}
```

Recall the definition of the `XArc` for representing arcs in X, as demonstrated in Figure 9.2.

Figure 9.2

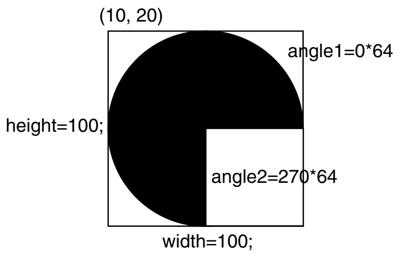
XArc defintion.



Consider the example of the arc defined in Figure 9.3.

Figure 9.3

Arc Object Bounds.



The example in Figure 9.3 produces an `XArc` defined as

```
arc.x = 10;
arc.y = 20;
arc.width  = 100;
arc.height = 100;
arc.angle  = 0;
arc.angle  = 270*64;
```

The data defining the arc object directly transfers to define the bounds of the object.

```
bounds.x = 10;
bounds.y = 20;
bounds.width  = 100;
bounds.height = 100;
```

Now that we have bounds definitions for sample objects of a line and an arc type, let us see how to employ them.

Employing Object Bounds

Consider the case where a user right-clicks in the drawing window to process a `ButtonPress` event for the location x,y as shown in Figure 9.4.

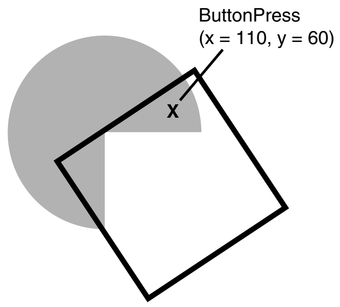


Figure 9.4

`ButtonPress` event for object selection.

By comparing the bounds calculated for each of the objects in the drawing window to the `ButtonPress` event location, an object can be deemed intended for selection by the user.

Consider the bounds calculated for the line object seen in Figure 9.1 and repeated in Figure 9.4 as compared to the sample event using the `gxObjSelect` function found in Listing 9.4.

Listing 9.4 Testing Object Selection

```
1:     Boolean gxObjSelect( Bounds bnds, XEvent *xe )
2:     {
3:       if( xe->xbutton.x > bnds->x &&
4:           xe->xbutton.y > bnds->y &&
5:           xe->xbutton.x < (bnds->x + bnds.width) &&
6:           xe->xbutton.y < (bnds->y + bnds.height)) {
7:
8:         return True; // object selected
9:       }
10:      return False; // object not selected
11:    }
```

The function shown in Listing 9.4 uses the bounds calculated from the line object passed to the `getLineBounds` shown in Listing 9.2.

Comparisons are made to determine whether the event occurred at a point greater than the origin of the object

```
3:      if( xe->xbutton.x > bnds->x &&
4:          xe->xbutton.y > bnds->y &&
```

and less than the extents of the object.

```
5:          xe->xbutton.x < (bnds->x + bnds.width) &&
6:          xe->xbutton.y < (bnds->y + bnds.height)) {
```

Notice that the extents of the object are defined as where the horizontal edge and vertical edge of the object are placed on the screen relative the object's origin.

If any element of the `if` statement fails, the entire test will fail because of the use of the Boolean AND (`&&`) function.

Applying the values of the illustration, the line object would appear selected by the user using this method of determination.

Refer to Listing 9.4 for a demonstration of the application of the `ButtonPress` event to the `XArc` example from Figure 9.3. It too appears to be selected.

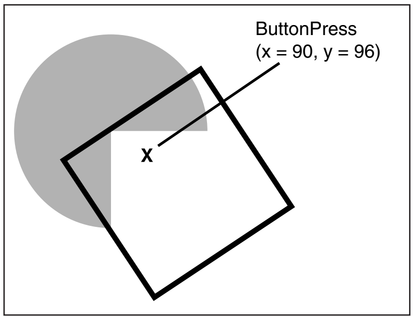
The shortcoming highlighted in these examples is the confusion that occurs using the method of selection shown in Listing 9.4.

Not only do both objects appear to be selected, but also the `ButtonPress` event did not even touch the line object.

Consider the `ButtonPress` event illustrated in Figure 9.5.

Figure 9.5

ButtonPress for object selection.



Again the `ButtonPress` event is within the bounds of both the sample objects. This time the event touches neither of the objects, which, when using the `gxObjSelect` from Listing 9.4, makes both objects appear selected.

This is a limitation of the selection function that must be overcome to ensure correct object control and manipulation.

Next Steps

To select objects properly using the Graphics Editor, a method for determining whether an object is selected must return `true` only if the `ButtonPress` event explicitly falls on the object.

To accomplish a selection function of the required sophistication, an understanding of certain trigonometric functions must first be discussed for detecting whether a point falls on a line or lies beyond the arc defined by the end and center points of an arc object.

Chapter 10, “Trigonometric and Geometric Functions,” demonstrates a finer method of determining object selection and reviews the mathematical skills necessary for the task.

Chapter 10

In this chapter

Trigonometric and Geometric Functions

Chapter 9, "Object Bounds Checking," showed the necessity of accurately selecting graphic objects created by the editor and the frailty of using solely the bounds of the object for making the determination of selection. This chapter reviews the mathematics required for a more precise selection algorithm.

Chapter 9 discussed the objects of the editor as being point-array–based or as being an arc. We first consider the intersection of a point and a line.

Calculating Point and Line Intersections

Objects created by our editor that are defined as an array of points can be considered a series of line segments where each pair of points in the array defines the segment's endpoints.

For instance, an array of four points defined as

```
XPoint pts[4] = {{10,10}, {20,10}, {20,20}, {10,20}};
```

would consist of three line segments:

1. `(10,10)` to `(20,10)`
2. `(20,10)` to `(20,20)`
3. `(20,20)` to `(10,20)`

Note

The function `XDrawLines` does not implicitly *close* the object represented by the point array, meaning that the endpoints are not connected. To accomplish this, the array would have to be expanded to contain five points and the starting point repeated as the last element of the array.

Having reduced our point-array–based object to a series of line segments, we can further clarify the precise selection issue by viewing the `ButtonPress` event entered in the canvas window if the user wants to select an object as the point where the event occurred.

Defining the graphic object as a series of line segments and the event for selection as merely a point clarifies the requirement of precise object selection. Knowing when an object is selected requires the capability to determine when the point represented by the event intersects a line segment of the object.

Listing 10.1 shows the algorithm for accomplishing this form of selection.

Listing 10.1 The `near_segment` Function

```
/*
 * near_segment
 */
Boolean near_segment( int x1, int y1, int x2, int y2, int xp, int yp )
{
  int xmin, ymin, xmax, ymax;
  float slope, x, y, dx, dy;

  if( abs(xp - x1) <= TOLERANCE && abs(yp - y1) <= TOLERANCE ) {
    return True;
  }

  if( abs(xp - x2) <= TOLERANCE && abs(yp - y2) <= TOLERANCE ) {
    return True;
  }

  if( x1 < x2 ) {
    xmin = x1 - TOLERANCE;
    xmax = x2 + TOLERANCE;
  } else {
    xmin = x2 - TOLERANCE;
    xmax = x1 + TOLERANCE;
  }

  if( xp < xmin || xmax < xp ) {
    return False;
  }

  if( y1 < y2 ) {
```

```
30:       ymin = y1 - TOLERANCE;
31:       ymax = y2 + TOLERANCE;
32:     } else {
33:       ymin = y2 - TOLERANCE;
34:       ymax = y1 + TOLERANCE;
35:     }
36:
37:     if (yp < ymin || ymax < yp)
38:       return False;
39:
40:     if( x2 == x1 ) {
41:       x = x1;
42:       y = yp;
43:     } else if( y1 == y2 ) {
44:       x = xp;
45:       y = y1;
46:     } else {
47:       slope = ((float) (x2 - x1)) / ((float) (y2 - y1));
48:       y = (slope * (xp - x1 + slope * y1) + yp) / (1 + slope * slope);
49:       x = ((float) x1) + slope * (y - y1);
50:     }
51:
52:     dx = ((float) xp) - x;
53:     dy = ((float) yp) - y;
54:
55:     if ( (float)(dx * dx + dy * dy) < (float)(TOLERANCE*TOLERANCE) ) {
56:       return True;
57:     }
58:
59:     return False;
60:   }
```

The function `near_segment` shown in Listing 10.1 considers only a single line segment represented by endpoints

```
int x1, int y1, int x2, int y2
```

passed as the first four parameters of the function.

```
4:    Boolean near_segment( int x1, int y1, int x2, int y2, int xp, int yp )
```

The final two parameters are the x and y points contained in the `ButtonPress` event.

The function begins by testing the distance of the event point `xp` and `yp` from each of the endpoints defining the line segment.

```
9:      if( abs(xp - x1) <= TOLERANCE && abs(yp - y1) <= TOLERANCE ) {
```

and

```
13:     if( abs(xp - x2) <= TOLERANCE && abs(yp - y2) <= TOLERANCE ) {
```

Note

The `TOLERANCE` variable used in various places throughout the Graphics Editor project is defined in a header file not yet introduced.

Its use is to set the allowable margin of error for interactive functions. Specifically, when the user attempts to click on the line object, he might be `TOLERANCE` distance from it and still successfully accomplish the selection.

This is necessary because the line object can only be a single pixel wide (line width of 1). As the event is also a single pixel (x, y) point, the level of precision required to have the event land exactly on the line object could be too great considering the steadiness of most of our hands.

EXCURSION

You Must Exercise Your abs

The function `abs` used in testing the difference between the event points and the line segment endpoints

```
abs(xp - x1)
```

and

```
abs(yp - y1)
```

is the absolute value function.

Because the `near_segment` function does not test whether the event point is less than the endpoint before doing the subtraction, it is possible to have a negative result.

The `abs` function requests that the resulting sign of the operation be stripped, forcing the result to be positive.

Without the `abs` function, the determination of the result to be less than or equal `TOLERANCE` would not always have the intended results, as an event point that was to the right of the endpoint would appear to be within a distance of `TOLERANCE` or 3 pixels.

The intended semantic is to determine whether the event is within 3 pixels to *either* side of the endpoint, necessitating that the absolute value of the distance be tested.

If the event points are an acceptable distance from one of the line segment's endpoints, the function returns `true` and the line object is considered selected.

If the event point is somewhere within the line segment, the function continues by ordering the points defining the line segment.

```
    if( x1 < x2 ) {
      xmin = x1 - TOLERANCE;
      xmax = x2 + TOLERANCE;
    } else {
      xmin = x2 - TOLERANCE;
```

```
        xmax = x1 + TOLERANCE;
      }
```

The ordering of the points is accomplished by testing to see whether `x1` is less than `x2`. If `x1` is the lesser of the two points, its value is assigned as `xmin`, making the value of `x2` the maximum or `xmax` value.

Note

By subtracting `TOLERANCE` from the `xmin` value and adding it to the `xmax` value, you increase the bounding box defined by the line segment.

By ordering the points, `near_segment` can perform a *range check* to determine whether the event point occurred in close enough proximity to the line segment to merit continuing in the function.

If the event point is outside of the bounds of the line segment, there is no reason to perform any more calculations and tests because it would be impossible for the event point to intersect the line.

Note

Notice that the bounds checking being performed here is not the same as the bounds checking done in the previous chapter.

Here only a single line segment comprising the whole object is being tested, making for a much finer determination. Further, an event point lying within the bounds of the segment is used only to determine whether further calculations and testing should be performed and not whether the object should be considered selected.

After determining that the event point is within the bounds of the line object, a test is made to ensure that the x value of the endpoints are not equal, indicative of a vertical line. If the line is vertical, the variables `x` and `y` are assigned the x value of the endpoints (`x1`) and the y component of the event point (`yp`) respectively.

```
    if( x2 == x1 ) {
      x = x1;
      y = yp;
```

If the line is not vertical (x components of the endpoints are not equal), the line is tested for being horizontal. A horizontal line will have equal y values for the endpoints and the y value of the endpoints and event x component will be used for the values of the variables `x` and `y`.

```
    } else if( y1 == y2 ) {
      x = xp;
      y = y1;
```

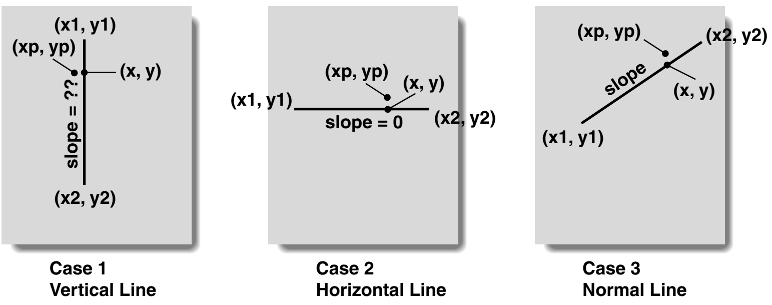
If the line is neither vertical nor horizontal, the slope of the line is calculated and used to determine the values of the x and y variables.

```
    } else {
      slope = ((float) (x2 - x1)) / ((float) (y2 - y1));
      y = (slope * (xp - x1 + slope * y1) + yp) / (1 + slope * slope);
      x = ((float) x1) + slope * (y - y1);
    }
```

Figure 10.1 demonstrates the relationship of the variables `slope`; `x1`, `y1`; `x2`, `y2`; `xp`, `yp`; and `x`, `y`.

Figure 10.1

A Depiction of the terms used when calculating Point Line Intersection.



As shown in Figure 10.1, the variables x and y will be some point on the line as determined by one of the three steps we've seen.

Calculating Slope

The slope of a line is defined as the steepness or tilt of the line. It is measured as a ratio of vertical change (rise) over horizontal change (run). In mathematics, slope is usually designated by the letter *m*.

Given a line passing through points (x1, y1), (x2, y2) the slope m of the line determined by

$$m = \frac{y2 - y1}{x2 - x1}$$

as long as $x2 \neq x1$.

Note

The slope of a vertical line is undefined and the slope of a horizontal line is zero. For this reason, `near_segment` special cases these conditions.

Converting the equation for calculating slope to the C programming language is straightforward

```
        slope = ((float) (x2 - x1)) / ((float) (y2 - y1));
```

Note

The use of the keyword `float` in the calculation of slope is called a *cast*.

Casting variables is necessary to promote their data types either for compatibility during assignments or precision in calculations.

The cast of the result of `(x2 - x1)` and `(y2 - y1)` to the data type of float ensures that the division operation is done with floating point numbers. A great deal of precision would be lost if slope were calculated using integers or whole numbers.

Calculating the `x`, `y` point on the line as required in Case 3 of Figure 10.1 is not as straightforward.

Case 3 requires that functions for determining `x` and `y` be derived from one or more of the forms for writing *linear equations* and applied to what is known about the line segment and event point.

```
        y = (slope * (xp - x1 + slope * y1) + yp) / (1 + slope * slope);
        x = ((float) x1) + slope * (y - y1);
```

EXCURSION

Common Equations for Representing a Line

Linear Equations are functions for representing lines based on the information known about them.

Table 10.1 shows three of the more common forms for writing linear equations.

Table 10.1 Linear Equations

Form	*Function*	*Description*
General	Ax + By = C	A, B, and C are not fractions
Slope-Intercept	y = mx + b	b is the y-intercept or point at which the y-axis is crossed
Point-Slope	y–y1 = m(x–x1)	The line is known to contain the point (x1, y1)

For instance, employing the Point-Slope form and solving independently for x and y identifies the point needed by Case 3 of Figure 10.1.

When the point (`x`, `y`) is found, the distance of this point from the event point is calculated:

```
dx = ((float) xp) - x;
dy = ((float) yp) - y;
```

Finally, the line is considered selected if twice the sum of the deltas squared (distance between the x and y component of the event point and the point `x`, `y` found on the line) is less than the square of the tolerance.

```
if ( (float)(dx * dx + dy * dy) < (float)(TOLERANCE + TOLERANCE) ) {
  return True;
}
```

When the evaluation of the `if` condition results in `False`, the program will fall through to

```
return False;
```

because the line segment was not selected.

If the `near_segment` evaluation for any one of the line segments comprising the entire point-array–based object results in a return value of `True`, the object is considered selected and no further consideration is necessary.

The solution presented by the `near_segment` function is clearly superior albeit more complex than using only the bounds of the object for determining the users intent for object selection.

Selection of point-array–based objects has been improved greatly over the bounds method introduced in Chapter 9. We now turn our attention to the selection of arcs.

Calculating Point and Arc Intersections

As introduced in the previous chapter and iterated again in the context of refining the selection process for point-array–based objects, using solely the bounds of an object to determine whether a `ButtonPress` event point was meant to select the object is insufficient.

In the same way that `near_segment` calculated the proximity of the event point to the segments defining the object, a method is needed to accomplish the same for arcs.

As introduced in Chapter 7, “Xlib Graphic Primitives,” in the section “XDrawArc”, page 192, arcs are described in the X Window System by

```
x, y, width, height, angle1, angle2
```

To facilitate this representation, X provides the `XArc` structure for holding the required arc data.

```
typedef struct {
    int x, y;
    Dimension width, height;
    int angle1, angle2;
} XArc;
```

The purpose of the `arc_find` function found in Listing 10.2 is to determine whether an event point intersects the arc defined by an XArc structure and specified as the first parameter.

Listing 10.2 The `arc_find` Function

```
/*
 * arc_find
 */
static Boolean arc_find( XArc *arc_data, XEvent *event )
{
  double rx,     ry,
         cx,     cy,
         ex,     ey,
         f1x,    f1y,
         f2x,    f2y,
         hmaj,   hmin,
         d,      angle,
         angle1, angle2;

  rx = (float)arc_data->width  / 2;
  ry = (float)arc_data->height / 2;

  cx = arc_data->x + rx;
  cy = arc_data->y + ry;

  d  = sqrt( fabs( sqr(rx) - sqr(ry) ) );
  if( rx >= ry ) {
    f1x  = cx - d;
    f1y  = cy;
    f2x  = cx + d;
    f2y  = cy;
    hmaj = rx;
    hmin = ry;
  } else {
    f1x  = cx;
    f1y  = cy - d;
    f2x  = cx;
    f2y  = cy + d;
    hmaj = ry;
    hmin = rx;
  }

```

continues

Listing 10.2 Continued

```
    ex = event->xbutton.x;
    ey = event->xbutton.y;

    angle  = atan2( cy - ey, ex - cx ) * 180/M_PI;
    if( angle < 0 )
      angle += 360;

    angle1 = (double)arc_data->angle1 / 64.0;
    angle2 = (double)arc_data->angle2 / 64.0;

    if( sqrt( sqr(f1x - ex) + sqr(f1y - ey) ) +
      sqrt( sqr(f2x - ex) + sqr(f2y - ey) ) > 2 * (hmaj + TOLERANCE))
      return( False );
    if( sqrt( sqr(f1x - ex) + sqr(f1y - ey) ) +
      sqrt( sqr(f2x - ex) + sqr(f2y - ey) ) < 2 * hmaj - TOLERANCE )
      return( False );
    return( True );
  }
```

Following the declaration of the many variables used to determine whether the event point intersects the arc is the calculation of half of the *major* and *minor* axes.

```
    rx = (float)arc_data->width  / 2;
    ry = (float)arc_data->height / 2;
```

Note

The *major* and *minor axis* refer to the diameter on the x-axis and y-axis. The longer of the two is called the major axis and the shorter is the minor axis.

Half the distance of the major and minor axis determines the radius on the x-axis and the radius on the y-axis.

Independent values are maintained for radius on the x-axis and radius on the y-axis because the *aspect ratio* of the arc might not be constant.

Note

Aspect ratio is the proportion of width to height.

A 360° arc with a constant aspect ratio has a width and height that are equal and the resulting object is a circle. Therefore, a circle's major and minor axes are equal, or in other words, the diameter is constant on both the x-axis and the y-axis.

If the width and height for the same arc object are not equal (the aspect ratio is not constant), the resulting object is an ellipse. An ellipse has a different diameter on the x-axis and the y-axis.

Care has been taken to not use the term *circle* or *ellipse* to describe the corresponding Graphics Editor objects because there is no enforcement of a constant aspect ratio. Instead, the generic description *arc* is used to describe both circles and ellipses as well as actual arcs (objects that are less than 360°) created by a user.

Following the calculation of the radii for the x and y-axes, the center point of the arc is found.

```
cx = arc_data->x + rx;
cy = arc_data->y + ry;
```

Next, we calculate the distance between the center of the ellipse and the *foci* as well as the coordinates of the foci. This calculation is dependent upon the orientation of arc (in the case of a non-constant aspect ratio). So a test is made on whether the radius on the x-axis is greater than or equal to the radius on the y-axis.

EXCURSION

It Was Not Foci Bearing Gifts at the Nativity?

The foci are *fixed points* from which all points on the *arc* (circle or ellipse) are a constant distance. Other than being a cool word to say, foci are crucial to the definition of an arc.

Foci is plural and in this context refers to two fixed points, one relative to the major axis and one to the minor axis.

In the case of a circle, the two points are the same and correspond with the center of the circle.

For an ellipse, the major axis containing the foci is always longer than the minor axis. Thus, a test to determine the longer of the two axes is necessary to correctly calculate the distance of the foci from the center and the foci coordinates.

```
if( rx >= ry ) {
```

This definition of foci prepares you for a definition of a circle and an ellipse to demonstrate the use of foci.

A circle is defined as

```
PO + PO = 2a
```

where *a* is the radius of the circle, *P* is any point on the circle, and *O* is the foci. (Remember that the major and minor axes are equal and therefore the foci are equal.)

Figure 10.2 illustrates the definition of a circle.

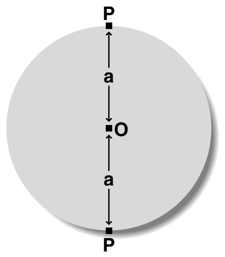


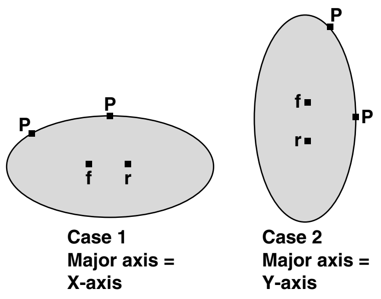
Figure 10.2
The definition of a circle.

As demonstrated in Figure 10.2, *PO = a* is the radius of the circle and therefore *2a* is the diameter. In other words, distance *PO* from any point *P* on the circle to the (convergence) of the foci *O* is the constant *a*.

In an ellipse, however, the foci are not equal and they do not correspond to the center point of the object, as demonstrated in Figure 10.3.

Figure 10.3

Foci of an ellipse.



If we name one focus (singular for foci) f and the other r, we can form the definition of an ellipse as

```
Pf + Pr = 2a
```

where *P* is any point on the ellipse.

The definition reads that the distance from the foci of an ellipse is the same for all points on the curve. Thus if *r* and *f* are the foci, the total distance *Pf* + *Pr* from the foci to any point *P* is constant (*2a*).

Consider the first case demonstrated in Figure 10.3 where the major axis coincides with the x-axis

```
    if( rx >= ry ) {
```

The foci are calculated to be

```
      f1x  = cx - d;
      f1y  = cy;
```

and

```
      f2x  = cx + d;
      f2y  = cy;
```

Consistent with the explanation, the major axis (`hmaj`) is assigned the value of radius on the x-axis, and the minor axis (`hmin`) the value on the y-axis.

```
      hmaj = rx;
      hmin = ry;
```

Case 2 illustrated in Figure 10.3, however, sets the major and minor axis to

```
      hmaj = ry;
      hmin = rx;
```

and assigns the coordinates of the foci to be

```
f1x  = cx;
f1y  = cy - d;
f2x  = cx;
f2y  = cy + d;
```

More information is required before actually determining whether the event point intersects the angle.

First, determine the angle measured from the positive x-axis (3 o'clock position) of the vector (ex, ey), (cx, cy) measured in degrees:

```
atan2( cy - ey, ex - cx ) * 180/M_PI
```

Note

The constant `M_PI` is defined as

```
#define M_PI 3.14159265358979323846
```

Its use in calculating the angle of the vector defined by the center point of the ellipse and the event point is to convert the Radian value returned by the `atan2` to Degrees as the function returns a value in the range –`M_PI` and `M_PI`.

```
angle  = atan2( cy - ey, ex - cx ) * 180/M_PI;
```

If the angle is less than zero, rotate it 360°:

```
if( angle < 0 )
    angle += 360;
```

Next, convert the angle1 and angle2 measurements used by X to degrees. As you recall from the introduction to XDrawArc in Chapter 7, section "XDrawArc," page 192, X maintains these angles as integers in the form "degrees × 64."

```
angle1 = (double)arc_data->angle1 / 64.0;
angle2 = (double)arc_data->angle2 / 64.0;
```

Finally, apply the information gathered and see whether the arc has been selected.

The determination is whether the sum of the distances between the event point and the foci is greater than the major axis by more than `TOLERANCE`. If it is greater, the arc has not been selected.

Consider each focus separately. First, (`f1x`, `f1y`)

```
if( sqrt( sqr(f1x - ex) + sqr(f1y - ey) ) +
    sqrt( sqr(f2x - ex) + sqr(f2y - ey) ) > 2 * (hmaj + TOLERANCE))
    return( False );
```

Then, (`f2x`, `f2y`)

```
if( sqrt( sqr(f1x - ex) + sqr(f1y - ey) ) +
    sqrt( sqr(f2x - ex) + sqr(f2y - ey) ) < 2 * hmaj - TOLERANCE )
    return( False );
```

If the sum of the differences is not greater, the function returns `true` to indicate that the arc has successfully been selected by the event point.

```
  return( True );
```

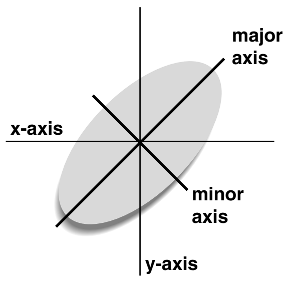
EXCURSION

Introducing a Limitation of the X Window System

A limitation of the X Window System is the inability to draw ellipses that have a major axis and minor axis that are not parallel to the x-axis and y-axis. Figure 10.4 illustrates such an ellipse.

Figure 10.4

Ellipse that is impossible to represent in X.



One solution is to convert the ellipse into a series of line segments using a Bézier algorithm and then treat it like a point-array–based object.

With a superior method of selecting both point-array–based objects and arc objects, you are ready to review the concepts required to perform geometric transformations.

Next Steps

Chapter 11, "Graphic Transformations," introduces and satisfies the requirements for performing actions such as moving, scaling, and rotating the objects of the Graphics Editor. Understanding these actions (known as geometric transformations) will bring you close to completing the foundation the last several chapters have so meticulously worked to lay out.

Chapter 11

In this chapter

- *Moving*
- *Scaling*
- *Rotating*
- *Next Steps*

Graphic Transformations

We have studied calculations for accurately determining when objects in the Graphics Editor are selected from the canvas, and we now shift our focus to the actions that can be applied to an object that has been successfully selected.

Graphic transformations, as implied by the phrase, are actions that alter the graphic object. The Graphics Editor supports the move, scale, and rotate functions for altering the objects of the editor.

Each of these is discussed in the following sections. As was necessary with the analysis of proper object selection, consideration is given separately to point-based objects and arc objects because the transformation methods are unique to the data fields of the varying objects.

Moving

Moving an object, regardless of type, requires the application to maintain the delta or distance the cursor has traveled between iterations of the move request.

To move an object in the Graphics Editor, first select the object and then press and hold the right mouse button over the object while moving the cursor. The distance the cursor travels is applied to the *location* designator of the object.

For point-array–based objects, the object's location is implicit to the values of the points contained in the array.

Arcs, however, have an explicit x, y value that describes their placement in the canvas window.

The following two sections show the functions that manage the updates required for the point-based and arc objects to move, based on the coordinates of an X event understood to be a `PointerMotion` event structure.

Moving a Line

Listing 11.1 demonstrates how the Graphics Editor tracks and applies the distance the cursor has traveled. Although the `GXObj` has not been defined, you know from Listing 9.2 in Chapter 9, "Object Bounds Checking," the definition of the `GXLine` structure. This structure provides the unique data field definition for `GXObj`s of type line.

➔ The **GXLine structure** is presented in Chapter 9, section "Point-Array–Based Object Bounds," page 204.

Listing 11.1 The `line_move` Function

```
static void line_move( GXObjPtr line, XEvent *event )
{
  static int x = 0, y = 0;

  GXLinePtr line_data = (GXLinePtr)line->data;
  int i;

  if( x && y ) {
    XDrawLines( XtDisplay(GxDrawArea), XtWindow(GxDrawArea), rubberGC,
                line_data->pts, line_data->num_pts, CoordModeOrigin );
  } else {
    /* our first time through */
    (*line->erase)( line );

    x = event ? event->xbutton.x : 0;
    y = event ? event->xbutton.y : 0;
  }

  if( event ) {
    for( i = 0; i < line_data->num_pts; i++ ) {
      line_data->pts[i].x += (event->xbutton.x - x);
      line_data->pts[i].y += (event->xbutton.y - y);
    }

    /*
     * draw rubberband line
     */
    XDrawLines( XtDisplay(GxDrawArea), XtWindow(GxDrawArea), rubberGC,
                line_data->pts, line_data->num_pts, CoordModeOrigin );

    x = event->xbutton.x;
    y = event->xbutton.y;
  } else {
```

```
34:         x = 0;
35:         y = 0;
36:       }
37:    }
```

The `line_move` function in Listing 11.1 begins by declaring the local static variables:

```
3:       static int x = 0, y = 0;
```

EXCURSION

The Keyword `static` Has Multiple Uses

Contrast the use of the keyword `static` as applied to local variables compared to its use in the declaration of the `line_move` function

```
1:     static void line_move
```

➔ See Chapter 3, page 103 for a discussion of the **`static`** keyword.

When applied to local variables, the `static` keyword ensures that their values are residual (maintained) between calls to the function. This is critical to the `line_move` function because it must retain the (x, y) values of the `PointerMotion` event driving the placement of the object being moved.

Applied to the function declaration, however, `static` simply limits the scope (visibility) of the function to this file.

The static variables `x`, `y` retain the coordinates of the X `PointerMotion` event, allowing only the incremental delta of the distance the pointer has traveled to be applied to the points defining the object.

In a moment, you will see that the value of `0` assigned to the `x` and `y` is important for knowing when the move action begins.

The next executable line of the function

```
5:       GXLinePtr line_data = (GXLinePtr)line->data;
```

declares a variable `line_data` and assigns it the unique data field from the `GXObjPtr line` passed as the first parameter to the `line_move` function.

After declaring a simple integer for looping, the function tests to see if `x` and `y` have non-zero values:

```
6:       int i;
7:
8;       if( x && y ) {
```

Non-zero values for the x and y indicate that this is not the first iteration of the move action, an important thing to know because the *rubber-banding* object must be erased from the screen.

```
      XDrawLines( XtDisplay(GxDrawArea), XtWindow(GxDrawArea), rubberGC,
                   line_data->pts, line_data->num_pts, CoordModeOrigin );
```

The rubber-banding effect used during object creation and transformation in the Graphics Editor project refers to drawing the object in an *interactive mode*.

Allowing the user to move an object around the screen, other objects existing on the canvas are not erased by the drawing and erasing of the object in interactive mode.

- ➔ Review the immediate graphic nature of X as discussed in Chapter 4, "Windowing Concepts," section "Expose," page 122.
- ➔ To prevent erroneously erasing other objects on the canvas, the GC function `GXxor` introduced in Chapter 7, "Xlib Graphic Primitives," section "The `GC` Function," page 182, is assigned the global `GC rubbergc`.

The `rubbergc` graphic context will draw the object the first time the X Graphic Primitive is requested and erase it the second time without harming the underlying background or objects.

Caution (and careful management) must be applied to the number of calls made to a primitive employing a `GC` with `GCFunction` set to `GXxor` because an odd number of calls will leave residual rubber-banded objects onscreen.

If the `x` and `y` values are zero, this is seen as the first iteration of the move request, in which case the `else` is reached. In the body of the `else`, the erase method of the `GXObj` is invoked to clear the object from the canvas. It is immediately redrawn in the interactive (non-destructive) mode using the `rubbergc`.

```
    } else {
      /* our first time through */
      (*line->erase)( line );

      x = event ? event->xbutton.x : 0;
      y = event ? event->xbutton.y : 0;
    }
```

In the body of the `else`, calling the object's erase method removes the object from the canvas, and then the values of `x` and `y` are set to equal the coordinates of the `PointerMotion` event *if it exists*.

Note

Although the specific `GXObj` definition is not introduced until Chapter 15, “Common Object Definition,” suffice it to say that an object will have methods (internal functions) for accomplishing all necessary self-management tasks pertinent to the type of object it is.

Object methods include functions for drawing, erasing, moving, scaling, rotating, copying, saving, restoring, and deleting.

Perhaps Chapter 15 should have been titled “A Horse for the Cart.”

Note

The object’s method for erasing is not compatible with the call to the X primitive `XDrawLine` that is used to manage the rubber-banding effect.

The incompatibility results from the difference in `GC` function used to draw the object on the screen.

All the `GXObj` methods for drawing employ the `GXcopy` function that cannot be erased or undone with the `GXxor` function.

➔ As Chapter 20, “Latex Line Object,” demonstrates, and Chapter 7, “X Lib Graphic Primitives,” section “GCTile,” page 188 introduced, the method of erasing objects placed onscreen with the `GXcopy` is to *tile* in the canvas background where the object being erased currently resides.

EXCURSION

A New Form of the `if then else` *Construct*

The C language syntax

```
var = test ? value1 : value2;
```

is equivalent to

```
if( test )
  var = value1;
else
  var = value2;
```

Although not clear or readily readable, the notation is very common with C programmers.

The necessity of testing for the implicit non-null value of `event` is to account for the transformation function `line_move` being interrupted or ended.

To abort or end the move action, a null event pointer is passed to the `line_move` function, allowing it to reset the values of `x` and `y` to zero so the transformation can start over with another object.

The implicit test for a non-null event follows the body of the `else` entered when the values of `x` and `y` were zero.

Note

An implicit test follows the form

```
if( variablePtr )
```

C interprets this as a test for a non-zero value. As NULL is defined in C as #define NULL (unsigned long)0 or #define NULL (void *)0.

The test is compatible with pointer as well as non-pointer variables.

An explicit test is clearly preferred over an implicit test because it prevents semantic errors and makes the test more readable. Explicit tests, as the phrase implies, explicitly test for the expected value. The equivalent explicit test for the sample given above is

```
if( variablePtr != NULL )
```

As discussed earlier, sending a null event pointer causes the function to reset the values of x and y so the move action can be repeated.

```
    if( event ) {
      for( i = 0; i < line_data->num_pts; i++ ) {
        line_data->pts[i].x += (event->xbutton.x - x);
        line_data->pts[i].y += (event->xbutton.y - y);
      }
```

If there is a valid event pointer, the difference between the previous and current `PointerMotion` event location is applied to every point in the `line_data->pts` array.

After updating all the points, the object is implicitly placed at the new location and the rubber-banding object is drawn to reflect this by using the global `GC` variable `rubberGC`.

```
      XDrawLines( XtDisplay(GxDrawArea), XtWindow(GxDrawArea), rubberGC,
                  line_data->pts, line_data->num_pts, CoordModeOrigin );
```

With the interactive object drawn to the screen, it is imperative that the current event location be retained so that the next iteration of the function can erase it.

```
      x = event->xbutton.x;
      y = event->xbutton.y;
```

As the `line_move` function is invoked, the line object is continually updated to the new cursor location by adding the difference that the cursor moved from the previous call.

A similar `move` function must exist for the `GXObj` of type arc.

Moving an Arc

The method of moving an arc object is simpler than the point-array–based object because its location is assigned explicitly by the `x` and `y` data fields in the `XArc` structure.

Listing 11.2 shows the function for performing the move transformation on an arc object.

Listing 11.2 The `arc_move` Function

```
/*
 * arc_move
 */
static void arc_move( GXObjPtr arc, XEvent *event )
{
  static int x = 0, y = 0;
  XArc *arc_data = (XArc *) arc->data;

  if( x && y ) {
    /*
     * erase the rubberband arc
     */
    XDrawArc( XtDisplay(GxDrawArea), XtWindow(GxDrawArea), rubberGC,
              arc_data->x, arc_data->y,
              arc_data->width, arc_data->height,
              arc_data->angle1, arc_data->angle2 );

  } else {
    /*
     * our first time through - erase the actual arc...
     */
    (*arc->erase)( arc );

    /*
     * ...store the current event location
     */
    x = event ? event->xbutton.x : 0;
    y = event ? event->xbutton.y : 0;
  }

  if( event ) {
    /*
     *  get the x,y delta
     */
    arc_data->x += (event->xbutton.x - x);
    arc_data->y += (event->xbutton.y - y);

    /*
     * draw a rubberband arc
     */
```

continues

Listing 11.2 Continued

```
   XDrawArc( XtDisplay(GxDrawArea), XtWindow(GxDrawArea), rubberGC,
             arc_data->x, arc_data->y,
             arc_data->width, arc_data->height,
             arc_data->angle1, arc_data->angle2 );

   x = event->xbutton.x;
   y = event->xbutton.y;
 } else {
   x = 0;
   y = 0;
 }
}
```

The `arc_move` function shown in Listing 11.2 should be very familiar because it follows the form of the `line_move` function except that the data type `XArc` is being managed instead of a `GXLine`.

Focusing only on the differences between the `line_move` and `arc_move` functions, consider the assignments that explicitly place the arc object at the new location.

```
   arc_data->x += (event->xbutton.x - x);
   arc_data->y += (event->xbutton.y - y);
```

and the graphic primitive, which draws and erases the interactive object:

```
   XDrawArc( XtDisplay(GxDrawArea), XtWindow(GxDrawArea), rubberGC,
             arc_data->x, arc_data->y,
             arc_data->width, arc_data->height,
             arc_data->angle1, arc_data->angle2 );
```

Beyond these few lines, the `arc_move` function behaves exactly like the `line_move` function.

Advancing an understanding of graphic transformations, the next action to consider is scaling.

Scaling

The transformation of scaling an object refers to changing the object’s height and width.

For many objects the width and height values refer only to the values defined by the object’s bounds. Altering the width and height of the bounds of the object will affect each of the segments comprising the object because they must be *proportionally* altered to accomplish or obtain the new bounds of the object.

The challenge of scaling an object is that the x and y deltas (determined as they were with the move transformation by calculating the difference of the current and previous event locations) are not applied linearly to the object. Rather, the *direction* of the scale action must be considered and a greater portion of the delta applied to the side corresponding to the direction of the scale than to the opposing side.

The Graphics Editor applied eight handles to an object made active by the selection process.

Figure 11.1 and Figure 11.2 illustrate how the handles are placed on Line and Arc objects respectively.

Figure 11.1

Illustrating the Line object's handles.

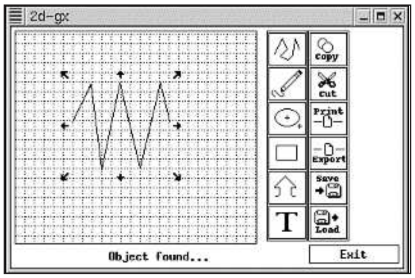
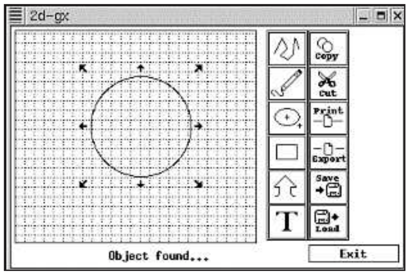


Figure 11.2

Illustrating the Arc object's handles.



The placement of object handles when the user selects an object corresponds to every corner and every side of the object as shown in Figures 11.1 and 11.2.

A *scale action* is instigated when the user selects an object to reveal its handles, positions the mouse cursor over the desired handle, and presses and holds the left mouse button to select the handle.

As long as the left mouse button is pressed, subsequent `PointerMotion` events will alter the width or height of the object in a manner relative to the handle selected.

When the mouse button is released, the object is redrawn with the new dimensions.

Returning to the discussion of the scaling challenge, consider the object shown in Figure 11.1. If the handle on the right side of the object is selected for the scale action, the point within the object closest to the handle must change with a greater proportion of the delta than the point furthest away.

If all points changed an equal amount, the result would be a move action instead of the intended scale action.

The final challenge when performing the scale transformation is the loss of *data integrity*.

As calculations are made to determine the proportion of the delta to apply to the object fields controlling the object's dimensions, *rounding errors* will occur.

EXCURSION

Rounding Errors Are a Loss of Precision

Rounding errors occur when a floating-point number such as 1.5 is assigned to a variable of type `int` (integer). The portion of the number following the decimal point is lost because an integer cannot represent a decimal value.

This can occur when the delta being applied to a point is 3; however, due to the position of the point within the object relative to the handle controlling the scale action, only 50 percent of the delta is applied.

The result is (3 × 0.5) or 1.5 applied to the data point.

Further, it is necessary to use integer variables to represent the points or data fields within the graphic objects because it is impossible to draw half a pixel; therefore, everything must be rounded to the nearest integer number or whole pixel.

As rounding errors compound, the data representing the object is affected, and, for instance, lines are not as straight as they should be.

The function implementing the scale transformation must account for rounding errors. One method to minimize the effect of rounding errors is to ensure that all transformations are done to the original points or data fields. In this way, the rounding errors will not compound. At worst only a fraction of a pixel is lost, as opposed to a fraction of a pixel being lost for each iteration of the scale transformation.

The following sections present the functions for scaling point-array–based and arc objects accounting for applying only a portion of the delta based on the handle selected and ensuring data integrity by minimizing rounding errors.

Scaling a Line

The code in Listing 11.3 is the entry point for accomplishing scaling of point-array–based objects. As discussion focuses on the body of the function, keep in mind the tasks that must be accomplished by the scale transformation: namely, properly proportioning the delta applied and minimizing the effects of rounding errors.

Listing 11.3 The `line_scale` Function

```
static void line_scale( GXObjPtr line, XEvent *event )
{
  static GXLinePtr tmp_data = NULL;
  GXLinePtr line_data = (GXLinePtr)line->data;

  if( tmp_data ) {
    XDrawLines( XtDisplay(GxDrawArea), XtWindow(GxDrawArea), rubberGC,
                tmp_data->pts, tmp_data->num_pts, CoordModeOrigin );
  } else {
    /* our first time... */
    (*line->erase)( line );

    tmp_data = (GXLinePtr)XtNew(GXLine);
    tmp_data->num_pts = line_data->num_pts;
    tmp_data->pts =
           (XPoint *)XtMalloc( sizeof(XPoint) * tmp_data->num_pts );
    get_bounds( line_data->pts, line_data->num_pts,
                &OrigX, &OrigY, &ExntX, &ExntY );
  }

  if( event ) {
    memcpy( (char *)tmp_data->pts, (char *)line_data->pts,
            sizeof(XPoint) * tmp_data->num_pts );

    apply_delta( tmp_data->pts, tmp_data->num_pts,
                 FixedX - event->xbutton.x,
                 FixedY - event->xbutton.y );

     XDrawLines( XtDisplay(GxDrawArea), XtWindow(GxDrawArea), rubberGC,
                 tmp_data->pts, tmp_data->num_pts, CoordModeOrigin );
  } else {
    if( tmp_data ) {
      memcpy( (char *)line_data->pts, (char *)tmp_data->pts,
              sizeof(XPoint) * line_data->num_pts );

      XtFree((char *)tmp_data->pts);
      XtFree((char *)tmp_data);

      tmp_data = NULL;
    }
  }
}
```

In reading through Listing 11.3, you have probably identified that the basic structure of the functions follows the management behavior of the `move` functions seen in previous sections.

It is still essential for the function to distinguish between the first iteration of a scale action and subsequent iterations.

As with the move functions seen in Listing 11.1 and Listing 11.2, the actual graphic object must be erased from the screen and the interactive rubber-banding object drawn instead.

The scale action, however, takes on another responsibility as well. Specifically, it maintains a pointer to a temporary `GXLine` structure

```
        static GXLinePtr tmp_data = NULL;
```

for managing when to erase the rubber-banding object, and for ensuring that the transformation is only applied once to the original set of points defining the object.

Consider the body of the `else`, which like the move function's use of `x` and `y` indicates that this is the first iteration for the current scale action, as no `tmp_data` has been created (implicit null test).

```
        } else {
            /* our first time... */
            (*line->erase)( line );
```

In addition to ensuring that the object's method is invoked to erase the object, the body of the `else` creates a new reference to a `GXLine` structure and assigns this value to `tmp_data`.

```
            tmp_data = (GXLinePtr)XtNew(GXLine);
```

EXCURSION

Introducing a New Function for Allocating Memory

The `XtNew` function is provided by the X Toolkit Intrinsics library. It is a function that could have an equivalent in C that looks like

```
    tmp_data = malloc( sizeof(GXLine) );
```

However, this is a good time to introduce it. It is important to remember that any memory allocated by an application must at some point be freed.

The call to free the memory allocated with the `XtNew` function is `XtFree` as you'll see shortly.

The function then assigns to `tmp_data` the number of points it will manage and creates a valid array for storing the same number of points as the object being scaled:

```
            tmp_data->num_pts = line_data->num_pts;
            tmp_data->pts =
                 (XPoint *)XtMalloc( sizeof(XPoint) * tmp_data->num_pts );
```

The `tmp_data` structure stores the original points defining the object. This allows the scale function to apply the transformation to the temporary copy of the points without affecting the original object.

Subsequent calls to the `line_scale` function result in the original points being copied into the `tmp_data` structure and again having the transformation applied to the temporary data structure, leaving the original data unaffected.

```
memcpy( (char *)tmp_data->pts, (char *)line_data->pts,
        sizeof(XPoint) * tmp_data->num_pts );

apply_delta( tmp_data->pts, tmp_data->num_pts,
             FixedX - event->xbutton.x,
             FixedY - event->xbutton.y );
```

The result is that only one transformation is ever applied to the data points in order to minimize the effects of rounding described earlier.

EXCURSION

Adding Functions to Your Memory Management Repertoire

The function `memcpy` is provided by the standard C library, and, as the name implies, serves to copy data from one area of memory to another.

In this context it is used to copy the original points contained in the `line_data->pts` array to the `tmp_data->pts` array.

The syntax of the call is

```
memcpy( destination, source, total size );
```

Because the `memcpy` function is written to transfer data of any type from the source address to the destination address, the structure references are cast to character points (`char *`) to satisfy the function's prototype found in the header

```
#include <stdlib.h>
```

Optionally, a `for` loop could be formed to explicitly copy all the points from one array to another.

Return to the body of the `else` indicating the first iteration of the scale action to examine the function call

```
get_bounds( line_data->pts, line_data->num_pts,
            &OrigX, &OrigY, &ExntX, &ExntY );
```

The variables `OrigX`, `OrigY`, `ExntX`, and `ExntY` are global variables used to define the upper and lower bounds of the object being scaled. Their use will become clear when we examine the `apply_delta` function.

Before discussing `apply_delta` however, let's look at the use of `FixedX` and `FixedY` passed as parameters the `apply_delta` function.

Having reserved the introduction of the cursor management function that invokes the `line_scale`, `line_move`, and `arc_move` functions evaluated in this chapter, accept for the moment that the value of `FixedX` and `FixedY` are assigned the location of the object handle at the start of the scale action.

Unlike the move function, which applied an incremental delta to the points of the object, the scale functions apply the total delta with each iteration.

This is necessary because the original object data points are copied into the `tmp_data` structure with each call to the `line_scale` function. The goal is to affect the points a total of one time by the scale operation. So the total delta from the start of the operation is applied to the original data points, instead of an incremental delta being applied to previously scaled points.

To calculate the overall delta from the start of the operation, the variables `FixedX` and `FixedY` record the location of the `ButtonDown` event that started the scale action. The values of `FixedX` and `FixedY` are then subtracted from the current event point to determine the distance the mouse cursor has moved since the start of the operation.

Now that you understand how the parameters to `apply_delta` are formed and the purpose they serve, consider Listing 11.4, which introduces the `apply_delta` function declaration.

Listing 11.4 The `apply_delta` Function

```
void apply_delta( XPoint *data, int num, int dx, int dy )
{

  switch( GxActiveHandle ) {
  case 0:
    apply_delta_top_left( data, num, dx, dy );
    break;
  case 1:
    apply_delta_top_side( data, num, dx, dy );
    break;
  case 2:
    apply_delta_top_right( data, num, dx, dy );
    break;
  case 3:
    apply_delta_right_side( data, num, dx, dy );
    break;
  case 4:
    apply_delta_bottom_right( data, num, dx, dy );
    break;
  case 5:
    apply_delta_bottom_side( data, num, dx, dy );
    break;
  case 6:
    apply_delta_bottom_left( data, num, dx, dy );
    break;
  case 7:
    apply_delta_left_side( data, num, dx, dy );
    break;
  default:
    setStatus( "LINE: The end is nigh!" );
  }
}
```

The `apply_delta` function invokes the correct `apply_delta_<direction>` based on the current `GxActiveHandle` value. The `GxActiveHandle` is set when the `FixedX` and `FixedY` values are assigned at the start of the scale action.

Listing 11.5 introduces the `apply_delta_<direction>` functions.

Listing 11.5 The `apply_delta_<direction>` Functions

```
static void
apply_delta_bottom_side( XPoint *pts, int num_pts, int dx, int dy )
{
  int i;
  if( ExntY == 0 ) return;

  for( i = 0; i < num_pts; i++ ) {

    pts[i].y -= (int)( dy *
                       ((float)(pts[i].y - OrigY) / (float)ExntY));
  }
}
static void
apply_delta_right_side( XPoint *pts, int num_pts, int dx, int dy )
{
  int i;
  if( ExntX == 0 ) return;

  for( i = 0; i < num_pts; i++ ) {

    pts[i].x -= (int)( dx *
                       ((float)(pts[i].x - OrigX) / (float)ExntX));
  }
}
static void
apply_delta_top_side( XPoint *pts, int num_pts, int dx, int dy )
{
  int i;
  if( OrigY == 0 ) return;

  for( i = 0; i < num_pts; i++ ) {
    pts[i].y +=
      (int)( dy * ((float)(pts[i].y - ExntY) / (float)OrigY));
  }
}
```

continues

Listing 11.5 Continued

```
static void
apply_delta_left_side( XPoint *pts, int num_pts, int dx, int dy )
{
  int i;

  if( OrigX == 0 ) return;

  for( i = 0; i < num_pts; i++ ) {
    pts[i].x +=
      (int)( dx * ((float)(pts[i].x - ExntX) / (float)OrigX));
  }
}
static void
apply_delta_bottom_right( XPoint *pts, int num_pts, int dx, int dy )
{
  apply_delta_right_side( pts, num_pts, dx, dy );
  apply_delta_bottom_side( pts, num_pts, dx, dy );
}
static void
apply_delta_bottom_left( XPoint *pts, int num_pts, int dx, int dy )
{
  apply_delta_bottom_side( pts, num_pts, dx, dy );
  apply_delta_left_side( pts, num_pts, dx, dy );
}
static void
apply_delta_top_right( XPoint *pts, int num_pts, int dx, int dy )
{
  apply_delta_top_side( pts, num_pts, dx, dy );
  apply_delta_right_side( pts, num_pts, dx, dy );
}
static void
apply_delta_top_left( XPoint *pts, int num_pts, int dx, int dy )
{
  apply_delta_top_side( pts, num_pts, dx, dy );
  apply_delta_left_side( pts, num_pts, dx, dy );
}
```

Only four directions require calculations when applying the scale delta to a line object. They are `apply_delta_right_side`, `apply_delta_left_side`, `apply_delta_top_side`, and `apply_delta_bottom_side`.

The functions shown in Listing 11.5 that combine directions are straightforward and require no explanation.

Consider instead the function

```
apply_delta_bottom_side( XPoint *pts, int num_pts, int dx, int dy )
```

After declaring an integer for looping,

```
    int i;
```

the function tests the `ExntY` value to ensure that it is non-zero. This prevents a fatal divide by zero, as you'll see shortly.

```
    if( ExntY == 0 ) return;
```

The function continues by looping through all the points in the points array `pts`. Remember, this array was `tmp_data->pts` as passed by the calling function, meaning the values being altered are a copy of the original points.

```
      pts[i].y -= (int)( dy *
                          ((float)(pts[i].y - OrigY) / (float)ExntY));
```

Because the function's purpose is to apply only the scale delta to the bottom of the object, only the y values are affected.

The variables `dx` and `dy` were passed from the calling function as the result of the evaluation of

```
FixedX - event->xbutton.x
```

and

```
FixedY - event->xbutton.y
```

To determine the proportion of the `dy` value to apply to each data point, the ratio of the point's distance from the origin of the object and the extent of the object is used.

Note

The variable `ExntY` (globally set in the calling function `line_scale`) is the y value of the lower-right corner of the object. Similarly, `OrigY` is the upper-left corner of the object.

The effect of this calculation is that when the bottom handle (or either bottom corner handle) is used to control the scale action, the farther from the origin of the object the data point lies the greater the portion of the delta applied to that data point.

The same ratio is used in the other *side functions* differing only in which component of the delta is applied as dictated by the direction of the scale action. In other words, the left and right side actions only apply the x delta (`dx`) to the x component of the data points and the top, like the bottom, scale direction only affects the y value by some portion of the y delta (`dy`).

Note

The reference to side functions for applying the scale delta is meant to generically refer to any of the `apply_scale_<direction>` functions, where *direction* is `top_side`, `bottom_side`, `left_side`, and `right_side`.

A final difference is that the bottom scale direction deducts the calculated portion of the delta, and the top scale direction adds the portion that is determined by the ratio of the point's distance from the extent relative to the origin. Effectively, the top scale direction inverts the bottom scale operation as is necessary for scaling in the opposite direction. For the same reason, the left and right scale operations are inverted.

Having successfully applied the delta to the sides dictated by the active scale handle, the `line_scale` function draws the rubber-banding object to the canvas window, as seen in Listing 11.3, lines 29–30:

```
29:        XDrawLines( XtDisplay(GxDrawArea), XtWindow(GxDrawArea), rubberGC,
30:                    tmp_data->pts, tmp_data->num_pts, CoordModeOrigin );
```

At some point the `line_scale` function will end as indicated by the passing of a null event point. When this happens, the second `else` statement is evaluated

```
31:     } else {
```

If there is a valid `tmp_data` structure determined by an implicit test for null

```
32:       if( tmp_data ) {
```

the contents of the temporary storage structure's `XPoint` array `pts` are transferred to the line object's data `pts` array

```
33:         memcpy( (char *)line_data->pts, (char *)tmp_data->pts,
34:                 sizeof(XPoint) * line_data->num_pts );
```

Note

Remember that the points contained in the temporary structure were only transformed once. The points transferred to the line object are the points gained from the last iteration of the `line_scale` function.

Because the `line_scale` function is ending, the `tmp_data` reference is no longer needed. Therefore, it is appropriate to free the memory allocated for its use.

```
36:         XtFree((char *)tmp_data->pts);
37:         XtFree((char *)tmp_data);
```

Note

When freeing *nested pointers* such as the pts field contained within the allocated tmp_data structure, you must free them in the reverse order that they were allocated.

This is necessary because after a pointer is freed it can no longer be legally referenced.

In others words, it would be a dangerous (and potentially fatal) mistake to free tmp_data and then reference it to free the pts field.

When freed, the value referenced by the tmp_data variable is no longer valid. Therefore set its value back to NULL in preparation for a subsequent scale action request.

```
39:        tmp_data = NULL;
```

Look now at the arc equivalent to the line_scale function shown in Listing 11.3.

Scaling an Arc

The arc_scale function shown in Listing 11.6 is nearly identical in its structure to the line_scale function seen in Listing 11.3.

Listing 11.6 The arc_scale Function

```
static void arc_scale( GXObjPtr arc, XEvent *event )
{
  static XArc *tmp_data = NULL;

  if( tmp_data ) {
    /*
     * erase the rubberband arc
     */
   XDrawArc( XtDisplay(GxDrawArea), XtWindow(GxDrawArea), rubberGC,
              tmp_data->x, tmp_data->y,
              tmp_data->width, tmp_data->height,
              tmp_data->angle1, tmp_data->angle2 );

  } else {
    /*
     * our first time through - erase the actual arc
     */
    (*arc->erase)( arc );

    tmp_data = (XArc *)XtNew( XArc );
  }

  if( event ) {
    memcpy( (char *)tmp_data, (char *)arc->data, sizeof(XArc));
```

continues

Listing 11.6 Continued

```
    apply_delta( tmp_data,
                 FixedX - event->xbutton.x,
                 FixedY - event->xbutton.y );

    /*
     * draw a rubberband arc
     */
    XDrawArc( XtDisplay(GxDrawArea), XtWindow(GxDrawArea), rubberGC,
              tmp_data->x, tmp_data->y,
              tmp_data->width, tmp_data->height,
              tmp_data->angle1, tmp_data->angle2 );

  } else {
    if( tmp_data ) {
       memcpy( (char *)arc->data, (char *)tmp_data,
              sizeof(XArc));

      XtFree( (char *)tmp_data );
      tmp_data = NULL;
    }
  }
}
```

Comfortable with the structure of this function based on the study of the `line_scale` function, note the changes in data references to account for the different object type.

Listing 11.7 shows the `apply_delta` function for applying the scale delta to the arc object. It, too, is nearly identical in structure to the `apply_delta` function used by the `line_scale` function.

EXCURSION

A Review of the Graphics Editor Project Structure

As demonstrated in the Graphics Editor program structure seen in Chapter 6, “Components of an X Window Application,” there is total separation of source code for line, arc, and text object routines.

This, as indicated in Appendix B, “Application Layout Code Listing,” is accomplished by having multiple program files. In this way the code controlling line objects is placed in a different file than the code for arcs.

Further, through use of the keyword `static` within the files containing the source code for the various objects, the function names can be reused for routines of similar purpose.

This is the case with the `apply_delta` function. As is made clear in Chapter 13, “Application Structure,” the resolution of the varying `apply_delta` functions is determined by the version of the function co-located in the file containing the source code for the different objects.

Listing 11.7 The `apply_delta` Function for Arcs

```
static void apply_delta( XArc *data, int dx, int dy )
{
  switch( GxActiveHandle ) {
  case 0:
    apply_delta_top_left( data, dx, dy );
    break;
  case 1:
    apply_delta_top_side( data, dx, dy );
    break;
  case 2:
    apply_delta_top_right( data, dx, dy );
    break;
  case 3:
    apply_delta_right_side( data, dx, dy );
    break;
  case 4:
    apply_delta_bottom_right( data, dx, dy );
    break;
  case 5:
    apply_delta_bottom_side( data, dx, dy );
    break;
  case 6:
    apply_delta_bottom_left( data, dx, dy );
    break;
  case 7:
    apply_delta_left_side( data, dx, dy );
    break;
  default:
    /* -shouldn't- happen in my time */
    setStatus( "ARC: The end is nigh!" );
  }
}
```

Though similar in structure to the `apply_delta` function employed by the `line_scale` function, some subtle differences do exist, specifically, the parameter lists for `apply_delta_<direction>` functions.

Listing 11.8 shows the definition of these functions for use by the arc object.

Listing 11.8 The Arc `apply_delta_<direction>` Functions

```
static void apply_delta_top_side( XArc *data, int dx, int dy )
{
  int y1, y2;

  y1 = data->y;
  y2 = data->y + data->height;

  data->y = min( y1 - dy, y2 + dy );
  data->height = max( y1 - dy, y2 + dy ) - data->y;
```

continues

Listing 11.8 Continued

```
}
static void apply_delta_right_side( XArc *data, int dx, int dy )
{
  int x1, x2;

  x1 = data->x;
  x2 = data->x + data->width;

  data->x = min( x1 + dx, x2 - dx );
  data->width = max(x1 + dx, x2 - dx) - data->x;
}
static void apply_delta_top_right( XArc *data, int dx, int dy )
{
  apply_delta_right_side( data, dx, dy );
  apply_delta_top_side( data, dx, dy );
}
static void apply_delta_bottom_side( XArc *data, int dx, int dy )
{
  int y1, y2;

  y1 = data->y;
  y2 = data->y + data->height;

  data->y = min( y1 + dy, y2 - dy );
  data->height = max(y1 + dy, y2 - dy) - data->y;
}
static void apply_delta_bottom_right( XArc *data, int dx, int dy )
{
  apply_delta_bottom_side( data, dx, dy );
  apply_delta_right_side( data, dx, dy );
}
static void apply_delta_left_side( XArc *data, int dx, int dy )
{
  int x1, x2;

  x1 = data->x;
  x2 = data->x + data->width;

  data->x = min( x1 - dx, x2 + dx );
  data->width = max( x1 - dx, x2 + dx ) - data->x;
}
static void apply_delta_bottom_left( XArc *data, int dx, int dy )
{
  apply_delta_bottom_side( data, dx, dy );
  apply_delta_left_side( data, dx, dy );
}
static void apply_delta_top_left( XArc *data, int dx, int dy )
{
  apply_delta_top_side( data, dx, dy );
  apply_delta_left_side( data, dx, dy );
}
```

As with line scaling, only the side functions actually perform calculations. The corner functions simply invoke the correct combination of side routines to accomplish their task.

Consider the function

```
static void apply_delta_top_side( XArc *data, int dx, int dy )
```

for applying the scale delta to the top side of the arc object. Evaluating it will shed light on how all the arc side functions work because the relationships follow those specified with the line side functions

```
    y1 = data->y;
    y2 = data->y + data->height;

    data->y = min( y1 - dy, y2 + dy );
    data->height = max( y1 - dy, y2 + dy ) - data->y;
```

By storing the upper and lower corner `y` values, the `apply_delta_top_side` function can determine the new values for the y and height field of the `XArc` structure.

The logic applied here is that the minimum point between the upper-right value *less* the delta and the lower-right value *plus* the delta will be the new y value for the arc.

This *minimum* evaluation of the points is important for determining when the user has, for instance, selected the handle in the lower-right corner and then dragged it to a value less than the upper-right corner of the object.

In other words, if the scale action *flips* the arc object, the `apply_delta_`*`<direction>`* routine must account for this, and not allow the y value to be less than the y + height value.

A look at the remaining *side* functions shows that the logic is the same, changing only the components affected and the direction in which the delta is applied. For instance, left and right side functions change only x and width values, where the top and bottom affect only the y and height values.

As you advance in the development of the Graphics Editor project and apply the concepts introduced in this chapter, the ideas will become clearer.

The last transformation to consider before doing so, however, is rotation.

Rotating

As the name implies, the rotation transformation revolves objects around a center point. For the purpose of the context editor, this center point will coincide with the center of the object as is demonstrated in the next section.

Rotating a Line

Rotating a point-based object such as a line is most simply done by rotating each individually.

Figure 11.3 shows the relationship of a point to a 90-degree angle. By representing a point in this way, the dissection of the right angle allows a relationship to be derived for performing the rotation.

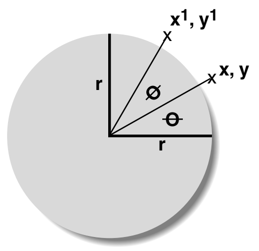


Figure 11.3
The rotation of a point.

The point (x, y) in Figure 11.3 is the point prior to rotation and point (x^1, y^1) is the point after rotation. To move from (x, y) to (x^1, y^1), you must apply the definition of the sine and cosine of an angle.

As you might recall from early trigonometry study, sine is defined as a function that maps an angle to the y coordinate of the angle's intersection with the unit circle. Similarly, cosine is the function that maps an angle to the x coordinate of the intersection.

Applying these definitions, the points x and y can be represented in terms of the angle > and radius r as

$$x = r \cos >$$
$$y = r \sin >$$

Therefore, to determine the values of x^1 and y^1 you only need to add the angle = to > as seen below.

$$x^1 = r \cos (> + =)$$
$$y^1 = r \sin (> + =)$$

Solving for x^1

$$r \cos (> + =) = r \cos > \cos = + r \sin > \sin =$$

Substituting the value of x for (r cos >) and y for (r sin >) yields

x^1 = x cos = - y sin =

Solving for y^1 in the same way

r sin (> + =) = r sin > cos = + r cos > sin =

or

y^1 = y cos = + x sin =

There are only a couple more issues to resolve before coding the solution for rotating a point.

First, the proof shown for calculating x^1 and y^1 assumes that the angle’s origin is at (0, 0). Second, as demonstrated in Figure 11.3, the point (x, y) does not sit exactly on the perimeter of the circle.

Both can be resolved by applying a principle introduced in Chapter 10, “Trigonometric and Geometric Functions.” All points are a constant distance from the foci of a circle. Therefore, the point before rotation must be transformed to an origin of (0, 0) and the point after rotation must be transformed to a point equal to the magnitude of the original point’s distance from the circle the point is perceived to sit on.

➔ See Chapter 10, page 218, for a discussion of **point and arc intersection**.

Listing 11.9 demonstrates the implementation of rotating a point through an angle.

Listing 11.9 The `rotate_point` Function

```
#define gxround(a)       (int)(((a)<0.0)?(a)-.5:(a)+.5)
void rotate_point( XPoint *pt, int deg, double cx, double cy )
{
  double dx, dy, theta, cosa, sina, mag;

  dx = (double)pt->x - cx;
  dy = cy - (double)pt->y;

  if (dx == 0.0 && dy == 0.0) {
    return;
  }

  if( deg == 0 ) return;

  theta = gx_compute_angle( dx, dy );
  theta -= (deg * M_PI / 180.0 );

```

continues

Listing 11.9 Continued

```
19:     if (theta < 0.0) {
20:       theta += M_2PI;
21:     } else if (theta >= M_2PI - 0.001) {
22:       theta -= M_2PI;
23:     }
24:
25:     mag = sqrt(dx * dx + dy * dy);
26:
27:     cosa = mag * cos(theta);
28:     sina = mag * sin(theta);
29:
30:     pt->x = gxround(cx + cosa);
31:     pt->y = gxround(cy - sina);
32:   }
33:   #undef gxround
```

Listing 11.9 begins by defining a macro for rounding a number

```
1:    #define gxround(a)        (int)(((a)<0.0)?(a)-.5:(a)+.5)
```

The function itself expects that a pointer to the point being rotated be passed as the first parameter, the degrees that the point should be rotated as the second, and lastly the coordinates of the center point of the object containing the point

```
2:    void rotate_point( XPoint *pt, int deg, double cx, double cy )
```

The variables employed by the function are of the data type `double`

```
4:      double dx, dy, theta, cosa, sina, mag;
```

➔ As you might recall from Chapter 2, "Programming Constructs," section "Data Types," page 70, the data type `double` is significantly more precise than `float`.

Precision during rotation is a serious issue. Consider the quantity of floating point operations required to accomplish the rotation of a point. Further, moving the floating point results to an integer variable (whole pixel representation) inherently causes further loss of precision. The use of the data type `double` aids in minimizing loss of precision during the rotation calculations.

Next, the `rotate_point` function must calculate the foci in the same way it was done in Chapter 10.

```
6:      dx = (double)pt->x - cx;
7:      dy = cy - (double)pt->y;
```

The angle > is then calculated and converted to radians:

```
16:     theta = gx_compute_angle( dx, dy );
17:     theta -= (deg * M_PI / 180.0 );
```

The function gx_compute_angle applies the definition of sine and cosine by working backward to determine the angle based on the intersection points. The gx_compute_angle is introduced later.

Bounds checking is performed on the angle theta to ensure that it is in the proper range for the rotation operation.

```
if (theta < 0.0) {
  theta += M_2PI;
} else if (theta >= M_2PI - 0.001) {
  theta -= M_2PI;
}
```

The magnitude of the foci is calculated so that it can be applied to the new point resulting from the rotation

```
mag = sqrt(dx * dx + dy * dy);
```

The actual rotation of the point is calculated

```
cosa = mag * cos(theta);
sina = mag * sin(theta);
```

Finally, the value of the rotation point is assigned to the XPoint structure pointer, replacing the previous values of the point with the rotated values.

```
pt->x = gxround(cx + cosa);
pt->y = gxround(cy - sina);
```

Notice that in conjunction with the assignment of the rotated point to the XPoint structure, the point is transformed relative to the center coordinate of the object containing the data point. Also, the gxround macro is employed to ensure that the values are rounded up to nearest whole pixel representation.

I show the management function that invokes the rotate_point for point-array–based objects shortly, but first consider the concept of rotating arcs.

Rotating an Arc

Because rotating a circle of 360 degrees has no meaning, the issue of rotating an arc is limited to ellipses and arcs of less than 360 degrees.

Rotating an Ellipse

As a caveat at the end of Chapter 10, the X Window System is incapable of representing an ellipse with a major and minor axis that is not parallel with the x and y axes.

Therefore, the rotation of an ellipse is not supported by the Graphics Editor projects.

Rotating Arcs Less than 360 Degrees

The X Window System inherently supports the rotation of an arc by management of the values of `angle1` and `angle2` provided in the `XArc` structure.

➔ If necessary, review the definitions of these fields as introduced with the `XDrawArc` primitive in Chapter 7, section “XDrawArc,” page 192.

Next Steps

The next chapter will introduce *coordinate systems* and illustrate the need to understand the system in which graphic transformations are performed. This is not important to the success of the Graphics Editor project, but will provide clarity to the issue of graphic transformations.

Demonstrated with the use of the X Window System primitives and X Window creation in Chapters 7 and 8, the origin is consistently in the upper-left corner of the screen. However, when the mathematics for rotating a point is discussed, the origin relative the operation was silently made the lower-left corner (refer to Figure 11.3).

Determining the origin for an operation or calculation is a factor of the coordinate system being employed.

Following the brief discussion on coordinate systems, the foundation will be complete, and full attention will focus on furthering the Graphics Editor.

Chapter 12

In this chapter

- *Rectangular Coordinates*
- *Polar Coordinate System*
- *Next Steps*

Coordinate Systems

The concept of coordinate systems must be addressed to make a book on the X Window System and graphic transformations complete.

This chapter introduces coordinate systems and programmatic considerations for each of them.

As the last building block in the foundation preparing us to focus solely on programming the Graphics Editor, this chapter marks the end of Section One.

A *coordinate system* determines the positive and negative directions of the x and y axes and their relationship to horizontal and vertical placement. Further, a coordinate system dictates the location of the origin or (0,0) position within the representation.

In computer graphics, a primary concern is in locating points on the surface of the canvas or drawing area. To specify a location requires knowledge of the coordinate system in use.

In a windowing environment, coordinates within the system are *discrete*, meaning that they represent the pixels used to position the elements bearing the coordinates.

Every `Drawable` (`Window` or `Pixmap`) in the X Window environment maintains its own coordinate system or management of element placement within its bounds.

The coordinate system employed by the `Drawables` in the X Window environment has its origin in the upper-left corner of the Drawable with the x-axis increasing to the right and the y-axis increasing downward, as depicted in Figure 12.1.

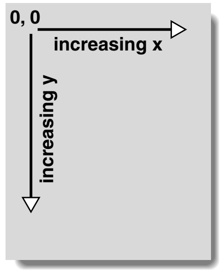


Figure 12.1

The coordinate system used in the X Window environment.

Within the X Window System's coordinate system, negative numbers can be used to represent coordinates; however, negative coordinates affect visibility.

The following sections introduce two common coordinate systems employed in computer graphics and windowing environments.

Rectangular Coordinates

A *rectangular coordinate system* has horizontal and vertical axes that are right angles to each other. The coordinates within the rectangular coordinate system are evenly spaced in both the x and y directions. The origin of the rectangular coordinate system is at the junction of the two axes, as shown in Figure 12.2.

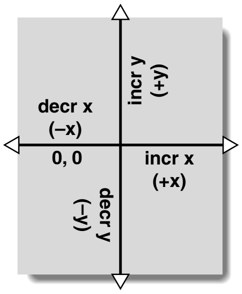


Figure 12.2

The rectangular coordinate system.

The rectangular coordinate system is also known as the *Cartesian coordinate system*. It is commonly used in digital graphics systems because of its easy adaptation to raster displays.

➔ Review Chapter 8, section "Raster Graphics," page 199, for a review of **vector versus raster graphics**.

Polar Coordinate System

The *polar coordinate system* is rarely used in graphics, because it is less valuable than the rectangular coordinate system for mapping pixels to a display.

A polar coordinate system defines angles and distances rather than *scalar* values.

geek speak

A *scalar value* in this context refers to an integer data type that is scaled in a linear manner.

The coordinate points in a rectangular coordinate system are known as scalar values, whereas the polar coordinate system uses a combination of angles and distances to indicate placement in system.

In the polar coordinate system, the position of a point (coordinate) is defined as the angular deflection of a line.

The endpoints of the line are formed from the origin and the specified location of the point.

To represent a location in the polar coordinate system, points are represented in terms of an angle called *theta* (θ) and a radius (the length of the line) referred to as *rho* (ρ).

Figure 12.3 illustrates the relationship between theta and rho within the polar coordinate system

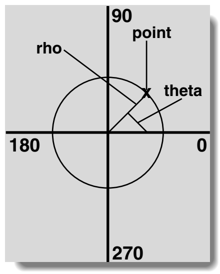


Figure 12.3

The polar coordinate system.

Applying what you now understand about coordinate systems, it should be obvious why both coordinate systems are useful to software engineers.

The placement of windows and the drawing of primitives in the X Window System environment uses a coordinate system very similar to and obeying many of the same properties as the rectangular coordinate system.

However, accomplishing graphic transformation such as rotation is most easily done using the polar coordinate system.

Related to the employment of a coordinate system, software engineers also must be aware of measuring conventions used within the system.

This issue will present itself again when adding a print driver to the Graphics Editor application because the representation of coordinates on a screen or monitor is affected by units of measure and scale factors when transferring the coordinates to paper using the PostScript language.

Next Steps

This chapter concludes Section One, "Starting Points," and the foundation I thought imperative before approaching the Graphics Editor Project.

As you begin the Section Two, "Graphics Editor Application," you should be adequately prepared, either directly by skipping over the Starting Points or indirectly by spending time with the introduction the initial chapters provide.

Graphics Editor Application

This section focuses strictly on developing the Graphics Editor project.

With a solid understanding of the concepts necessary for accomplishing C and X Window System programming provided in Section One, I pay less attention to the details of the language and environment and more attention to the structure and execution of the Graphics Editor project.

Where to Begin

Unlike Section One, this section does not afford the flexibility to start anywhere except at the beginning. Each chapter in this section builds on the previous one.

As the project advances, code samples reflect only the changes to functions or files previously introduced, when possible to do so without sacrificing clarity.

What's at the End

At the completion of Section Two, you will have a functional, two-dimensional graphical editor capable of drawing, moving, scaling, rotating, and altering the attributes of a variety of graphic objects.

Part IV

Laying Out the Parts

Chapter 13

In this chapter

Application Structure

This chapter launches the Graphics Editor project by structuring the main program file and creating the graphical user interface.

Before proceeding, it is important that, independent of the knowledge and experience you bring to the project, you have reviewed the necessary sections in Part I to prepare adequately for the information presented here.

Project structure and organizations are critical elements of software design. As the Graphics Editor project progresses, it will grow in complexity very quickly.

The goal of this chapter is to lay out the application structure, including graphical user interface, parsing the command line, and finally, configuring the Canvas for receiving and processing the X events that will control user actions.

Because much of the source code for structuring the application is introduced in Chapter 6, focus here is given mainly to new ideas.

Note

As new features are added to code listings already introduced, I will show *incremental listings*. An incremental listing generally shows only the line before and after the lines being inserted.

When starting a new application, you should first consider how the code will be structured based on elements present in the project. Second, you should consider how the application will be structured.

Project Structure

Structuring the code addresses the issue of where source and header files will reside relative to each other. Addressed as well is the placement of the object and executable files.

Note

The directory structure for the Graphics Editor was introduced in Figure 1.11.

If you haven't reviewed `make` file syntax (discussed in Chapter 1, "UNIX for Developers," in the section "Makefile") and how to structure the project correctly, as discussed in the "make.defines" section of Chapter 1, you should do so now.

The layout and project management presented in Chapter 1 is intended for multiple-platform support. In other words, through use of the GNU `make` utility and directory layout introduced in Chapter 1, the project can be built successfully using any version of the UNIX operating system.

Table 13.1 shows the organization of the application source code by describing the contents of each of the files contained in the project and showing a relative path for their placement.

Table 13.1 Graphics Editor Project Layout

File	*Purpose*
`src/GNUmakefile`	Configuration file for GNU `gmake` utility. Defines source files, virtual paths, and build options for project management.
`make.defines`	Included by the `GNUmakefile` to set flags, paths, and variables required by the `GNUmakefile`.
`src/gxMain.c`	Defines the program entry point, initializes a connection to the X Server, and manages control of the graphical user interface.
`src/gxGraphics.c`	Holds functions invoked from `gxMain` for creating the user interface and defines and assigns global variables used during program execution.
`src/gxGx.c`	Functions used for controlling graphic objects created by the editor and managing events for the widgets of the graphical user interface.
`src/include/gxGraphics.h`	Declares macros, global variables, and structures used by all source files.
`src/include/gxIcons.h`	Contains all bitmaps used as icons for the various buttons of the graphical user interface.
`src/include/gxProto.h`	Contains prototypes for all functions called externally from the file in which they are declared.
`src/gxLine.c`	Contains `GXLine` object creation and method definitions.

`src/gxArc.c`	Contains `GXArc` object creation and method definitions.
`src/gxText.c`	Contains `GXText` object creation and method definitions.
`src/include/vfonts`	Contains files necessary for accomplishing the vector fonts used by the `GXText` object.

The descriptions provided in Table 13.1 will serve as a guide for how the source code of the Graphics Editor project is structured.

Note

Because the files introduced in Table 13.1 are relative, it is expected that a project root directory (value of `GxHome` variable in the `make.defines`) be decided before implementing the structure to support the introduction of the files in the project.

Let us focus now on the structure of the application by deciding how best to lay out the graphical user interface.

geek speak

Making the decision of *what goes where* on a graphical user interface requires understanding the features that the application will support.

Attention to the feature list will enable you to determine whether the required number of icons can be placed on the interface or whether their number requires a menu system or some management method to prevent the application from becoming cluttered.

Table 13.2 shows the feature list that the graphical user interface of the Graphics Editor must support.

Table 13.2 Graphics Editor Feature List

Creation Functions	
	Latex Line Object
	Pencil Line Object
	Arc Object
	Box Object
	Arrow Object
	Text Object
Management Functions	
	Copy
	Cut

continues

Table 13.2 continued

Management Functions	
	Print
	Export
	Save
	Load
Assigning Attributes	
	Foreground color
	Background color
	Fill Style
	Line Width
	Degrees of Rotation
Miscellaneous	
	Exit

Great care must be given to the production of *any* graphical user interface to ensure that elements of aesthetics and intuitiveness are well balanced.

Intuitiveness

If features are hidden from the users or not logical in their placement within the user interface, users will probably not use them much. This has an obvious and direct impact on the impression of quality assigned to the application.

Aesthetics

The aesthetics of the interface is as important as creating an interface that is intuitive in its use. The attractiveness of the design has equal impact on the impression of quality.

Addressing the issues of creating an intuitive and aesthetically pleasing interface based on the features that must be supported by the application, you can begin to sketch either mentally or literally what the interface should look like.

Figure 13.1 shows my mental image of what the interface should look like.

Note

Notice in Figure 13.1 the incredible clarity of my thoughts.

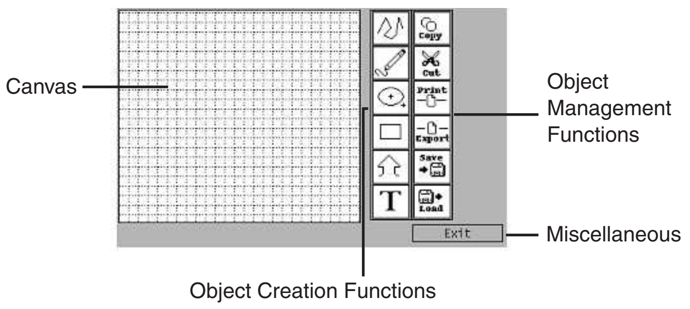


Figure 13.1

The Graphics Editor interface.

The only feature listed in Table 13.2 not addressed in the proposed interface shown in Figure 13.1 is a mechanism for changing the attributes of objects.

To address this requirement, a menu listing the possible attributes will be presented to the user when she right-clicks over an active object.

This approach does not adhere to the requirement of being initially intuitive because the user will not know that the menu exists until instructed of its presence. The solution does offer two things, however. First is the chance to demonstrate programmatically how to create pop-up and cascading menus and second is a tidy workspace that is not littered with icons that are not general to the entire application.

The following section shows how to create the interface presented in Figure 13.1.

Laying Out the User Interface

The first file to create for the Graphics Editor is the `gxMain.c` file, which controls the entry point and creation of the interface components. This file is the easiest to create for the project because the functions it invokes are external to the file.

Listing 13.1 shows the contents of the `gxMain.c` file.

Listing 13.1 The `gxMain.c` File

```
/**
 ** 2D Graphical Editor (2d-gx)
 **
 ** gxMain.c
  */
#include <stdlib.h>
#include <stdio.h>

#include <X11/Intrinsic.h>   /* for creation routines */
#include <X11/StringDefs.h>  /* resource names        */
#include <X11/Xaw/Form.h>

#include "gxGraphics.h"
#include "gxProtos.h"
```

continues

Listing 13.1 continued

```
15:
16:    /*
17:     * Entry point for the application
18:     */
19:    int main( int argc, char **argv )
20:    {
21:      XtAppContext appContext;
22:      Widget       toplevel;
23:      Widget       form;
24:      Widget       canvas;
25:
26:      toplevel = XtVaAppInitialize( &appContext, "2D Graphical Editor",
27:                                    NULL,0, &argc, argv, NULL,
28:                                    NULL );
29:
30:      form = XtVaCreateWidget( "topForm",
31:                               formWidgetClass, toplevel,
32:                               NULL );
33:
34:      canvas = create_canvas( form );
35:      create_status( form, canvas );
36:      create_buttons( form );
37:
38:      XtManageChild( canvas );
39:      XtManageChild( form );
40:
41:      XtRealizeWidget( toplevel );
43:      XtAppMainLoop( appContext );
44:
45:      exit(0);
46:    }
47:
48:    /**
49:     ** end of gxMain.c
50:     */
```

Listing 13.1 begins, following some comments, by including header files from various sources.

The first two come from the C environment and the delimiters < and > surrounding the filename provide a hint for where the C preprocessor should look for them:

```
6:     #include <stdlib.h>
7:     #include <stdio.h>
```

These header files provide most of the commonly used built-in functions provided by the C language and are generally included at the beginning of all C source files.

The next few included header files are supplied by the X Window System development environment:

```
#include <X11/Intrinsic.h>   /* for creation routines */
#include <X11/StringDefs.h>  /* resource names        */
#include <X11/Xaw/Form.h>
```

As the comments following the preprocessor `include` directives indicate, `Intrinsic.h` contains the prototypes for the X initialization and widget creation routines used by the functions in this file. The `StringDefs.h` file resolves the resource names used to configure the attributes of the widgets created and, finally, the `Form.h` header file contains the form widget class definition, and therefore must be included before instantiating a widget of type `formWidgetClass`.

EXCURSION

Including a Path Component with Header Files

Header files often contain path components in addition to the actual filename as seen with the three header files included from the X environment.

The convention is to specify only to the compiler the *home* directory of header locations when using the compiler flag `-I`. From the home or root component the compiler is instructed to search, subdirectories are nested with the header filename.

For example, the compiler flag `-I` used to compile the file shown in Listing 13.1 would include the path component

```
-I /usr/include
```

for the Linux operating system or

```
-I /usr/openwin/include
```

under the Sun Solaris operating system.

Then the remaining path components needed by the preprocessor for finding the file are embedded in the `include` directive as seen with `X11/Intrinsic.h`, `X11/StringDefs.h`, and `X11/Xaw/Form.h`.

In other words, the actual path under Linux for finding the file `Form.h` is `/usr/include/X11/Xaw`. A portion of this is provided to the compiler by merit of the `-I` flag, and the remainder is nested with the header filename.

Notice that the header files provided by X, like those provided by C, are delimited with < and > symbols. The following two header files, however, use double quotes (" ") to contain the filenames, indicating to the preprocessor that the files are local.

```
#include "gxGraphics.h"
#include "gxProtos.h"
```

Although the C preprocessor will find the files regardless of how the filenames are delimited, the search order is determined by these delimiters and therefore affects the speed with which the files are located.

Another advantage of correctly delimiting the filenames is to ensure that anyone who follows behind you will know where to begin looking as well.

The header file `gxProtos.h`, however delimited, is included to ensure that the functions invoked in this file but defined elsewhere in the project are correctly prototyped.

As you scan further through Listing 13.1 you will identify `create_canvas`, `create_status`, and `create_buttons` as functions unique to the Graphics Editor project. The forward declaration for these is contained in `gxProtos.h` and required before the compiler can successfully check whether the parameter types and return values are consistent with the usage. This level of error checking done by the compiler prevents many semantic and syntactical errors introduced by the programmer.

Because many of the functions prototyped in `gxProtos.h` require references to structures defined by the project, the inclusion of `gxGraphics.h` provides the necessary type definitions to satisfy the compiler. This implies that the order the files are included is important as well, as the compiler must have the type definitions before they are valid for use.

Listing 13.2 shows the contents of the `gxGraphics.h` file.

Listing 13.2 The `gxGraphics.h` File

```
/**
 ** 2D Graphical Editor (2d-gx)
 **
 ** gxGraphics.h
 */
#include <X11/Intrinsic.h>

#ifndef _GX_GRAPHICS_H_INC_
#define _GX_GRAPHICS_H_INC_

#endif /* _GX_GRAPHICS_H_INC_ */

#ifndef GLOBAL
#define GLOBAL
#else
#undef  GLOBAL
#define GLOBAL extern
#endif

GLOBAL Widget GxStatusBar;
GLOBAL Widget GxDrawArea;

/**
 ** end of gxGraphics.h
 */
```

In its current state, `gxGraphics.h` is a very simple file. As the Graphics Editor project grows in complexity, so will the `gxGraphics.h` header file.

After ensuring that the `Intrinsic.h` header file has been included to resolve the `Widget` data type used later in the file, `gxGraphics.h` checks for the existence of an environment directive `_GX_GRAPHICS_H_INC_`. If the variable has not been defined, `gxGraphics.h` defines it to ensure that a portion of the file is included only once:

```
8:     #ifndef _GX_GRAPHICS_H_INC_
9:     #define _GX_GRAPHICS_H_INC_
```

This mechanism for preventing multiple inclusions of the same file is common in C syntax when new data types or variables are created because multiple inclusions would redefine the types or variables defined the first time the file was included.

Although `gxGraphics.h` is in its fledgling state, the syntax is included for future expansion. Line 11 of Listing 13.2 closes the implicit body of the `ifndef` directive:

```
11:    #endif /* _GX_GRAPHICS_H_INC_ */
```

A similar syntax is seen with the environment variable `GLOBAL`:

```
13:    #ifndef GLOBAL
14:    #define GLOBAL
```

Not meant to prevent multiple inclusions, the test of the presence of `GLOBAL` indicates whether this is the first inclusion of the file. If so, it *declares* the variables prefaced with the `GLOBAL` variable as in the first iteration. `GLOBAL` is defined to be *nothing*:

```
20:    GLOBAL Widget GxStatusBar;
21:    GLOBAL Widget GxDrawArea;
```

Subsequent inclusions of the header file and the test for the presence of `GLOBAL` result in `GLOBAL` being redefined to the keyword `extern`.

This is a powerful mechanism that enables a variable to first be declared and subsequently externed. Effectively, the first file to include this header is responsible for declaring all the global variables used by the project. Further, inclusions simply inform the compiler that the variable has already been declared for use.

Listing 13.3 shows the contents of the `gxProtos.h` file in its current state. This file also grows in complexity as more and more functions are shared throughout the project.

Listing 13.3 The `gxProtos.h` File

```
1:     /**
2:      ** 2D Graphical Editor (2d-gx)
3:      **
4:      ** gxProtos.h
5:      */
```

continues

Listing 13.3 continued

```
#include <X11/Intrinsic.h>
#include "gxIcons.h"

#ifndef EXTERN
#define EXTERN
#else
#undef  EXTERN
#define EXTERN extern
#endif

/*
 * Creations routines in gxGraphics.c
 */
EXTERN Widget create_canvas ( Widget );
EXTERN void   create_status ( Widget, Widget );
EXTERN void   create_buttons( Widget );
EXTERN void   drawAreaEventProc( Widget, XtPointer, XEvent *, Boolean );

/*
 * Drawing functions used in GxIcons.h
 */
EXTERN void gx_line( void );
EXTERN void gx_pencil( void );
EXTERN void gx_arc( void );
EXTERN void gx_box( void );
EXTERN void gx_arrow( void );
EXTERN void gx_text( void );

/*
 * Control functions used in GxIcons.h
 */
EXTERN void gx_copy( void );
EXTERN void gx_delete( void );
EXTERN void gx_select( void );
EXTERN void gx_save( void );
EXTERN void gx_load( void );
EXTERN void gx_export( void );
EXTERN void gx_exit( Widget, XtPointer, XtPointer );

/*
 * Utilities in gxGx.c
 */
EXTERN void setStatus( char * );
EXTERN void draw_manager( Widget, XtPointer, XtPointer );
EXTERN void ctrl_manager( Widget, XtPointer, XtPointer );

/**
 ** end of gxProtos.h
 */
```

Notice that the use of `EXTERN` follows the same form and serves the same purpose as the environment directive `GLOBAL` used in `gxGraphics.h`. However, a different name is used here to avoid conflicts in the compiler environment.

Many functions here have not been introduced yet. Focus for the moment on only the lines

```
19:    EXTERN Widget create_canvas ( Widget );
20:    EXTERN void   create_status ( Widget, Widget );
21:    EXTERN void   create_buttons( Widget );
```

which are pertinent to our current analysis of Listing 13.1.

The `gxIcons.h` file and remaining prototypes will be introduced shortly as you begin to create the buttons and fill them with their respective icons.

The prototypes that are found on lines 19–20 satisfy function calls made from the `gxMain.c` file and justify the inclusion of the `gxProtos.h` header file.

Continuing the discussion of `gxMain.c` with lines 26–28

```
26:      toplevel = XtVaAppInitialize( &appContext, "2D Graphical Editor",
27:                                    NULL,0, &argc, argv, NULL,
28:                                    NULL );
```

a call to `XtVaAppInitialize` is made to establish a connection to the X server, initialize the X Resource Database for the application, and fill the application context.

➔ Review Chapter 6, "Components of an X Window Application," see the section "Connecting to the X Server," page 144, for a description of the components in an X Window application and use of the `XtVaAppInitialize` function.

To the `toplevel` widget returned from the initialization of the X environment, function `main` in `gxMain.c` adds a form widget as its sole child:

```
30:      form = XtVaCreateWidget( "topForm",
31:                               formWidgetClass, toplevel,
32:                               NULL );
```

To the form widget are added a canvas area, a status bar, and the object management and control buttons:

```
34:      canvas = create_canvas( form );
35:      create_status( form, canvas );
36:      create_buttons( form );
```

Prototypes for these functions are found in `gxProtos.h`; however, the actual function definitions are in the file `gxGraphics.c` as seen in Listing 13.4.

Listing 13.4 The `gxGraphics.c` File

```
/**
 ** 2D Graphical Editor (2d-gx)
 **
 ** gxGraphics.c
 */
#include <stdio.h>

#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <X11/Xaw/Form.h>
#include <X11/Xaw/Command.h>
#include <X11/Xaw/Box.h>

#include "gxGraphics.h"
#include "gxIcons.h"

/*
 * Create the region of the application where we will draw
 */
Widget create_canvas( Widget parent )
{
  GxDrawArea = XtVaCreateWidget( "drawingArea",
                                 formWidgetClass, parent,
                                 XtNbackground,
                                   WhitePixelOfScreen(XtScreen(parent)),
                                 XtNtop,            XawChainTop,
                                 XtNleft,           XawChainLeft,
                                 XtNbottom,         XawChainBottom,
                                 XtNright,          XawChainRight,
                                 XtNheight,         220,
                                 XtNwidth,          250,
                                 NULL );

  XtAddEventHandler(GxDrawArea, PointerMotionMask, False,
                    (XtEventHandler)drawAreaEventProc, (XtPointer)NULL);
  XtAddEventHandler(GxDrawArea, ButtonPressMask|ButtonReleaseMask,False,
                    (XtEventHandler)drawAreaEventProc, (XtPointer)NULL);

  return GxDrawArea;
}

/*
 * create_status
 */
void create_status( Widget parent, Widget fvert )
{
  GxStatusBar = XtVaCreateManagedWidget( "statusBar",
                                         labelWidgetClass, parent,
                                         XtNtop,          XawChainBottom,
                                         XtNleft,         XawChainLeft,
                                         XtNbottom,      XawChainBottom,
                                         XtNright,       XawChainRight,
                                         XtNfromVert,    fvert,
                                         XtNborderWidth,0,
                                         NULL );
  setStatus( "2D-GX (c)Starry Knight Software - Ready..." );
}

/*
 * statusProc
 */
void statusProc( Widget w, XtPointer msg, XEvent *xe, Boolean flag )
{
  if( msg != NULL )
    setStatus( msg );
  else
    setStatus( "\0" );
}

/*
 *  create_icons
 */
void create_icons( Widget parent, GxIconData *iconData,
                   void (*callback)( Widget, XtPointer, XtPointer ))
{
  Widget btn;
  Pixmap pix;

  while( iconData->info != NULL ) {
    if( iconData->info->bits != NULL ) {
      pix = create_pixmap( parent, iconData->info );

      btn = XtVaCreateManagedWidget( "",
                                     commandWidgetClass, parent,
                                     XtNwidth,  iconData->info->w + 1,
                                     XtNheight, iconData->info->h + 1,
                                     XtNbackgroundPixmap,     pix,
                                     XtNhighlightThickness,   1,
                                     NULL );

      XtAddEventHandler( btn, EnterWindowMask, False,
                        (XtEventHandler)statusProc,
                        ➥(XtPointer)iconData->mesg);
      XtAddEventHandler( btn, LeaveWindowMask, False,
                        (XtEventHandler)statusProc, (XtPointer)NULL );

      XtAddCallback( btn, XtNcallback, callback,
        ➥(XtPointer)iconData->func );
    }
    /*
    * go to the next element
     */
```

continues

Listing 13.4 continued

```
    iconData++;
  }
}

/*
 * Create a panel of buttons that will allow control of the application
 */
void create_buttons( Widget parent )
{
  Widget butnPanel, exitB;

  /*
   * create a panel for the drawing icons
   */
  butnPanel = XtVaCreateWidget( "drawButnPanel",
                                 boxWidgetClass, parent,
                                 XtNtop,           XawChainTop,
                                 XtNright,         XawChainRight,
                                 XtNbottom,        XawChainTop,
                                 XtNleft,          XawChainRight,
                                 XtNhorizDistance, 10,
                                 XtNfromHoriz,     GxDrawArea,
                                 XtNhSpace,        1,
                                 XtNvSpace,        1,
                                 NULL );

  create_icons( butnPanel, gxDrawIcons, draw_manager );
  XtManageChild( butnPanel );

  /*
   * create a panel for the control icons
   */
  butnPanel = XtVaCreateWidget( "ctrlButnPanel",
                                boxWidgetClass, parent,
                                XtNtop,       XawChainTop,
                                XtNright,     XawChainRight,
                                XtNbottom,    XawChainTop,
                                XtNleft,      XawChainRight,
                                XtNfromHoriz, butnPanel,
                                XtNhorizDistance,0,
                                XtNhSpace,        1,
                                XtNvSpace,        1,
                                NULL );

  create_icons( butnPanel, gxCntrlIcons, ctrl_manager );
  XtManageChild( butnPanel );

  exitB = XtVaCreateManagedWidget( "    Exit    ",
                                   commandWidgetClass, parent,
                                   XtNtop,      XawChainBottom,
151:                                     XtNbottom,    XawChainBottom,
152:                                     XtNleft,      XawChainRight,
153:                                     XtNright,     XawChainRight,
154:                                     XtNfromVert,  butnPanel,
155:                                     XtNfromHoriz, GxStatusBar,
156:                                     NULL );
157:
158:    XtAddCallback( exitB, XtNcallback, gx_exit, NULL );
159:  }
160:
161:  /*
162:   * create_pixmap
163:   */
164:  Pixmap create_pixmap( Widget w, XbmDataPtr data )
165:  {
166:   return( XCreatePixmapFromBitmapData(XtDisplay(w),
167:                                       DefaultRootWindow(XtDisplay(w)),
168:                                       (char *)data->bits,
169:                                       data->w, data->h,
170:                                       BlackPixelOfScreen(XtScreen(w)),
171:                                       WhitePixelOfScreen(XtScreen(w)),
172:                                       DefaultDepthOfScreen(XtScreen(w))));
173: }
174:
175: /**
176:  ** end of gxGraphics.c
177:  */
```

Lines 8–12 of Listing 13.4 issue the `include` compiler directive for inclusion of the header files required to resolve variable, structure, and prototype definitions external to the `gxGraphics.c` file.

The first function defined in `gxGraphics.c` shown in Listing 13.4 is the `create_canvas` routine, and at lines 17–40.

The `create_canvas` function accepts one parameter for specifying the parent of the form widget it creates to act as the canvas area. Notice that the variable storing the value returned by the creation function is the global variable `GxDrawArea`, which is found in `gxGraphics.h`.

The placement of the canvas area created by `create_canvas` is specified as having its edges chained to the corresponding edges of the parent. Clearly, the parent is expected to be of the class `formWidgetClass` because relative positioning has no meaning with manager widgets of other classes.

➔ In Chapter 5, the section “The Form Widget,” page 136, demonstrates the use of composite resources for specifying relative placement of the children it manages.

Finally, the `create_canvas` function registers for the receipt of X events that would not normally be sent to a form widget.

By use of the X Toolkit Intrinsic call `XtAddEventHandler`, the `create_canvas` function can request that the function `drawAreaEventProc` is called when `PointerMotion` events are received. The prototype for `drawAreaEventProc` is found in `gxProtos.h`, and its definition will be introduced shortly.

Although the variable `GxDrawArea` is global to the project, its value is returned for use by the calling function `main` found in `gxMain.c`. Even though `main` could refer to the value by the global reference, the `return` is included to demonstrate its use.

Continuing in the review of `gxGraphics.c`, consider lines 42–57, which define the function `create_status`.

The `create_status` function creates a label widget for use as place to display context-sensitive help messages. Assignment of the help message is seen with the definition of the object drawing and management buttons. Optionally, messages can be explicitly displayed by invoking the `setStatus` function that will be introduced shortly.

Notice in the placement of the label widget (lines 49–54) that the resource `XtNfromVert` used to indicate the vertical placement of the widget is relative to the value of the second parameter passed to the function. Looking back at Listing 13.1 at the invocation of the `create_status` function reveals that the second parameter is the `GxDrawArea` widget.

Lines 59–68 of the file `gxGraphics.c` shown in Listing 13.4 define the function `statusProc` that is used to process the event handler assigned to buttons created by the function `create_icons`.

The function `statusProc` is responsible for updating the `GxStatusBar` label widget to reflect the context-sensitive help message for any button that gains focus.

The `create_icons` function is located at lines 70–97 in Listing 13.4 and is responsible for parsing the `GxIconData` reference passed as the second parameter from the function `create_buttons` function.

The definition of the `GxIconData` structure pointer dictates everything about the buttons being added to the interface.

Listing 13.5 introduces the `gxIcon.h` header file containing the `GxIconData` references, bitmap icons, callback functions, and context-sensitive help strings used to define the icons created for the application.

Listing 13.5 The `gxIcons.h` File

```
/**
 ** 2D Graphical Editor (2d-gx)
 **
 ** gxIcons.h
 */
#ifndef _GX_ICONS_H_INC_
#define _GX_ICONS_H_INC_

#include "gxProtos.h"

 /*
  * Storage for pertinent XBM data
  */
 typedef  struct _xbm_data {
   unsigned char *bits;
   int           w, h;
 } XbmData, *XbmDataPtr;

 /*
  * IconData necessary to create icon
  */
 typedef struct _gx_icon_data {
   XbmDataPtr info;

   void (*func)(void);
   char *mesg;
 } GxIconData, *GxIconDataPtr;

 #define icon_static( name, bits, width, height ) \
  static XbmData name = { bits, width, height }

 /*
  * drawing icons
  */
 static unsigned char line_bits[] = {
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x20,0x00,0x04,0x00,0x00,0x30,0x00,
    0x0c,0x00,0x00,0x50,0x00,0x0c,0x00,0x00,0x48,0x00,0x14,
    0x00,0x00,0x88,0x00,0x14,0x00,0x00,0x84,0x00,0x14,0x00,
    0x00,0x04,0x01,0x22,0x00,0x00,0x02,0x01,0x22,0x00,0x00,
    0x02,0x02,0x22,0x00,0x00,0x01,0x02,0x42,0x00,0x00,0x01,
    0x04,0x42,0x00,0x80,0x00,0x04,0x42,0x00,0x80,0x00,0x02,
    0x01,0x00,0x40,0x00,0x01,0x01,0x00,0x40,0x80,0x00,0x01,
    0x00,0x20,0x40,0x00,0x01,0x00,0x20,0x20,0x00,0x01,0x00,
    0x00,0x10,0x00,0x01,0x00,0x00,0x08,0x80,0x00,0x00,0x00,
    0x30,0x40,0x00,0x00,0x00,0xc0,0x20,0x00,0x00,0x00,0x00,
    0x13,0x00,0x00,0x00,0x00,0x0c,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00};
 icon_static( line_icon, line_bits, 36, 32 );
```

continues

Listing 13.5 continued

```

static unsigned char pencil_bits[] = {
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0xc0,0x00,0x00,0x00,0x00,0xe0,0x01,0x00,0x00,0x00,0xd0,
   0x03,0x00,0x00,0x00,0x88,0x03,0x00,0x00,0x00,0x14,0x01,0x00,
   0x00,0x00,0xa6,0x00,0x00,0x00,0x00,0x49,0x00,0x00,0x00,0x80,
   0x30,0x00,0x00,0x00,0x40,0x10,0x00,0x00,0x00,0x20,0x08,0x00,
   0x00,0x00,0x10,0x04,0x00,0x00,0x00,0x08,0x02,0x00,0x00,0x00,
   0x04,0x01,0x00,0x00,0x00,0x82,0x00,0x00,0x00,0x00,0x41,0x00,
   0x00,0x00,0x80,0x20,0x00,0x00,0x00,0x40,0x10,0x00,0x00,0x00,
   0xa0,0x08,0x00,0x00,0x00,0x10,0x05,0x00,0x00,0x00,0x10,0x02,
   0x00,0x00,0x00,0x30,0x01,0x00,0x00,0x28,0xf0,0x00,0x00,0x00,
   0x44,0x10,0x00,0x00,0x00,0x84,0x20,0x00,0x00,0x00,0x04,0x41,
   0x00,0x00,0x00,0x08,0x42,0x00,0x00,0x00,0x10,0x44,0x00,0x00,
   0x00,0x20,0x38,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00};
icon_static( pen_icon, pencil_bits, 36, 32 );

static unsigned char arc_bits[] = {
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0x3f,0x00,0x00,0x00,
   0x70,0xc0,0x01,0x00,0x00,0x0c,0x00,0x06,0x00,0x00,0x02,0x00,
   0x08,0x00,0x00,0x01,0x00,0x10,0x00,0x80,0x00,0x00,0x20,0x00,
   0x40,0x00,0x00,0x40,0x00,0x40,0x00,0x04,0x40,0x00,0x20,0x00,
   0x04,0x80,0x00,0x20,0x00,0x1f,0x80,0x00,0x20,0x00,0x04,0x80,
   0x00,0x40,0x00,0x04,0x40,0x00,0x40,0x00,0x00,0x40,0x00,0x80,
   0x00,0x00,0x20,0x00,0x00,0x01,0x00,0x10,0x00,0x00,0x02,0x00,
   0x08,0x00,0x00,0x0c,0x00,0x86,0x00,0x00,0x70,0xc0,0x81,0x00,
   0x00,0x80,0x3f,0xe0,0x03,0x00,0x00,0x00,0x80,0x00,0x00,0x00,
   0x00,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00};
icon_static( arc_icon, arc_bits, 36, 32 );

static unsigned char box_bits[] = {
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,0x1f,
   0x00,0x80,0x00,0x00,0x10,0x00,0x80,0x00,0x00,0x10,0x00,
   0x80,0x00,0x00,0x10,0x00,0x80,0x00,0x00,0x10,0x00,0x80,
   0x00,0x00,0x10,0x00,0x80,0x00,0x00,0x10,0x00,0x80,0x00,
   0x00,0x10,0x00,0x80,0x00,0x00,0x10,0x00,0x80,0x00,0x00,
   0x10,0x00,0x80,0x00,0x00,0x10,0x00,0x80,0x00,0x00,0x10,
   0x00,0x80,0x00,0x00,0x10,0x00,0x80,0x00,0x00,0x10,0x00,
   0x80,0x00,0x00,0x10,0x00,0x80,0xff,0xff,0x1f,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00};
         icon_static( box_icon, box_bits, 36, 32 );

         static unsigned char arrow_bits[] = {
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x00,
    0x00,0x00,0x80,0x02,0x00,0x00,0x00,0x40,0x04,0x00,0x00,0x00,
    0x20,0x08,0x00,0x00,0x00,0x10,0x10,0x00,0x00,0x00,0x08,0x20,
    0x00,0x00,0x00,0x04,0x40,0x00,0x00,0x00,0x02,0x80,0x00,0x00,
    0x00,0x01,0x00,0x01,0x00,0x80,0x00,0x00,0x02,0x00,0x40,0x00,
    0x00,0x04,0x00,0x20,0x00,0x00,0x08,0x00,0xe0,0x07,0xc0,0x0f,
    0x00,0x00,0x04,0x40,0x00,0x00,0x00,0x04,0x40,0x00,0x00,0x00,
    0x04,0x40,0x00,0x00,0x00,0x04,0x40,0x00,0x00,0x00,0x04,0x40,
    0x00,0x00,0x00,0x04,0x40,0x00,0x00,0x00,0x04,0x40,0x00,0x00,
    0x00,0x02,0x80,0x00,0x00,0xc0,0x01,0x00,0x07,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00};
         icon_static( arr_icon, arrow_bits, 36, 32 );

         static unsigned char text_bits[] = {
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,
    0x1f,0x00,0x80,0x83,0x1f,0x1c,0x00,0x80,0x01,0x0f,0x18,
    0x00,0x80,0x00,0x0f,0x10,0x00,0x00,0x00,0x0f,0x00,0x00,
    0x00,0x00,0x0f,0x00,0x00,0x00,0x00,0x0f,0x00,0x00,0x00,
    0x00,0x0f,0x00,0x00,0x00,0x00,0x0f,0x00,0x00,0x00,0x00,
    0x0f,0x00,0x00,0x00,0x00,0x0f,0x00,0x00,0x00,0x00,0x0f,
    0x00,0x00,0x00,0x00,0x0f,0x00,0x00,0x00,0x00,0x0f,0x00,
    0x00,0x00,0x00,0x0f,0x00,0x00,0x00,0x00,0x0f,0x00,0x00,
    0x00,0x00,0x0f,0x00,0x00,0x00,0x00,0x0f,0x00,0x00,0x00,
    0x80,0x1f,0x00,0x00,0x00,0xe0,0x7f,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00};
         icon_static( text_icon, text_bits, 36, 32 );

         /*
          * control icons
          */
         static unsigned char    copy_bits[] = {
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
    0x00,0x00,0x7c,0x00,0x00,0x00,0x00,0x82,0x00,0x00,0x00,0x00,
    0x01,0x01,0x00,0x00,0x80,0x00,0x00,0x00,0x00,0x80,0x80,0x0f,
    0x00,0x00,0x80,0x40,0x10,0x00,0x00,0x80,0x20,0x20,0x00,0x00,
    0x80,0x10,0x40,0x00,0x00,0x00,0x11,0x40,0x00,0x00,0x00,0x12,
    0x40,0x00,0x00,0x00,0x14,0x40,0x00,0x00,0x00,0x10,0x40,0x00,
    0x00,0x00,0x20,0x20,0x00,0x00,0x00,0x40,0x10,0x00,0x00,0x00,
    0x80,0x0f,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xc0,0x03,0x00,
```

continues

Listing 13.5 continued

```
   0x00,0x00,0x60,0x73,0xdf,0x1d,0x00,0x60,0xd8,0xb6,0x0d,0x00,
   0x60,0xd8,0xb6,0x0d,0x00,0x60,0xdb,0x36,0x05,0x00,0xc0,0x71,
   0x1e,0x07,0x00,0x00,0x00,0x06,0x03,0x00,0x00,0x00,0xcf,0x03,
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00};
icon_static( copy_icon, copy_bits, 36, 32 );

static unsigned char    delete_bits[] = {
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x03,0x00,0x03,
   0x00,0x00,0x05,0x80,0x02,0x00,0x00,0x09,0x40,0x02,0x00,0x00,
   0x11,0x20,0x01,0x00,0x00,0x22,0x90,0x00,0x00,0x00,0x44,0x48,
   0x00,0x00,0x00,0x88,0x25,0x00,0x00,0x00,0x10,0x1f,0x00,0x00,
   0x00,0xe0,0x0f,0x00,0x00,0x00,0xc0,0x0e,0x00,0x00,0x00,0xfc,
   0xfe,0x01,0x00,0x00,0xfe,0x9f,0x01,0x00,0x00,0x73,0x1e,0x03,
   0x00,0x00,0x33,0x18,0x03,0x00,0x00,0x1f,0x78,0x03,0x00,0x00,
   0x0e,0xf0,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xc0,0x00,0x00,
   0x00,0x3c,0xc0,0x00,0x00,0x00,0xf6,0xed,0x03,0x00,0x00,0x86,
   0xcd,0x00,0x00,0x00,0x86,0xcd,0x00,0x00,0x00,0xb6,0xcd,0x06,
   0x00,0x00,0x1c,0x9f,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00};
icon_static( delete_icon, delete_bits, 36, 32 );

qstatic unsigned char select_bits[] = {
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0xf0,0xff,0x07,0x00,0x00,0x10,0x00,0x04,0x00,0x00,0x10,0xf0,
   0x05,0x00,0x00,0x10,0x40,0x04,0x00,0x00,0x10,0x40,0x04,0x6c,
   0xdb,0x10,0x40,0x04,0x00,0x00,0x10,0x40,0x04,0x04,0x7c,0x91,
   0x47,0x04,0x04,0x10,0x51,0x08,0x04,0x00,0x10,0x50,0x08,0x04,
   0x04,0x10,0x91,0x07,0x04,0x04,0x10,0x11,0x00,0x04,0xe0,0x11,
   0xf0,0xff,0x07,0x14,0x02,0x01,0x00,0x00,0x14,0x02,0x01,0x00,
   0x00,0xe0,0x01,0x00,0x00,0x00,0x04,0x00,0x01,0x00,0x00,0xb4,
   0x6d,0x01,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0xe0,0x81,0x01,0x20,0x00,0x30,0x81,0x01,0x30,0x00,0x70,0xb8,
   0x39,0x77,0x00,0xe0,0xac,0xad,0x35,0x00,0xc0,0xbd,0xbd,0x31,
   0x00,0x90,0x8d,0x8d,0x35,0x00,0xf0,0xb8,0x39,0x67,0x00,0x00,
   0x00,0x00,0x00,0x00};
icon_static( select_icon, select_bits, 36, 32 );


static unsigned char save_bits[] = {
   0x80,0x07,0x00,0x00,0x00,0xc0,0xe4,0x6e,0x0e,0x00,0xc0,0xb3,
   0x6d,0x1b,0x00,0x00,0xe7,0x39,0x1f,0x00,0x40,0xb6,0x39,0x03,
   0x00,0xc0,0xf3,0x13,0x1e,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00,0xfc,0x1f,0x00,0x00,0x00,0x06,
   0x32,0x00,0x00,0x00,0x05,0x55,0x00,0x00,0x00,0x05,0xd2,0x00,
   0x00,0x06,0xf9,0xcf,0x00,0x00,0x0e,0x01,0xc0,0x00,0xe0,0x1f,
   0x01,0xc0,0x00,0xe0,0x3f,0x01,0xc0,0x00,0xe0,0x1f,0x01,0xc0,
   0x00,0x00,0x0e,0xf1,0xc7,0x00,0x00,0x06,0x15,0xd4,0x00,0x00,
   0x00,0xd5,0xd5,0x00,0x00,0x00,0x11,0xc4,0x00,0x00,0x00,0xfe,
   0xff,0x00,0x00,0x00,0xfc,0x7f,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x38,0x0c,0x07,0x00,0xe0,0x31,0x00,0x06,0x00,0x30,0x31,
   0x8e,0xe7,0x00,0xf0,0x30,0xcc,0xb6,0x01,0xc0,0x31,0xcc,0xf6,
   0x01,0x90,0x31,0xcc,0x36,0x00,0xf0,0xfc,0xbf,0xef,0x01,0x00,
   0x00,0x00,0x00,0x00};
        icon_static( save_icon, save_bits, 36, 32 );

        static unsigned char load_bits[] = {
   0x00,0x00,0x00,0x38,0x00,0xc0,0x03,0x00,0x30,0x00,0x80,0xc1,
   0x71,0x3c,0x00,0x80,0x61,0xdb,0x36,0x00,0x80,0x61,0xf3,0x36,
   0x00,0x80,0x6d,0xdb,0x36,0x00,0xc0,0xcf,0xf9,0x7d,0x00,0x00,
   0x00,0x00,0x00,0x00,0xc0,0xff,0x01,0x00,0x00,0x60,0x20,0x03,
   0x00,0x00,0x50,0x50,0x05,0x00,0x00,0x50,0x20,0x0d,0x00,0x00,
   0x90,0xff,0x0c,0x03,0x00,0x10,0x00,0x0c,0x07,0x00,0x10,0x00,
   0xcc,0x0f,0x00,0x10,0x00,0xcc,0x1f,0x00,0x10,0x00,0xcc,0x0f,
   0x00,0x10,0x7f,0x0c,0x07,0x00,0x50,0x41,0x0d,0x03,0x00,0x50,
   0x5d,0x0d,0x00,0x00,0x10,0x41,0x0c,0x00,0x00,0xe0,0xff,0x0f,
   0x00,0x00,0xc0,0xff,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x38,0x0c,0x07,0x00,0xe0,0x31,0x00,0x06,0x00,0x30,0x31,
   0x8e,0xe7,0x00,0xf0,0x30,0xcc,0xb6,0x01,0xc0,0x31,0xcc,0xf6,
   0x01,0x90,0x31,0xcc,0x36,0x00,0xf0,0xfc,0xbf,0xef,0x01,0x00,
   0x00,0x00,0x00,0x00};
        icon_static( load_icon, load_bits, 36, 32 );

        static unsigned char export_bits[] = {
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x06,0x30,0x00,0x7c,0x00,
   0x00,0x30,0x00,0xd8,0x76,0xb7,0xf9,0x00,0xd8,0x5c,0x66,0x33,
   0x00,0x78,0x0c,0x66,0x33,0x00,0x18,0x0c,0x66,0xb3,0x01,0x3c,
   0x9e,0x6f,0xe3,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xe0,0x03,
   0x00,0x00,0x00,0x10,0x06,0x00,0x00,0x00,0x10,0x0a,0x00,0x00,
   0x00,0x10,0x1e,0x00,0x00,0x00,0x10,0x10,0x00,0x00,0x00,0x10,
   0x10,0x00,0x00,0xfc,0x17,0xd0,0xff,0x00,0x00,0x10,0x10,0x00,
   0x00,0x00,0x10,0x10,0x00,0x00,0x00,0x10,0x10,0x00,0x00,0x00,
   0x10,0x10,0x00,0x00,0x00,0x10,0x10,0x00,0x00,0x00,0xe0,0x0f,
   0x00,0x00,0x00,0x00,0x00,0x80,0x01,0x1f,0x00,0x00,0x80,0x01,
   0xd3,0xfd,0x38,0xfb,0x07,0x8f,0xb7,0x6d,0xae,0x01,0x03,0xb3,
   0x6d,0x86,0x01,0x9b,0xb7,0x6d,0x86,0x0d,0xdf,0xf6,0x38,0x0f,
   0x07,0x00,0x30,0x00,0x00,0x00,0x00,0x78,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00};
        icon_static(export_icon, export_bits, 36, 32 );

        static GxIconData gxDrawIcons[] = {
        { &line_icon, gx_line,   "Draw an elastic line..."        },
        { &pen_icon,  gx_pencil, "Draw a freestyle line..."       },
        { &arc_icon,  gx_arc,    "Draw a circle..."               },
        { &box_icon,  gx_box,    "Draw a square or rectangle..." },
        { &arr_icon,  gx_arrow,  "Draw an arrow..."               },
        { &text_icon, gx_text,   "Draw dynamic text..."           },
        /*-----------------------------------*/
        /* this list MUST be NULL terminated */
```

continues

Listing 13.5 continued

```
256:    /*-----------------------------------*/
257:    { NULL },
258:    };
259:
260:    static GxIconData gxCntrlIcons[] = {
262:    { &copy_icon,   gx_copy,   "Copy selected object..."              },
263:    { &delete_icon, gx_delete, "Delete selected object..."            },
264:    { &select_icon, gx_select, "Select multiple objects..."           },
265:    { &save_icon,   gx_save,   "Save current drawing..."              },
266:    { &load_icon,   gx_load,   "Load saved drawing..."                },
267:    { &export_icon, gx_export, "Save drawing as GIF or PostScript..." },
268:    /*-----------------------------------*/
269:    /* this list MUST be NULL terminated */
270:    /*-----------------------------------*/
271:    { NULL },
272:    };
273:
274:    /* prototypes */
275:    extern void   create_icons  ( Widget, GxIconDataPtr,
276:                                  void (*)(Widget, XtPointer, XtPointer ));
277:    extern Pixmap create_pixmap ( Widget, XbmDataPtr    );
278:
279:    #endif /* _GX_ICONS_H_INC_ */
280:    /**
281:     ** end of gxIcons.h
282:     */
```

Following the `ifndef` preprocessor directive to prevent multiple inclusions of this header file and the `include` directive for including `gxProtos.h` for the forward declarations of the `draw` and `management` functions, the `gxIcons.h` file shown in Listing 13.5 defines two important structures for managing icon creation.

Lines 11–17 show the definition for the `XbmData` structure used to store the bit array data, width, and height values for the bitmap created with the X Client `bitmap`.

Lines 19–27 in Listing 13.5 hold the definition for the `GxIconData` structure. The `GxIconData` structure nests a reference to an `XbmData` structure and includes fields for the callback function to be assigned to the icon and the context-sensitive help message, which is displayed when the mouse cursor enters the icon.

Following the structure definitions, the `gxIcons.h` header file shown Listing 13.5 defines a macro for declaring and initializing the `XbmData` references from the bitmap data found in lines 32–245:

```
29:   #define icon_static( name, bits, width, height ) \
30:    static XbmData name = { bits, width, height }
```

The bitmap data is divided into two groups. Lines 32–139 define the object drawing icons and lines 141–245 define the object management icons. The bitmap data used to define the different icons, as mentioned, is created using the X Client bitmap (or equivalent). After the desired icon is created using a utility capable of saving data in XBM format, the contents of the saved file is inserted into the `gxIcons.h` header for visibility in the project.

➔ Chapter 6, in the section “Creating Buttons,” on page 159, discusses in greater detail the generation of bitmaps for use by an application.

After the many icons defined for use in the Graphics Editor user interface, lines 147–258 show the `GxIconData` array definition for `gxDrawIcons`. This array defines the icons, callbacks, and context-sensitive help messages for the object-drawing buttons. Lines 260–272 show a similar definition for the `gxCntrlIcons` used to define the object management buttons.

These arrays are passed independently by `create_buttons` in Listing 13.4 to the `create_icons` function for parsing and conversion into command widgets that will act as holders for the different icons:

```
128:      create_icons( butnPanel, gxDrawIcons, draw_manager );
```

and

```
146:      create_icons( butnPanel, gxCntrlIcons, ctrl_manager );
```

The `create_icons` function defines a loop and is executed as long as there is a valid element of the `GxIconData` pointer `iconData`.

Notice in Listing 13.5 that the `GxIconData` arrays are `NULL` terminated. The `create_icons` function loops until it finds the end of the array as indicated by the presence of the `NULL`:

```
80:      while( iconData->info != NULL ) {
```

As long as there is a valid array element, a `Pixmap` is created from the `XbmData` field of the `GxIconData` structure called `info`:

```
82:          pix = create_pixmap( parent, iconData->info );
```

This `Pixmap` is then used as the background `Pixmap` of the command widget whose parent is the button panel created by `create_buttons` and passed as the first parameter of the `create_icons` function.

Event handlers are added to this button to manage the context-sensitive help string—displayed when the button gains focus (mouse cursor enters button’s window) and to clear the string when the focus is lost.

Added to the button, as well, is a callback function dictated by the type of button that is being created. The buttons from the `gxDrawIcons` array are assigned the `draw_manager` callback function and the `gxCntrlIcons` are given the `cntrl_manager` function.

The draw_manager and cntrl_manager functions are prototyped in gxProtos.h and passed as the third parameter to the create_icons function.

The call data specified to the XtAddCallback function is very important:

```
96:            XtAddCallback( btn, XtNcallback, callback,
               ➥(XtPointer)iconData->func );
```

The func field of the GxIconData corresponds to the specific action that will be invoked within the draw_manager or cntrl_manager functions.

Look again at the declaration of the gxDrawIcons and gxCntrlIcons arrays gxIcons.h shown in Listing 13.4. The second field of the elements being initialized determines the specific action assigned to the icon.

Functions such as gx_line, gx_pencil, gx_arc, and so forth are assigned to the gxDrawIcons array elements. The gxCntrlIcons elements have functions such as gx_copy, gx_delete, and gx_select assigned to them.

All the functions that are found in the GxIconData array elements declared in gxIcons.h are prototyped in gxProtos.h. Their declarations will be introduced shortly as will the draw_manager and cntrl_manager functions.

First, however, we end the analysis of Listing 13.4 by noting the creation of the exit button on lines 148–156:

```
148:      exitB = XtVaCreateManagedWidget( "    Exit    ",
```

This is the first button that has been created to use a label type of string. All the previous command widgets had a background pixmap assigned, which excludes the use of text in the button.

The label assigned to a command button is its instance name by default. Notice that the instance name has spaces nested in it to extend the size of the button to approximately equal the menu button panes created in the create_buttons function.

Although valid syntax, a danger exists in using spaces in widget instance names: the difficulty in including the value in an external resource file such as .Xdefaults.

Returning to Listing 13.1, a few final requirements must be met in the function main defined in gxMain.c.

Specifically, the form and canvas widget created unmanaged are explicitly managed:

```
38:       XtManageChild( canvas );
39:       XtManageChild( form );
```

All X applications require the realization of the toplevel widget: recursively mapping the windows of all its descendents to the screen with a call to XtRealizeWidget:

```
41:       XtRealizeWidget( toplevel );
```

The last step of the function `main` is to enter an infinite loop in which the application will continually monitor its event queue:

```
43:       XtAppMainLoop( appContext );
```

As the X Server sends events to the application, the Graphics Editor will remove them from the queue and dispatch them to the widgets that have registered a request to receive them.

Note

Notice line 45 of the function `main`:

```
45:       exit(0);
```

The presence of the call to `exit` is only to satisfy the GNU C Compiler because without it the compiler would issue the warning

```
Control reaches end of non-void function
```

geek speak

Because the `XtAppMainLoop` function contains an infinite loop, this line of the function will *never* be reached.

Having completed structuring the application, there are only a few more housekeeping issues to discuss.

Listing 13.6 introduces the functions defined in the `gxCntrlIcons` array elements declared in `gxIcons.h`.

Listing 13.6 The `gxGx.c` File

```
/**
 ** 2D Graphical Editor (2d-gx)
 **
 ** gxGx.c
 */
#include <stdio.h>

#include "gxGraphics.h"
#include "gxProtos.h"

 #include <X11/Xaw/Label.h>

 static void (*draw_mgr_func)( XEvent * ) = NULL;

 /*
  * gx_exit
  */
```

continues

Listing 13.6 continued

```
void gx_exit( Widget w, XtPointer cd, XtPointer cbs )
{
  exit(0);
}

/*
 * gx_copy
 */
void gx_copy( void )
{
  printf( "gx_copy\n" );
}

/*
 * gx_delete
 */
void gx_delete( void )
{
  printf( "gx_delete\n" );
}

/*
 * gx_select
 */
void gx_select( void )
{
  printf( "gx_select\n" );
}

/*
 * gx_save
 */
void gx_save( void )
{
  printf( "gx_save\n" );
}

/*
 * gx_load
 */
void gx_load( void )
{
  printf( "gx_load\n" );
}

/*
 * gx_export
 */
void gx_export( void )
{
  printf( "gx_export\n" );
}

/*
 * setStatus
 */
void setStatus( char *message )
{
  XtVaSetValues( GxStatusBar, XtNlabel, message, NULL );
}

/*
 * draw_manager
 */
void draw_manager( Widget w, XtPointer cd, XtPointer cbs )
{
  void (*draw_func)( XEvent * ) = (void (*)(XEvent *))cd;

  if( draw_func != NULL ) (*draw_func)( NULL );
  draw_mgr_func = draw_func;
}

/*
 * ctrl_manager
 */
void ctrl_manager( Widget w, XtPointer cd, XtPointer cbs )
{
  void (*ctrl_func)( void ) = (void(*)(void))cd;
  if( ctrl_func != NULL ) ctrl_func();
}

/*
 * drawAreaEventProc
 */
void drawAreaEventProc( Widget w, XtPointer cd, XEvent *event,
➥Boolean flag )XEvent *event, Boolean flag )
{
  if( draw_mgr_func != NULL ) (*draw_mgr_func)( event );
}

/**
 ** end of gxGx.c
 */
```

Most of the functions defined in the file `gxGx.c` shown in Listing 13.6 are only stubs at this point, doing little more than printing out a statement that the function has been reached.

The bodies of these functions will evolve over time; however, their presence even in this simple form is absolutely necessary to satisfy the link phase when building this phase of the project.

The functions used in the `gxDrawIcon` array elements defined `gxIcons.h` are not contained in a single file. Instead, a file specific to the graphic object type created by the icon is defined in Listings 13.7 through 13.9.

Listing 13.7 The `gxLine.c` File

```
/**
 ** 2D Graphical Editor (2d-gx)
 **
 ** gxLine.c
 */
#include <stdio.h>
#include "gxGraphics.h"

void gx_line( XEvent *event )
 {
   printf( "draw a line...\n" );
 }

 void gx_pencil( XEvent *event )
 {
   printf( "draw freestlye\n" );
 }

 void gx_arrow( XEvent *event )
 {
   printf( "draw an arrow\n" );
 }

 void gx_box( XEvent *event )
 {
   printf( "draw a box\n" );
 }

 /**
  ** end of gxLine.c
  */
```

The file `gxLine.c` shown in Listing 13.7 controls the creation of all point-based objects.

Listing 13.8 The `gxArc.c` File

```
/**
 ** 2D Graphical Editor (2d-gx)
 **
 ** gxArc.c
 */
#include <stdio.h>

void gx_arc( void )
{
   printf( "draw an arc...\n" );
 }

 /**
  ** end of gxArc.c
  */
```

The `gxArc.c` file shown in Listing 13.8 contains only support for the arc graphic object.

Listing 13.9 The `gxText.c` File

```
/**
 ** 2D Graphical Editor (2d-gx)
 **
 ** gxText.c
 */
#include <stdio.h>

void gx_text( void )
{
   printf( "draw text...\n" );
 }

 /**
  ** end of gxText.c
  */
```

The `gxText.c` file shown in Listing 13.9 contains only source code for supporting the vector text or *Hershey font* graphic object.

The functions shown to satisfy the references assigned to the drawing icons in the `gxDrawIcons` array are, like most of the functions in `gxGx.c` shown in Listing 13.6, only stubs at this point in the project. These files will grow significantly in complexity during the next several chapters as the actual objects are defined, controlled, saved, and restored.

The next section is a diversion from advancing the project in that there are currently no command-line parameters supported by the application. However, I would be negligent to not demonstrate ways to add flags and parameters to the Graphics Editor program.

Parsing the Command Line

Providing information to an application is often crucial for proper use of the program. Because the application is responsible for extracting any information specified by the user when the program was invoked, this section demonstrates two methods of parsing the command line.

The C programming language provides a function called getopt. Similar to getopt, but significantly more robust, is the function supplied by the X Toolkit Intrinsics called XtVaGetApplicationResources.

The getopt Function

The getopt function is a built-in function provided by the C language and is prototyped in the header file stdlib.h included with most C source code files.

The getopt function is dependent upon several external variables and is destructive to the command line; that is, the command line can be parsed only one time using the getopt function.

A prototype for the getopt function follows:

```
int getopt( int argc, char *argv, const char *optstring );
```

The getopt function returns the next option letter found in argv that matches a letter in the optstring value.

The definition of optstring made by the programmer is done to specify the option letters used by the application and therefore enable the getopt function to recognize them.

Note

The letters placed in the definition of the optstring value are synonymous with what has been referred to as the flags passed to an application.

In forming the value for optstring, if a colon follows a letter, the option is expected to have an associated argument.

The external variables used by getopt must be declared to the file actually employing the getopt function. These variables include the following:

```
extern char *optarg;
extern int optind, opterr, optopt;
```

The getopt function places the index of the next argument to be processed in the optind variable. The optind is external and initialized to 1 before the first call to getopt.

When all options have been processed, meaning nothing remains on the command line, getopt returns the constant EOF.

If flags (letters) are specified on the command line that are not contained in the optstring value, getopt prints an error message to the standard error (stderr) and returns a question mark. However, if the external variable `opterr` is set to `0` before invoking the `getopt` function, error messages are disabled and `getopt` will not report unidentified options found on the command line.

Listing 13.10 shows a code fragment for how the `getopt` function can be used to process command-line arguments.

Listing 13.10 The `getopt` Function

```
#include <stdlib.h>
#include <stdio.h>
main (int argc, char **argv)
{
   int c;

   extern char *optarg;
   extern int optind;

   int errCnt    = 0;
   int errCnt    = 0;
   int verbose   = 0;
   char *infile  = NULL;
   char *outfile = NULL;

   while ((c = getopt(argc, argv, "vi:o:")) != EOF) {
     switch (c) {
       case 'v':
         verbose = 1;
         break;
       case 'i':
         infile = optarg;
         printf("infile = %s\n", infile);
         break;
       case 'o':
         outfile = optarg;
         printf("outfile = %s\n", outfile);
         break;
       case '?':
         errCnt++;
     }
   }
   if( errCnt > 0 ) {
     fprintf( stderr,
             "usage: %s -v, -i <input filename> -o <output filename>\n",
             argv[0] );
     exit( 2 );
   }
   exit( 0 );
 }
```

The flags identified by the sample code shown in Listing 13.10 are `v`, `i`, and `o`. Because the options `i` and `o` are followed by a colon (:) in the `optstring` value, it is expected that these flags will have an associated argument, which the `getopt` functions assigns to `optarg`:

```
           case 'i':
             infile = optarg;
```

A caveat in the use of the `getopt` function is that `getopt` does not enforce the inclusion of arguments for flags that require them.

For example, if the code fragment above was invoked without the user specifying a value for the `-i` flag, `getopt` would use the next item placed on the command line, which potentially would be the `-o` flag.

A more resilient approach to parsing the command line is afforded Intrinsics-based applications as demonstrated in the next section.

The `XtVaGetApplicationResources` Function

The `XtVaGetApplicationResources` function enables a much more convenient and safer method of parsing the command line. Note, however, that the function is responsible for much more than just the command line.

In fact, the options specified can also appear in a resource file for the application in the same way that widget attribute resources are defined.

The `XtVaGetApplicationResources` function has the following prototype:

```
void XtVaGetApplicationResources( Widget w,
                                  XtPointer base,
                                  XtResourceList resources,
                                  Cardinal num_resources,  ..., NULL );
```

The first parameter specifies a widget that is used to identify the resource database to search for the values of the `resources` specified. The widget that is specified is generally the application shell returned from the call to `XtVaAppInitialize`, as the class name of the application is used to determine the resource database maintained for this application.

The second parameter, `base`, is an address to the structure that will be used to store the values obtained by the call to `XtVaGetApplicationResources`. The relationship to the fields of the structure referenced by the second parameter and the `resources` will be clear shortly.

The next parameter, `XtResourceList resources`, is an array of `XtResource` elements.

The `XtResource` structure is defined as follows in the `Intrinsic.h` header file:

```
typedef struct _XtResources {
  String    resource_name;
  String    resource_class;
  String    resource_type;
  Cardinal  resource_size;
  Cardinal  resource_offset;
  String    default_type;
  XtPointer default_addr;
} XtResource, *XtResourceList;
```

The field `resource_name` is the name assigned by the programmer to the resource being created for inclusion on the command line or in a resource file.

By convention, the first letter of a resource name is lowercase, and subsequent words are capitalized, as in `cardHeight`, `useFrontPanel`, and so on.

The `resource_class` enables resources to be affected by a single entry in a resource file by referencing the class name of the resource instead of their individual names. For instance, `tabCardFont` and `bottomBannerFont` can both be of the class `Font`. Using only the class reference in an `.Xdefaults` file would assign the value to both. The syntax in the resource file to support this would be as follows:

```
*Font: 8x10
```

A `resource_type` identifies the data type of the resource value. Table 13.3 shows many of the common resource types understood by Intrinsics and the related C type.

Table 13.3 Resource Classes and Types

X Resource Type	C Type
XtRBoolean	Boolean
XtRColor	XColor
XtRCursor	XCursor
XtRDimension	Dimension
XtRFloat	Float
XtRFont	Font
XtRInt	int
XtRShort	short
XtRString	String

The `resource_size` specifies the number of bytes that the field of the structure referenced by `base` has reserved for this resource's value. The `sizeof` operator is used to determine this value.

The field `resource_offset` specifies the offset of the field in the structure referenced by `base` for storing the value associated with this variable. The `XtOffsetOf` function is used to determine this value.

The last two fields of the `XtResource` structure determine the default type and default value for this resource if no assignment is made either on the command line or in a resource file.

Listing 13.11 shows a code sample to properly define a resource array to be passed to the `XtVaGetApplicationResource` function.

Listing 13.11 The `XtVaGetApplicationResources` Function

```
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <stdio.h>
#include <stdlib.h>


struct _cmdArgs {
   char    *inFile;
   char    *outFile;
   Boolean verbose;
};

#define offset(field) XtOffsetOf(struct _cmdArgs, field)
static XtResource resources[] = {
 { "inFile", "InFile", XtRString, sizeof (char *),
       offset( inFile ), XtRString, (XtPointer)NULL },

 { "outFile", "OutFile", XtRString, sizeof (char *),
       offset( outFile ), XtRString, (XtPointer) NULL },

 { "verbose", "Verbose", XtRBoolean, sizeof (Boolean),
       offset(verbose), XtRString, (XtPointer)"off" },
};
#undef offset

static XrmOptionDescRec optionList[] = {
  { "-inFile",      "*inFile",     XrmoptionSepArg, (XPointer) NULL },
  { "-outFile",     "*outFile",    XrmoptionSepArg, (XPointer) NULL },
  { "-verbose",     "*verbose",    XrmoptionNoArg,  (XPointer) "on" },
};

void main( int argc, char **argv )
{
   struct _cmdArgs cmdArguments;

   XtAppContext appC;
   Widget       toplevel;

   toplevel = XtVaAppInitialize( &appC, "Demo",
                   optionList, XtNumber(optionList),
                                 &argc, argv,
                                 NULL, NULL );
```

```

    XtVaGetApplicationResources( toplevel,
                               (XtPointer)&cmdArguments,
                               resources, XtNumber(resources),
                               NULL);


  if( cmdArguments.inFile  )
    printf( "infile  = %s\n", cmdArguments.inFile  );

  if( cmdArguments.outFile )
    printf( "outfile = %s\n", cmdArguments.outFile );
}
```

Following the definition of a structure to contain all the values supported on the command line, the array `resources` is defined to inform Intrinsics of all the details about the resources understood by the application.

The resource names, classes, sizes, placement in the `_cmdArgs` structure, and even default values are provided in the `XtResource` array.

Next, a second array is created for inclusion in the call to `XtVaAppInitialize` in order to inform of the command-line arguments:

```
static XrmOptionDescRec optionList[] = {
```

When the call to `XtVaAppIniatalize` is made, all the resources known by the application are loaded into the X Resource Database. The second field to the `optionList` informed the database manager of pertinent resource file entries that would satisfy these resources and the first field instructed the valid command-line parameters.

Note

Resource names do not have to be the same as the field names of the structure which houses them. I've simply done this for clarity in determining the relationships of the resources, structure fields, and command-line options.

Finally, when the call to `XtVaGetApplicationResources` is made specifying the address of the `cmdArguments` structure, the values for the resources specified in the array are copied from the resource database to the offsets of each of the fields in the structure.

The same caveat must be applied here as was given to the `getopts` implementation for command-line processing. If a required argument is omitted following a flag that requires it, there is no enforcement.

Continuing with the progression of the Graphics Editor project, it is important to establish a complete understanding of how the canvas will receive and process the events needed for object creation and management.

Setting Up a Canvas

Focusing attention on the routine `create_canvas` responsible for creating the drawing area `GxDrawArea` in Listing 13.4, we must be clear in the understanding of the relationship to the event handlers assigned to the widget

```
34:    XtAddEventHandler( GxDrawArea, PointerMotionMask, False,
35:                       (XtEventHandler)drawAreaEventProc, (XtPointer)NULL);
36:    XtAddEventHandler( GxDrawArea, ButtonPressMask|ButtonReleaseMask, False,
37:                       (XtEventHandler)drawAreaEventProc, (XtPointer)NULL);
```

and the actual cursor management routines found in Listing 13.6 at lines 82–88

```
82:    void draw_manager( Widget w, XtPointer cd, XtPointer cbs )
83:    {
84:      void (*draw_func)( XEvent * ) = (void (*)(XEvent *))cd;
85:
86:      if( draw_func != NULL ) (*draw_func)( NULL );
87:      draw_mgr_func = draw_func;
88:    }
```

and lines 99–105:

```
99:    /*
100:    * drawAreaEventProc
101:    */
102:    void drawAreaEventProc( Widget w, XtPointer cd,
        ➥XEvent *event, Boolean flag )
103:    {
104:      if( draw_mgr_func != NULL ) (*draw_mgr_func)( event );
105:    }
```

As you might recall, the `draw_manager` routine was assigned as the command widget callback function for all the `gxDrawIcons` parsed and created by the `create_icons` function.

Review the client data passed as the last parameter to the `XtAddCallback` function in `create_icons`:

```
97:          XtAddCallback( btn, XtNcallback, callback,
             ➥(XtPointer)iconData->func );
```

Earlier I pointed out that the `iconData` reference is an element of the `gxDrawIcons` array. Looking again at the `gxDrawIcons` array definition, you see the object creation routines are being passed to the `draw_manager` callback function as client data when the user selects one of the object's drawing icons.

When the `draw_manager` function is invoked because a user selects one of these icons, it casts the client data to a function of type `void`, which expects an `XEvent` structure pointer as its only parameter:

```
84:      void (*draw_func)( XEvent * ) = (void (*)(XEvent *))cd;
```

If the function pointer is not `NULL`, the function is invoked with a `NULL XEvent` pointer reference to ensure that it has been reset from any previous creation:

```
86:      if( draw_func != NULL ) (*draw_func)( NULL );
```

Finally, the `draw_manager` function assigns the function pointer value to a global variable called `draw_mgr_func`:

```
87:      draw_mgr_func = draw_func;
```

All this occurs as a result of a graphic object draw icon being selected and it finishes before the user has even had a chance to move the cursor to the `GxDrawArea` canvas.

How does the event handler `drawAreaEventProc` assigned to the `GxDrawArea` relate to the `draw_manager` function?

Every time a `ButtonPress`, `ButtonRelease`, or `PointerMotion` event occurs in the window of the `GxDrawArea` widget, the `drawAreaEventProc` is invoked.

If the value of the global variable `draw_mgr_func` is not `NULL`, the `drawAreaEventProc` continually invokes the routine passing to it the current event:

```
104:      if( draw_mgr_func != NULL ) (*draw_mgr_func)( event );
```

The object-specific creation and management routines are introduced in Chapter 15, "Common Object Definition." These functions are responsible for performing the correct action based on current event type being sent.

Building the Project

Many files have been introduced in this chapter. All of them must be compiled and linked together to form a functional executable.

Chapter 1 provides an excellent introduction to the UNIX `make` utility and the `make` configuration file syntax. In fact, the sample configuration files can be used verbatim to build the source code files introduced here.

Note

If the examples for building the project introduced in Chapter 1 are used *as is*, the proposed project structure must be honored or the `vpath` values will not be correct.

➔ See Chapter 1, section "Makefile," page 31, for a complete review of the `make` utility and its configuration syntax.

Next Steps

After you have successfully compiled and linked the files introduced in this chapter (and ensured that the elements are functional), you are ready to advance to the next chapter.

Chapter 14, "Program Flow," provides a brief discussion of the overall flow and program execution to clarify how functions such as event handlers and callbacks are invoked.

In this chapter

- *Processing Events*
- *X Event Hooks*
- *Next Steps*

Chapter 14

Program Flow

Chapter 13, "Application Structure," focused on structuring the application and creating the graphical user interface. This chapter explains in more depth the nature of event-driven programming and the interaction between the application and the X Window System.

When the Graphics Editor is built and executed from the source code presented in Chapter 13, control is quickly relinquished to the `XtAppMainLoop` function.

Beyond the creation of the graphical user interface and initialization of global variables for use during program execution, the program does not follow the conventional program flow that you might have experienced in the past.

After the `XtAppMainLoop` is entered, execution lies somewhere in the Intrinsics library well out of sight, only returning control to the Graphics Editor through one of several *entry points*.

It is not possible to predict absolutely where in the code execution will be at any given moment. Nor is it possible to predict the order in which functions will be entered because program flow at this point depends on several factors, primary of which are the habits of the user.

The user's navigation of the application leads to the generation of events. These events can be in the form of `ButtonPress`, `ButtonRelease`, or `PointerMotion` in the drawing area window or callbacks invoked from the selection of a command widget. Other events of interest to the Graphics Editor are the `EnterNotify` and `LeaveNotify` events for updating the context-sensitive help message in the status window.

All the events registered with the widgets that form the user interface account for the multiple entry points into the editor application.

The event-driven behavior of an X-based application directly affects the heuristic method applied to the development of the application. Specifically, the programmer must account for and be prepared for entry into the application from any one of the many points registered with X for responding to user input.

Processing Events

The X events sent by the server to an application in response to user actions are, in fact, data structures. Approximately 33 different event structures are defined by the X Window System, and all of these events are relative to a window. Behaviors such as a mouse pointer motion, entering and leaving windows, and even requests to change the width or height of a window are communicated to the application by an appropriate event structure being filled and passed by the X Server to the client.

Of the 33 events understood by X, most are not communicated to the application unless explicitly requested. As was demonstrated in Chapter 13, section "Setting up a Canvas," page 296, specifying the appropriate event mask for events of interest requests of the X Server that these events are communicated to the application.

The most complex aspect of X Window System programming is the task of creating a function to account for all possible events that can occur in the windows and subsequently invoking the functions registered by the user for the event. This task is commonly known as the *event-loop*.

Fortunately, the X Toolkit Intrinsics, through the library functions `XtAppNextEvent` and `XtDispatchEvent`, simplifies the task. An understanding of how the presence of events within a window of the application translates to a function being called by the X Server is important to this discussion.

Several *hooks* are available for the X Window System to communicate events to an application. These mechanisms included callbacks, event handlers, and translations.

EXCURSION

Application Program Hooks

In computer programming, a *hook* is a place (usually accessed through an interface) provided in packaged code that enables a programmer to attach customized functions. This enables a programmer to insert additional capability into the package or library. In the context of the X Window System, these hooks enable a programmer to specify re-entry points to the application based on the occurrence of specific events.

Typically, hooks are provided for a stated purpose and are documented for the programmer. Some writers use *hook* to also refer to the function that is inserted.

As discussed, there is no assurance of when events will be communicated to the client application. Nor can it be known which event hook will be invoked at any given time.

Two reasons attribute to the seeming randomness by which events are communicated to an application are the *asynchronous* nature of X, and (as mentioned) the habits of the user as she navigates the application.

EXCURSION

The Asynchronous Nature of the X Window System

The X Server handles events *asynchronously*, meaning that events can occur in any order, and there is not a linear relationship to the requests generated by an application and the events that are returned to satisfy them.

The X Server takes a continuous stream of events from the display hardware and dispatches them accordingly to the appropriate applications (remember, an X Server addresses more than one client at any given time), which then take appropriate actions.

Four types of messages are passing between the client and server:

- *Requests* in which the client asks the server to perform a task or provide information.
- *Replies* from the server that can be immediate if the request by the client is a *roundtrip* request such as `XQueryGeometry`.
- *Events* that the X Server can *surprise* the client with in response to user action.
- *Errors* reported by the server to reflect an unrecoverable condition.

Asynchronous implies that the X Server sends events whether or not the client is ready to receive them.

Many `Xlib` functions (requests) made by the application cause the X Server to generate events. Additionally, the user's typing or moving the pointer will generate events. Both requests and events are buffered and the X Server addresses them *asynchronously* to maximize efficient use of the network.

When debugging an X Window application, it is often very convenient to request the X Server behave synchronously so that errors are reported at the moment they occur.

The following function lets you disable or enable synchronous behavior:

```
XSynchronous( Display *, Boolean );
```

The second parameter is expected to be either `True` or `False` indicating whether the X Server should start (`True`) or stop (`False`) its synchronous behavior.

On POSIX-conformant systems a global variable `_Xdebug` exists that, if set to a non-zero value before starting the program, will force synchronous behavior.

Note that graphics can update significantly slower (30 times or more) when synchronization is enabled, and it is therefore only used for debugging.

In normal asynchronous operations, X Window clients should be constantly prepared for any event that could be sent to them. For instance, windows within an application can be obscured by other applications and then exposed, requiring the applications to redraw.

X Event Hooks

The following sections address the mechanisms by which X invokes the functions registered as the event hooks for an application.

Widget Callbacks

The X Server generates events as it monitors the display hardware. The manner in which the user manipulates widgets by pushing a button, selecting a list item, choosing a menu item, or even moving the mouse pointer results in the X Server creating a corresponding event.

Graphical, event-driven programming is different from traditional programming in that the program does not dictate the flow of control.

Instead, the users, by their choice of actions, indirectly generate events, which the program might or might not respond to.

One type of response to events is a *callback*. Each callback has an associated reason (event type) for which the programmer can register a procedure. This procedure is then called every time the associated reason or event occurs.

For example, if you registered a callback for a command widget, it would be called each time the button was pressed.

The Intrinsics library does not guarantee an order. This is because both the widget author and the application programmer can modify the entire contents of the callback list.

The widget writer is not required to document when the widget will internally add or remove callbacks from the list associating callback procedures to the widget. Therefore, the functionality contained in a callback procedure should be independent of the functionality contained in other callback functions registered by the programmer.

If a widget does not maintain a callback list for the reason (event) needed by the application programmer, an event handler can be assigned to the widget.

Event Handlers

When a widget is created, it will know how to respond to certain events, such as a window manager's request to change size, color, background, or border size. This knowledge enables a command widget to appear highlighted when the mouse is clicked within its window or a menu to cascade when selected by the user.

However, hooks do not exist for all events understood by a widget. The command widget will not inherently enable a programmer to associate a new procedure for invocation at occurrence of the `EnterNotify` event as it does for `ButtonPress`. (Remember, at the presence of a `ButtonPress` event the command widget will call the function registered with the `XtNcallback` resource.)

To overcome this, a programmer would use the `XtAddEventHandler` to force a widget to call a registered action at the presence of a specific event.

As seen in Chapter 13, command widgets do not enable an action to be associated for the events notifying the widget when the mouse cursor enters or leaves the widget's window. The `XtAddEventHandler` expands a widget's capabilities by instructing it to look for and act on events that it otherwise would ignore.

The relationship between callbacks and event handlers should be clear. Specifically, callbacks are lists inherent to a widget used to associate procedures to events. Event handlers are a less direct approach that expands the capabilities of the widget.

Translation tables are a final method for linking an application function to the internals of a widget.

Widget Translation Tables

Every widget has a *translation table* that defines how it will respond to particular events. These events can invoke one or more *actions*.

An example of part of the translation table for the command widget is the following:

```
BSelect Press: Arm()
BSelect Click: Activate(), Disarm()
```

`BSelect Press` corresponds to a mouse press and the translation table associates the action `Arm` to this translation. The `Arm` action maps to a function internal to the widget that causes, for instance, the command widget to highlight for appearing pressed.

If the mouse is clicked (pressed and released), `Activate` and `Disarm` functions are called. The `Activate` function internal to the widget looks for functions registered with the widget resource `XtNcallback` and invokes those it finds. The `Disarm` function returns the widget to normal coloring by turning off the widget highlighting.

Keyboard events can also be listed in the table to provide facilities such as hot keys, function or numeric select keys, and help facilities.

Examples of keyboard translations include `KActivate`, which typically refers to the Enter key, or `KHelp` for the HELP or F1 key. Pertinent actions are associated with these translations.

An advanced feature of the X Window System is the capability to add actions to existing translations or add new translations to a widget's translation table.

This feature, though not exercised in the Graphics Editor application, is employed with the functions `XtParseTranslationTable` and `XtAugmentTranslations`.

The function `XtParseTranslationTable`, as the name implies, parses a translation table established by the application developer, the result of which can be installed with the function `XtAugmentTranslations`.

The introduction of translation tables provides a complete coverage of the hooks available for an application developer to expand and advance the X Window environment.

Next Steps

The discussion of the application's flow in this chapter should bridge the understanding of the functions you author and their relationship to the processing of X Window events.

With a better understanding of the program flow of the Graphics Editor and its relationship to the X libraries and Server, you are ready to structure the graphic objects that are used to represent entities of the application.

The next chapter will advance the structure of the Graphics Editor project by building the common object component from which all objects in the editor will be based.

Chapter 15

In this chapter

Common Object Definition

Following the detailed look provided in Chapter 14, "Program Flow," of the Graphics Editor program flow and, specifically, the X Event hooks important to the editor project, you are ready to advance the structure of the Graphics Editor. This chapter presents the common object definition from which all objects supported by the editor are created.

Note All the data structures introduced in this chapter are targeted for inclusion in the `gxGraphics.h` header file.

➔ Chapter 9, "Object Bounds Checking," page 203, introduced the idea that the objects supported by the Graphics Editor are divided into two families: point-array–based objects and arc objects.

Line Object Data Structure

The line objects presented in Figure 15.1 include the LatexLine (star shape), PolyLine, Box, and Arrow graphic objects.

These objects are all point-array–based: An array of points is used to represent them in the Graphics Editor application.

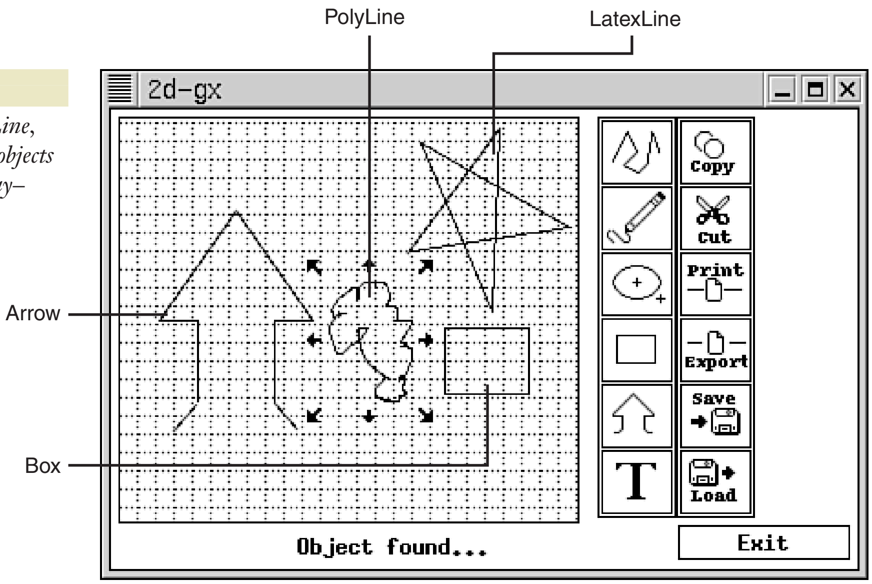


Figure 15.1
LatexLine, PolyLine, Box, and Arrow objects are all point-array–based objects.

Note

Notice in Figure 15.1 that the PolyLine object is *selected* as indicated by the scale handles surrounding the object.

The section "Common Object Data Structure," later in this chapter, explains fully the necessity of selecting the objects created in the Graphics Editor application and the data fields that manage an object's activity.

Listing 15.1 provides a structure for representing all the objects shown in Figure 15.1.

Listing 15.1 The Graphic Line Object Structure Definition

```
1:    typedef struct _gxline {
2:        XPoint *pts;
3:        int    num_pts;
4:    } GXLine, *GXLinePtr;
```

The `GXLine` structure is very simple, requiring only the array of points

```
2:        XPoint *pts;
```

and the number of elements contained in the array:

```
3:        int    num_pts;
```

The XPoint array pts stores the vertices of the line object in the elements of the array. Clearly, the number of vertices corresponds directly to the number of array elements as reflected by num_pts.

This structure definition declares two new data types for use in the Graphics Editor application:

```
4:    } GXLine, *GXLinePtr;
```

The GxLine data type will refer to an actual occurrence of the structure and GxLinePtr, as the name implies, will point to an occurrence of the structure.

Not nearly as simple is the structure required for representing a vector text object in the Graphics Editor application.

Text Object Data Structure

The vector text object is the most complex object supported by the Graphics Editor application. What makes it so complicated is that the text object comprises many lines.

To illustrate, try the following: Using a piece of scrap paper or the back of your hand, try to draw the letter T (consisting of two lines—a vertical stem and a horizontal crossbar) without picking up your pencil and without backtracking over a line that you've already drawn.

The need to lift the pen in order to put the crossbar on the stem should be evident.

To accomplish this programmatically, the Graphics Editor uses an array of lines. As seen in the previous section introducing the Line Object data structure, the line object is itself an array of points. In the end, the text object data structure is an array of arrays where implicitly the routine for drawing the lines that compose a character lifts the pen before drawing the next line.

The concept of lifting a pen, of course, is figurative. The text draw routine accomplishes this by issuing separate draw requests for each line in the array.

Figure 15.2 shows a sample screen of three text objects created by the Graphics Editor application.

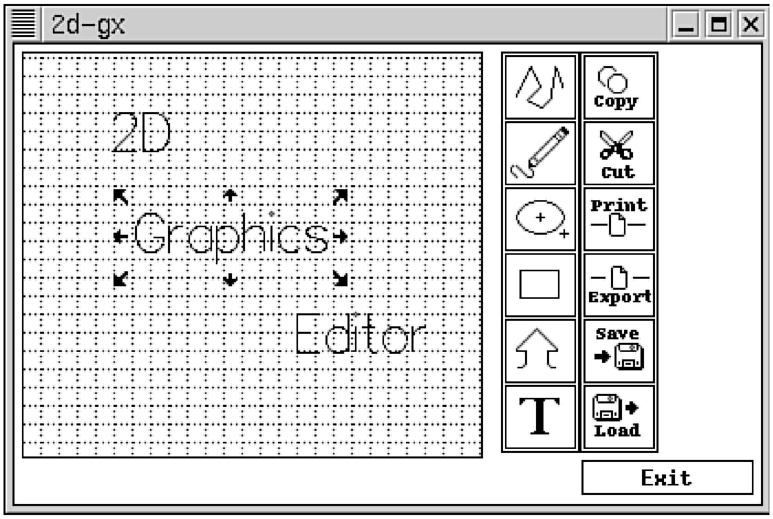


Figure 15.2

The point-array–based Text object.

The data structure used to represent the Graphics Editor Text is shown in Listing 15.2.

Listing 15.2 The Graphic Text Object Structure Definition

```
1:    typedef struct _gxtext {
2:        int x, y;    /* upper-left */
3:        int dx, dy;
4:        char *text;
5:        int  len;
6:        GXFont  vpts;
7:        GXFont  font;
8:        GXFontP fontp;
9:    } GXText, *GXTextPtr;
```

The `GXText` data type introduced in Listing 15.2 contains elements for tracking the position in the drawing area of the text string managed by the object

```
2:        int x, y;    /* upper-left */
```

as well as the actual string and the string’s length:

```
4:        char *text;
5:        int  len;
```

The fields of the structure

```
3:        int dx, dy;
```

are used to manage the scale feature of the text object and will be discussed in detail in Chapter 24, “Vector Text Object.”

The remaining three fields

```
GXFont  vpts;
GXFont  font;
GXFontP fontp;
```

are required to manage specifics of the character definitions for the vector font set that is employed for this graphic text object.

The `font` and `fontp` fields

```
GXFont  font;
GXFontP fontp;
```

are effectively read-only and refer to the font assigned to this object.

A significant complication related to the vector text object involves the font or character definitions that govern the appearance of characters drawn by the text object.

The following section introduces vector font concepts and the complicated data management employed to support them.

Understanding Vector Fonts

Any font that is defined by mathematical routines that can reproduce the outlines of each character at any size is a vector font.

Vector fonts, also known as *scalable*, *hershey*, or *outline fonts*, describe the shape of each character but not the character's size.

A character's shape definition consists of the vectors (positions relative to an origin) of the endpoints of the line segments that comprise each *stroke* of the character.

For example, consider the shape definition shown in Listing 15.3 for drawing the letter *A* using a simple vector font.

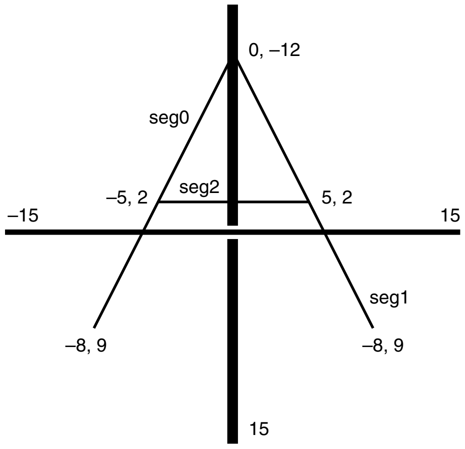
Listing 15.3 Vector Font Character Definition for the Letter *A*

```
static XPoint seg0_A[] = {
    {0,-12}, {-8,9},
};
static XPoint seg1_A[] = {
    {0,-12}, {8,9},
};
static XPoint seg2_A[] = {
    {-5,2}, {5,2},
};
static XPoint *char65[] = {
    seg0_A, seg1_A, seg2_A,
    NULL,
};
```

The character definition for the letter A shown in Listing 15.3 consists of three line segments. Figure 15.3 illustrates the orientation of the vectors comprising each of the segments.

Figure 15.3

Representation of the line segments defining the letter A *for a simple vector font.*



As shown in Figure 15.3, the origin of the graph defining a character in this vector font set is located in the center of the cell. As this origin differs from the origin used by the X graphic primitives, the definition of the text object data structure must account for correctly placing the text by tracking the character string's location.

A review of what you now know about vector fonts is important.

A single character consists of multiple line segments that are composed of multiple points. In other words, an array of points forms a line segment and an array of line segments (array of points) defines a character.

Putting this all together, we can say that an array of arrays of points forms a character. Therefore, an entire font set (that is all character definitions) is an array of characters, which is an array of arrays of points. An entire font set is represented as

```
XPoint **plain_simplex[] = {
  char32,  char33,  char34,  char35,  char36,  char37,
  char38,  char39,  char40,  char41,  char42,  char43,
  char44,  char45,  char46,  char47,  char48,  char49,
  char50,  char51,  char52,  char53,  char54,  char55,
  char56,  char57,  char58,  char59,  char60,  char61,
  char62,  char63,  char64,  char65,  char66,  char67,
  char68,  char69,  char70,  char71,  char72,  char73,
  char74,  char75,  char76,  char77,  char78,  char79,
  char80,  char81,  char82,  char83,  char84,  char85
  char86,  char87,  char88,  char89,  char90,  char91,
  char92,  char93,  char94,  char95,  char96,  char97,
  char98,  char99,  char100, char101, char102, char103,
  char104, char105, char106, char107, char108, char109,
  char110, char111, char112, char113, char114, char115
  char115  char116, char117, char118, char119, char120,
  char121, char122, char123, char124, char125, char126,
};
```

The elements of the `plain_simplex` font set are character definitions. The arrays of line segments defining each of the characters are presented in Chapter 24, "Vector Text Object." Important to our present discussion is the understanding that the font set consists of an array in which each element is itself an array and each of these arrays further contain arrays of points.

The `GXFont` data type has been defined in the Graphics Editor project to represent a font set.

```
typedef XPoint ***GXFont;
```

The length of the arrays is important to every array nested in the `GXFont` data type defining a vector font set. To track this important length information, a second array is defined:

```
typedef int    ** GXFontP;
```

The `GXFontP` array, however, contains elements of type `int` where each element corresponds to a length value for the corresponding element in the `GXFont` data type.

Use of these two arrays is closely linked and follows the convention reflected in the following code snippet:

```
static void GXDrawText( GXTextPtr text, GC gc )
{
  char *txt = text->text;
  int  c, chr, nsegs, num_pts;

  for( c = 0; c < text->len; c++, txt++ ) {
    chr = *txt - ' ';
    nsegs = 0;

    while( text->font[chr][nsegs] != NULL ) {
      num_pts = text->fontp[chr][nsegs];

      if( num_pts > 0 ) {
        XDrawLines( XtDisplay(GxDrawArea),
                    XtWindow(GxDrawArea), gc,
                    text->vpts[c][nsegs], num_pts,
                    CoordModeOrigin );
      }
      nsegs++;
    }
  }
}
```

This snippet is the actual code used to draw the text string contained in a `GXText` reference. It is officially introduced in Chapter 24, but is included here to show how a vector font set is employed.

The routine begins by extracting the text string from the `GXText` structure referenced by `text`:

```
3:      char *txt = text->text;
```

Then for each character in the text string

```
6:      for( c = 0; c < text->len; c++, txt++ ) {
```

an index into the font set is calculated by subtracting the decimal value of a space (‘ ’) from the character to be drawn:

```
7:        chr = *txt - ' ';
```

This works because the characters in the font set are ordered identically to the printable characters found in the ASCII table as shown in Figure 15.4.

Figure 15.4

The ASCII table.

Dec	Char	Dec	Char	Dec	Char
32	sp	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	'	76	L	108	l
45	–	77	M	109	m
46	'	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Notice in Figure 15.4 that the decimal value of a space is 32. As the first character defined in the vector font set is the space (`char32`), if a space is to be rendered by the `GXDrawText` function, it subtracts a space (decimal 32) from a space resulting in the value of `chr` being 0. An index of zero corresponds to the first element of the array, which is in fact the definition of the space character.

Note

Although a space can't be *drawn*, its definition in the vector font set is important for separating words an appropriate distance when it is used in a text string managed by the graphic text object.

Applying this to the letter *A* (`char65`) you see that 'A' – ' ' or 65 – 32 is 33, which corresponds to the 34th element of the array, which is the character definition for `char65`.

After calculating a value for `chr` (index of the character definition in the font set), the `GXDrawText` function can use `chr` to index into the array defining the number of points for the segments of which this character is composed:

```
11:         num_pts = text->fontp[chr][nsegs];
```

With this data, the routine has all the information it needs to draw a single segment of the character.

As we saw with the definition of a single character

```
static XPoint *char65[] = {
    seg0_A, seg1_A, seg2_A,
    NULL,
};
```

the character definitions are `NULL` terminated. This enables us to loop until all segments of the character have been drawn:

```
10:       while( text->font[chr][nsegs] != NULL ) {
```

After drawing a segment, the routine increments to the next one

```
19:         nsegs++;
```

and again tests for `NULL`.

This finishes the introduction to vector fonts and simple character definitions. Chapter 25, "Introduction to PostScript," will complete the subject and provide the remaining character definitions for this font.

The following section continues the discussion of the text object data structure.

The `GXText` Data Structure

Looking closer at the requirements of the text object supported by the graphics editor application, notice that the `vpts` field found in the `GXText` data structure in Listing 15.2

```
6:        GXFont  vpts;
```

are the actual points used to draw the object to the canvas. The value of `vpts` is derived by transposing the character definitions in the font set for each character of the text string to the location referenced by the `x` and `y` fields of the `GXText` structure.

Understanding the `GXText` data structure and the complex `GXFont` data type will take some time. These definitions will be reviewed and explained in greater detail in Chapter 24.

The complexity of the `GXText` data structure is inversely proportional to the `GXArc` data structure introduced in the next section.

Arc Object Data Structure

The arc objects are the simplest data structure of the graphic objects supported by the Graphics Editor. Examples of the graphic arc object are illustrated in Figure 15.5.

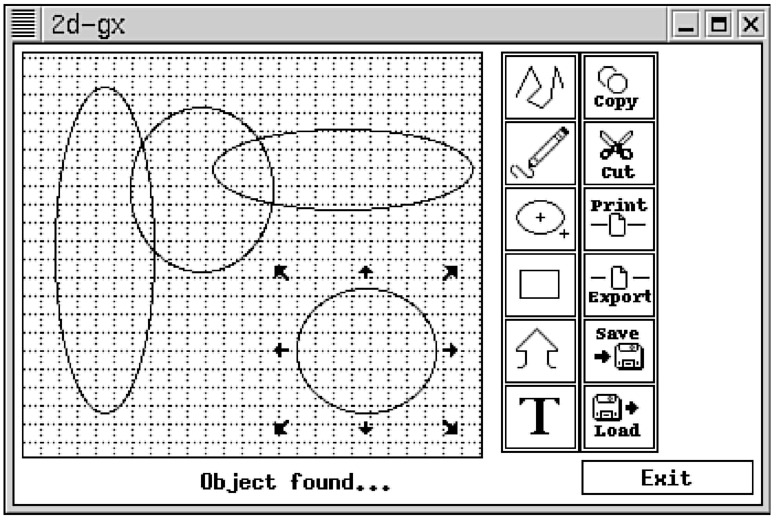


Figure 15.5

Examples of the graphic Arc object.

The data types for the structure used to manage the graphic arc objects shown in Figure 15.5 are found in Listing 15.4

Listing 15.4 `GXArc` Data Type Definition

```
typedef XArc GXArc;
typedef XArc *GXArcPtr;
```

Fortunately, the X Window System provides a structure for representing the Arc objects, and this structure is sufficient for use in the Graphics Editor.

With the specific data requirements for the various graphic objects defined and explained in the preceding sections, we are ready to introduce the common object data structure and demonstrate its relationship to the specific graphic objects.

Common Object Data Structure

All the graphic objects supported by the Graphics Editor application require many pieces of information.

These data fields are not repeated for every object type but are contained in a common object data structure from which individual objects are derived.

Listing 15.5 introduces the common object definition used by all graphic objects in the Graphics Editor.

Listing 15.5 Common Object Data Structure

```
typedef struct _gx_obj {
  /*
   * attributes
   */
  Pixel fg; /* foreground */
  Pixel bg; /* background */

  Pixmap fs; /* fill style */
  int    ls; /* line style */
  int    lw; /* line width */

  Boolean selected;

  XRectangle *handles;
  int        num_handles;

  void *data;

  void (*draw)  ( struct _gx_obj * );
  void (*erase) ( struct _gx_obj * );

  XRectangle *(*bounds)( struct _gx_obj * );
  Boolean    (*find)  ( struct _gx_obj *, XEvent * );

```

continues

Listing 15.5 Continued

```
25:     void (*select)    ( struct _gx_obj * );
26:     void (*deselect)  ( struct _gx_obj * );
27:     void (*copy)      ( struct _gx_obj * );
28:
29:     void (*move)  ( struct _gx_obj *, XEvent * );
30:     void (*scale) ( struct _gx_obj *, XEvent * );
31:
32:     void (*action)  ( struct _gx_obj *, XEvent * );
33:
34:     /*
35:      * link to other objects
36:      */
37:     struct _gx_obj *next;
38:
39:   } GXObj, *GXObjPtr;
```

The first several fields maintain the values of the attributes assigned to an individual object:

```
5:      Pixel fg; /* foreground */
6:      Pixel bg; /* background */
7:
8:      Pixmap fs; /* fill style */
9:      int    ls; /* line style */
10:     int    lw; /* line width */
```

These fields will be used to create the Graphic Context (`GC`) used to draw the object.

Following the attributes are fields that manage whether the object is selected

```
12:     Boolean selected;
```

and drawing handles to reflect an active status:

```
14:     XRectangle *handles;
15:     int        num_handles;
```

The `selected` field is set to `True` when the object is selected and `False` otherwise.

When an object is selected (or active), eight handles are drawn to reflect this status and indicate the scale directions available for changing the size of the object. Only when an object is active can it be scaled, moved, copied, or cut from the screen.

The next field in the common object data structure is used to store the object-specific data structures for the various graphic object types understood by the editor:

```
17:     void *data;
```

The `data` field can point to any one of the structures introduced early for representing the specific data requirements of the editor objects.

Assigning the `data` field a reference to a valid `GXLinePtr`, `GXTextPtr`, or `GXArcPtr` data structure uniquely creates the specific graphic type.

A specific relationship exists between the structure reference used in the `data` field assignment when an object is created and the value of the remaining fields of the common object data structure. These remaining fields are pointers to functions, and when nested in a structure definition they are called *methods*.

The following methods are defined for all objects and, as stated, the value of these methods will be relative the structure type referenced in the data field:

```
void (*draw)  ( struct _gx_obj * );
void (*erase) ( struct _gx_obj * );

XRectangle *(*bounds)( struct _gx_obj * );
Boolean    (*find)  ( struct _gx_obj *, XEvent * );

void (*select)    ( struct _gx_obj * );
void (*deselect)  ( struct _gx_obj * );
void (*copy)      ( struct _gx_obj * );

void (*move)  ( struct _gx_obj *, XEvent * );
void (*scale) ( struct _gx_obj *, XEvent * );
```

These methods are explained by their names. The `draw` and `erase` methods require a single parameter referencing the object to be drawn or erased.

The `bounds` method also requires a reference to the object and returns a reference to an `XRectangle` reflecting the bounds of the object specified.

The `find` method requires, in addition to an object reference, a reference to an `XEvent` structure defining the `x` and `y` location of the mouse cursor when the object selection was attempted.

The `select` and `deselect` methods toggle the `selected` field to reflect the current status and create or destroy the `handles` reference as required by the change in status.

The `copy` action is accessed by the icon on the button panel to the right of the graphical user interface. When an object is active (selected) and the copy button is clicked, the active object is duplicated, with the copy placed next to the original.

Finally, the `move` and `scale` methods are implied by the user's navigation of the mouse cursor. Because these actions are modal, meaning they are dependent on the user's wishes, the `action` field

```
void (*action)  ( struct _gx_obj *, XEvent * );
```

tracks the current mode (`move` or `scale`) in use.

The last field of the common object data structure

```
37:     struct _gx_obj *next; /* entire list of all objects */
```

is used to create a *linked list*.

EXCURSION

What Is a Linked List?

A *linked list* is a programming construct that enables *nodes* (data structures) to be linked together by a reference to the next node:

```
node (head) -> node -> node -> node -> NULL
```

The usefulness of a linked list lies in the fact that it can contain an unknown number of nodes. Contrast this to an array that has a finite number of elements determined when the array is either defined or allocated.

It is important when using the linked list construct that the *head*, or beginning of the list, be meticulously maintained. If for any reason a reference to the beginning of the list is lost, the entire list is also lost. As demonstrated with this previous linked list, a `NULL` pointer marks the last node in the list.

Chapter 16, "Object Manipulation," section "Deleting an Object," page 321, and Chapter 17, "Utilities and Tools," section "Linked List Management," page 346, discuss functions used by the Graphics Editor for adding and removing nodes to the linked list supported by the common object data structure.

The linked list provided for in the common object data structure is used to maintain the objects created in the editor.

All objects created by the Graphics Editor will be appended to the `next` field of the previously created object with the first object defining the head of the list.

This completes the definition of the vital data structures required by the Graphics Editor application.

Next Steps

An understanding of the data fields and methods defined in the common object data structure will allow us to introduce functions that employ them.

Chapter 16 will demonstrate how the graphic objects created by the editor are treated identically despite the specific type represented by the object. This generic treatment of the graphic objects is accomplished by using the methods and data elements of the common object data structure.

Chapter 16

Object Manipulation

Chapter 15, “Common Object Definition,” led you through the definition of the data structures used to create the common graphic object as well as the unique structures for representing the supported graphic object types in the Graphics Editor.

Using the methods and data fields provided by the common object data structure, the Graphics Editor can build functions capable of managing objects without regard to their type.

This chapter introduces the functions that manipulate objects generically.

Note

Many of the functions presented in this chapter complete the stub (empty) functions used to satisfy necessary function resolutions in Chapter 13, “Application Structure.”

All the functions introduced in this chapter are intended for inclusion in the `gxGx.c` source file of the Graphics Editor project.

Copying an Object

The source code in Listing 16.1 satisfies the function invoked when the Copy icon from the button panel of the Graphics Editor interface is pressed.

Listing 16.1 Source Code for the `copy` Function

```
void gx_copy( void )
{
  if( gxObjCurrent ) {
```

continues

Listing 16.1 Continued

```
4:          (*gxObjCurrent->copy)( gxObjCurrent );
5:          gx_refresh();
6:        } else
7:          setStatus( "Select an object to copy!" );
8:      }
```

The `gx_copy` function defined in Listing 16.1 checks for the presence of the global variable `gxObjCurrent`

```
3:        if( gxObjCurrent ) {
```

which refers to the currently active object in the Graphics Editor application.

If there is a currently active object, the `gx_copy` function invokes the `copy` method contained in the common object structure definition, passing the current object to satisfy the function's parameter requirements:

```
4:          (*gxObjCurrent->copy)( gxObjCurrent );
```

This method, as explained in Chapter 15, "Common Object Definition," in the section "Common Object Data Structure," page 315, is assigned based on the specific graphic object type contained in the `data` field of the common object data structure.

EXCURSION

C Is Not C++

If you are familiar with the C++ programming language, you know that a feature of the language is the capability of all objects to make an implicit self-reference with the keyword `this`.

Unfortunately, this mechanism is not available in the C programming language. For this reason, a consistent parameter requirement for the methods contained in the common object data structure is a reference to the graphic object whose method is being invoked.

Following the invocation of the copy method, all the objects are redrawn with a call to the `gx_refresh` function.

If there is not an object currently selected, the `else` clause of the initial test is entered

```
6:        } else
7:          setStatus( "Select an object to copy!" );
```

and an information message is placed in the status label, indicating the intended order of actions for copying an object.

Located near the Copy icon in the Graphics Editor button panel is the Cut icon. The function to satisfy this icon is discussed in the following section.

Deleting an Object

Previously introduced as a stub function in Chapter 13, the code in Listing 16.2 provides the functionality to remove the currently selected object from the canvas and the linked list of created objects.

Listing 16.2 Source Code for the `delete` Function

```
void gx_delete( void )
{
  GXObjPtr gx_objs;
  Boolean  found;

  if( gxObjCurrent ) {

    (*gxObjCurrent->deselect)( gxObjCurrent );
    (*gxObjCurrent->erase)( gxObjCurrent );

    if( gxObjCurrent->data ) {
      XtFree((char *)gxObjCurrent->data );
      gxObjCurrent->data = NULL;
    }

    found = False;
    gx_objs = gxObjHeader;

    /*
     * find this object in the list and remove it
     */
    while( gx_objs->next && (!found)) {

      /* we want the reference just before us... */
      found = (gx_objs->next == gxObjCurrent );

      /* ...now instead of pointing at us,
       * point to what we point to
       */
      if( found ) {
        gx_objs->next = gxObjCurrent->next;
        gxObjCurrent->next = NULL;
      } else {
        gx_objs = gx_objs->next;
      }
    }

    /*
     * if we didn't find ourselves in the list, we are the list!
     */
    if( found == False ) {
      if( gxObjHeader == gxObjCurrent )
        gxObjHeader = gxObjCurrent->next;
```

continues

Listing 16.2 Continued

```
44:         else
45:           setStatus( "Panic: object to delete not found list!" );
46:       }
47:
48:       /* do some house cleaning */
49:       XtFree((char *)gxObjCurrent );
50:       gxObjCurrent = NULL;
51:
52:       /* redraw remaining objects (if any) */
53:       if( gxObjHeader ) gx_refresh();
54:     } else
55:       setStatus( "Select an object to delete!" );
56:   }
```

As with the `gx_copy` function, the `gx_delete` routine shown in Listing 16.2 begins by testing for the presence of a currently selected object. In its absence, an information message is placed in the status label, providing a hint to the user of the intended sequence of events for deleting an object.

If, however, an object is currently selected, the body of the `if` statement is entered where the object's `deselect` method is invoked to remove the object's handles

```
8:        (*gxObjCurrent->deselect)( gxObjCurrent );
```

and then the object is erased from the screen by a call to its `erase` method:

```
9:        (*gxObjCurrent->erase)( gxObjCurrent );
```

A test for the existence of the object-specific data is performed

```
11:       if( gxObjCurrent->data ) {
```

and if it exists, the memory associated with it is returned to the heap:

```
12:         XtFree((char *)gxObjCurrent->data );
13:         gxObjCurrent->data = NULL;
```

It is then necessary to parse the entire linked list of objects in search of the one being deleted so that it can be removed from the list.

This process is started by assigning a variable to point to the beginning of the list:

```
17:       gx_objs = gxObjHeader;
```

It is important that the variable `gxObjHeader` is not used directly because the reference will be advanced through the list and it is crucial that the beginning of the list is maintained; otherwise the entire list will be lost.

The construct of the loop

```
22:       while( gx_objs->next && (!found)) {
```

reads, “So long as there is another object in the list (`gx_objs->next`) and the one being deleted has not been found (`!found`), continue looping.”

The variable found is assigned directly the Boolean result of the comparison of the next object in the list and the object being deleted:

```
25:         found = (gx_objs->next == gxObjCurrent );
```

As mentioned previously, when the objects match, `found` will interrupt the loop and the `next` field of the current object will point to the one being deleted.

When a match is found, it is removed from the list by reassigning what the current node’s `next` field points to:

```
30:         if( found ) {
31:           gx_objs->next = gxObjCurrent->next;
32:           gxObjCurrent->next = NULL;
```

As long as the object has not been found, progress continues through the list by moving to the next node:

```
33:         } else {
34:           gx_objs = gx_objs->next;
35:         }
```

EXCURSION

Managing Pointers in a Linked List

It is important that you understand how the fragment

```
31:           gx_objs->next = gxObjCurrent->next;
```

removes `gxObjCurrent` from the list of graphic objects known to the Graphics Editor application.

A linked list, introduced in Chapter 16, “Object Manipulation,” in the Excursion “An Introduction to Linked Lists,” page xx, is a construct that allows a dynamic number of items to be managed by an application.

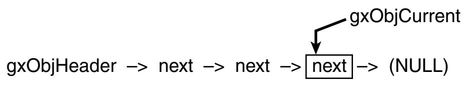
The `next` field of the common object data structure enables us to link the graphic objects created in the Graphics Editor together.

Conceptually, we depict this list as

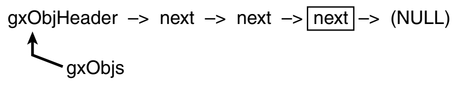
```
gxObjHeader->next->next->next->next->(NULL)
```

meaning the `gxObjHeader`, whose value is a reference to the first object created by the application, has as the value of its `next` field a reference to the second object created, which has as the value of its `next` field a reference to the third and so forth. The most recently created object will not have a valid reference assigned to its `next` field, thus marking the end of the list.

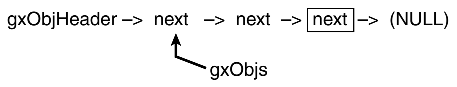
When the list is traversed in search of the object being deleted, the value of `gxObjCurrent` is somewhere in the list.



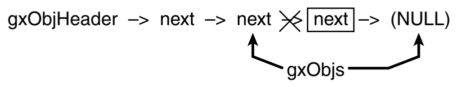
On the first iteration of the traversal, the variable `gxObjs` refers to the beginning of the list.



With subsequent iterations, the `gxObjs` value is advanced to equal the next node.



This continues until the current object is found, at which point the `next` field *referring* to the current object is assigned the reference *pointed to* by the current object.



Linked lists are covered again in Chapter 17, “Utilities and Tools,” in the section “Linked List Management,” page 346.

Notice again the test for the object being deleted being found:

```
25:         found = (gx_objs->next == gxObjCurrent );
```

The test is against the next field of the `gx_objs` variable. This means that the `gxObjHeader` value (first object created) is not considered in the body of the loop. This is important because the head of a linked list is always a special case.

If the loop ends without the object being found

```
41:       if( found == False ) {
```

an explicit test is performed against the beginning of the list:

```
42:         if( gxObjHeader == gxObjCurrent )
```

If the object being sought is indeed the beginning of the list, its removal is accomplished with the line

```
43:           gxObjHeader = gxObjCurrent->next;
```

and a new linked list head is assigned.

Finally, the object is permanently deleted from memory

```
49:       XtFree((char *)gxObjCurrent );
```

and the objects still maintained in the editor are refreshed to redraw any portion of them that might have been obscured by the object that was deleted.

```
53:       if( gxObjHeader ) gx_refresh();
```

The following section introduces the `gx_refresh` function used to redraw the entire list of objects controlled by the Graphics Editor application.

Refreshing Objects

To account for the immediate graphic nature of the X Window System as introduced in Chapter 4, "Windowing Concepts," in the section "Expose," page 122, it is often necessary to refresh the graphic objects displayed on the drawing area canvas. The `gx_refresh` function introduced in Listing 16.3 accomplishes this task.

Listing 16.3 Source Code for the `refresh` Function

```
1:    void gx_refresh( void )
2:    {
3:      GXObjPtr obj = gxObjHeader;
4:
5:      while( obj ) {
6:        (*obj->draw)( obj );
7:        gx_draw_handles( obj );
8:        obj = obj->next;
9:      }
10:   }
```

The `gx_refresh` function starts at the beginning of the linked list

```
3:      GXObjPtr obj = gxObjHeader;
```

and loops until it finds the end of the list marked by `NULL`:

```
5:      while( obj ) {
```

For every object in the list, the object's `draw` method is invoked

```
6:        (*obj->draw)( obj );
```

its handles are updated

```
7:        gx_draw_handles( obj );
```

and the loop proceeds to the next node in the list:

```
8:        obj = obj->next;
```

Perhaps one of the simplest functions acting against the common object definition, `gx_refresh`, is one of the most vital. Without it the objects would never get redrawn to the drawing area window.

Another critical function is the one introduced in the next section that enables us to find an object selected by a `ButtonPress` event.

Parsing for an Object

The capability of a user to select an object by clicking the mouse cursor on it is a vital function in the Graphics Editor. By selecting a graphic object, the user can scale, move, copy, cut, rotate, and modify the attributes of an object.

The `parse_all_objects` function introduced in Listing 16.4 will compare the `x` and `y` components of an `XEvent` to all the objects managed by the Graphics Editor to determine whether an object has been selected.

Listing 16.4 Source Code for Parsing the Objects

```
1:    static void
2:    parse_all_objects( GXObjPtr obj, XEvent *event,
3:                       GXObjPtr *gx_obj )
4:    {
5:      if( obj && (*gx_obj == NULL)  ) {
6:        if( (*obj->find)( obj, event ) ) {
7:          *gx_obj = obj;
8:          setStatus( "Object found..." );
9:        } else {
10:
11:         setStatus( "No objects found..." );
12:         parse_all_objects( obj->next, event, gx_obj );
13:       }
14:     }
15:   }
```

The `parse_all_objects` function expects three parameters. The first parameter is a reference to a graphic object where the current iteration of the search is to begin.

The second parameter is the `XEvent` containing the x, y location of the position of the mouse cursor when the mouse button was clicked.

Last is a pointer to a graphic object pointer:

```
3:                       GXObjPtr *gx_obj )
```

This enables the return of the object found by the parse function to the calling function.

This function is recursive, meaning that it calls itself. Therefore, the initial test of `parse_all_objects` is to ensure that neither the object selected by the event nor the end of the list has been found:

```
  if( obj && (*gx_obj == NULL)  ) {
```

If there is an object to test and no object has been found, the return value of the object's `find` method is used to determine whether the event successfully selected the current object:

```
    if( (*obj->find)( obj, event ) ) {
```

A `True` returned from the `find` method indicates that the object should be saved in the return field of the parameter list:

```
      *gx_obj = obj;
```

A status is reported and the recursion unfolds, returning the selected object.

The `find` method returning `False` leads to another level of recursion being invoked

```
      parse_all_objects( obj->next, event, gx_obj );
```

with the `next` element of the list passed for testing against the event reference.

At some point, either an object will be deemed selected or the `obj->next` reference passed to the subsequent level of recursion will be `NULL` and the test

```
  if( obj && (*gx_obj == NULL)  ) {
```

will end the cycle.

After the user successfully selects an object, many issues must be managed. The first consideration is indicating the active status to the user. This is accomplished by drawing the object's handles.

Managing Object Handles

When an object is active, handles are drawn surrounding the object to reflect the object's status. The `gx_draw_handles` code found in Listing 16.5 draws the handles associated with a selected object.

Listing 16.5 Source Code for the `gx_draw_handles` Function

```
void gx_draw_handles( GXObjPtr obj )
{
  static unsigned char mask_bits[8][HNDL_SIZE]  = {
    {0x7f, 0x3f, 0x1f, 0x1f, 0x3f, 0x73, 0xe1, 0xc0},
    {0x18, 0x3c, 0x7e, 0xff, 0x18, 0x18, 0x18, 0x00},
```

continues

Listing 16.5 Continued

```
    {0xfe, 0xfc, 0xf8, 0xf8, 0xfc, 0xce, 0x87, 0x03},
    {0x10, 0x30, 0x70, 0xff, 0xff, 0x70, 0x30, 0x10},
    {0x03, 0x87, 0xce, 0xfc, 0xf8, 0xf8, 0xfc, 0xfe},
    {0x00, 0x18, 0x18, 0x18, 0xff, 0x7e, 0x3c, 0x18},
    {0xc0, 0xe1, 0x73, 0x3f, 0x1f, 0x1f, 0x3f, 0x7f},
    {0x08, 0x0c, 0x0e, 0xff, 0xff, 0x0e, 0x0c, 0x08} };

  static Boolean masks_created = False;
  static Pixmap  masks[8];

  GC  gc;
  int i;

  if( masks_created == False ) {
    for( i = 0; i < 8; i++ )
      masks[i] =
        XCreatePixmapFromBitmapData(XtDisplay(GxDrawArea),
                                    XtWindow(GxDrawArea),
                                   (char *)mask_bits[i],
                                    HNDL_SIZE, HNDL_SIZE,
                                    1, 0, 1 );

    masks_created = True;
  }

  if( obj && obj->handles && obj->num_handles > 0 ) {
    gc = gx_allocate_gc( obj, False );
    XSetFillStyle( XtDisplay(GxDrawArea), gc, FillSolid );

    if( obj->num_handles != 8 ) {
      fprintf( stderr,
       "Don't know how to draw objects with %d handles\n",
               obj->num_handles );
    } else {
      for( i = 0; i < 8; i++ ) {
        XSetClipOrigin( XtDisplay(GxDrawArea), gc,
                        obj->handles[i].x,
                        obj->handles[i].y );
        XSetClipMask(XtDisplay(GxDrawArea), gc, masks[i]);

        XFillRectangle( XtDisplay(GxDrawArea),
                        XtWindow(GxDrawArea), gc,
                        obj->handles[i].x,
                        obj->handles[i].y,
                        obj->handles[i].width,
                        obj->handles[i].height );
      }
    }
    XtReleaseGC(GxDrawArea, gc);
  }
}
```

The gx_draw_handles function found in Listing 16.5 begins with the definition of an array of bitmapped character data:

```
static unsigned char mask_bits[8][HNDL_SIZE]  = {
  {0x7f, 0x3f, 0x1f, 0x1f, 0x3f, 0x73, 0xe1, 0xc0},
  {0x18, 0x3c, 0x7e, 0xff, 0x18, 0x18, 0x18, 0x00},
  {0xfe, 0xfc, 0xf8, 0xf8, 0xfc, 0xce, 0x87, 0x03},
  {0x10, 0x30, 0x70, 0xff, 0xff, 0x70, 0x30, 0x10},
  {0x03, 0x87, 0xce, 0xfc, 0xf8, 0xf8, 0xfc, 0xfe},
  {0x00, 0x18, 0x18, 0x18, 0xff, 0x7e, 0x3c, 0x18},
  {0xc0, 0xe1, 0x73, 0x3f, 0x1f, 0x1f, 0x3f, 0x7f},
  {0x08, 0x0c, 0x0e, 0xff, 0xff, 0x0e, 0x0c, 0x08} };
```

Each element corresponds to an arrow indicating the direction that the handle will scale the object if this handle is selected. Because all the handles are based on a square, the HNDL_SIZE directive indicates the length of the handle bit data array.

It is defined in the gxGraphics.h header file as

```
#define HNDL_SIZE   8
```

A Pixmap created from each of the elements of bit data defined by the array mask_bits need only be created once. To manage this a static flag

```
static Boolean masks_created = False;
```

is used to ensure that only with the first call to this function the Pixmaps are created:

```
masks[i] =
  XCreatePixmapFromBitmapData(XtDisplay(GxDrawArea),
                              XtWindow(GxDrawArea),
                             (char *)mask_bits[i],
                              HNDL_SIZE, HNDL_SIZE,
                              1, 0, 1 );
```

Notice the last parameter passed to the XCreatePixmapFromBitmapData is a value of 1. Because this parameter specifies the depth of the Pixmap created, the result is a Pixmap with a depth of 1, which is synonymous with a Bitmap and satisfies the definition of a valid ClipMask.

After the Pixmaps have been created, a test is performed to ensure that the object's handles have been created and in fact the object is active:

```
if( obj && obj->handles && obj->num_handles > 0 ) {
```

If there are handles to be drawn, a Graphics Context is created using the gx_allocate_gc utility introduced in Chapter 17, "Utilities and Tools":

```
  gc = gx_allocate_gc( obj, False );
```

Because the creation of the Graphics Context is based on the attributes of the object specified as the first parameter of the creation utility, the `FillStyle` must explicitly be set to `FillSolid` for these handles to appear correctly:

```
XSetFillStyle( XtDisplay(GxDrawArea), gc, FillSolid );
```

Next, a test is performed to ensure the correct number of handles has been specified for the object:

```
if( obj->num_handles != 8 ) {
```

The test of the `num_handles` is necessary to prevent a potentially fatal error occurring when less than eight handles are created for the object and the subsequent `for` loop expects to index eight elements of the `handles` array:

```
  for( i = 0; i < 8; i++ ) {
```

One further change to the Graphics Context employed for drawing the handles and the routine is complete. Specifically, the assignment of a `ClipMask` corresponding to the shape of the arrow `Pixmap` created from the bitmap data defined by the `mask_bits` array must be assigned to the `gc`. Always when using a `ClipMask` a `ClipOrigin` must be specified to align the mask with the item being drawn:

```
    XSetClipOrigin( XtDisplay(GxDrawArea), gc,
                    obj->handles[i].x,
                    obj->handles[i].y );
    XSetClipMask(XtDisplay(GxDrawArea), gc, masks[i]);
```

The origin specified for the arrow `Pixmap` used as the `ClipMask` is the same as the location for the position of the handle being drawn.

Finally, the individual handle is drawn using the `XFillRectangle` graphic primitive:

```
    XFillRectangle( XtDisplay(GxDrawArea),
                    XtWindow(GxDrawArea), gc,
                    obj->handles[i].x,
                    obj->handles[i].y,
                    obj->handles[i].width,
                    obj->handles[i].height );
```

As a result of to the assignment of the `ClipMask` to the Graphics Context, the rectangle is drawn only where the bits of the arrow `Pixmap` are set to 1. In other words, the item being drawn is clipped (not drawn) where there is not a bit set in the `ClipMask`.

Finally, a call to return the Graphics Context to the pool cached by the X Toolkit is called to complete the function:

```
XtReleaseGC(GxDrawArea, gc);
```

To erase the handles associated with an object, the `gx_erase_handles` function shown in Listing 16.6 is used.

Listing 16.6 Source Code for the `gx_erase_handles` Function

```
1:    void gx_erase_handles( GXObjPtr obj )
2:    {
3:      GC gc;
4:
5:      if( obj && obj->handles && obj->num_handles > 0 ) {
6:        gc = gx_allocate_gc( obj, True );
7:
8:        XFillRectangles( XtDisplay(GxDrawArea),
9:                         XtWindow(GxDrawArea), gc,
10:                        obj->handles, obj->num_handles );
11:
12:       XtReleaseGC( GxDrawArea, gc );
13:     }
14:   }
```

A little simpler than the `gx_draw_handles` function, `gx_erase_handles` must only ensure the correct number of handles are assigned to the object

```
5:      if( obj && obj->handles && obj->num_handles > 0 ) {
```

and create a Graphics Context using the `gx_allocate_gc` utility:

```
6:        gc = gx_allocate_gc( obj, True );
```

Notice a difference in the invocation of the `gx_allocate_gc` utility for the `gx_draw_handles` and `gx_erase_handles` functions.

The second parameter for this function indicates whether the `gc` should be created with a background tile assigned. A `False` as passed from the `gx_draw_handles` function is used to draw objects to the canvas and a `True` is used to erase them.

➔ This function is presented in Chapter 17, "Utilities and Tools," in the section "Graphics Context Tiling," page 348, where the use of a background tile for erasing objects is discussed in detail.

Having looked at the first level of management required for objects activated by the user, namely managing the handles associated with the object, the discussion continues in the next section to managing the object's status.

Managing the Status of an Object

The behavior and treatment of a graphic object is influenced by the object's status. The user's navigation of the drawing area and specifically the explicit selection of objects that the user has created affect this status.

The process of selecting an object begins by the user clicking the mouse cursor while it is positioned over a graphic object. The Graphics Editor application must then apply the `ButtonPress` event to all the objects currently known to it in order to determine which object was targeted by the user's actions.

The source code that is found in Listing 16.7 for the `find_graphic` function begins the process of applying an event to the objects managed by the editor.

Listing 16.7 Source Code for the `find_graphic` Function

```
1:    static void find_graphic( XEvent *event )
2:    {
3:      GXObjPtr gx_obj = NULL;
4:
5:      if( event->type == ButtonPress ) {
6:        parse_all_objects( gxObjHeader, event, &gx_obj );
7:
8:        if( gx_obj ) {
9:          /* set this object as selected */
10:         activate_obj( gx_obj );
11:
12:       } else {
13:         /* deselect any currently selected */
14:         deactivate_objs();
15:       }
16:     }
17:   }
```

The function ensures that the event under consideration is a `ButtonPress` event:

```
5:      if( event->type == ButtonPress ) {
```

This is necessary to prevent events such as `MotionNotify` from erroneously selecting the graphic objects.

If the event type is correct, the function `parse_all_objects` introduced in Listing 16.4 is called to traverse the linked list of objects, invoking each object's `find` method in an attempt to discover any object that might have been targeted for selection:

```
6:        parse_all_objects( gxObjHeader, event, &gx_obj );
```

If an object is returned from the `parse_all_objects` function, the object is activated:

```
8:        if( gx_obj ) {
9:          /* set this object as selected */
10:         activate_obj( gx_obj );
```

The source code for the `activate_obj` function is found in Listing 16.9.

Otherwise, if the event did not successfully return from any known object's `find` method, all objects are deactivated:

```
14:         deactivate_objs();s
```

Listing 16.8 shows the source code for the `deactivate_objs` function.

Listing 16.8 Source Code for the `deactivate_objs` Function

```
1:    static void deactivate_objs( void )
2:    {
3:      GXObjPtr obj;
4:
5:      if( gxObjCurrent ) {
6:        obj = gxObjCurrent;
7:
8:        (*obj->deselect)( obj );
9:      }
10:   }
```

The `deactivate_objs` function in Listing 16.8 is absolute. For any object currently active in the application, the `deselect` method is invoked. When the `deactivate_objs` function completes, there will be no active objects in the Graphics Editor application and the value of `gxObjCurrent` will be `NULL`.

Listing 16.9 Source Code for the `activate_obj` Function

```
1:    static void activate_obj( GXObjPtr obj )
2:    {
3:      if( gxObjCurrent ) deactivate_objs();
4:
5:      /* ensure we have an object */
6:      if( obj ) {
7:        gxObjCurrent = obj;
8:        (*obj->select)( obj );
9:      }
10:   }
```

The `activate_objs` function begins by considering that another object can be currently active:

```
3:      if( gxObjCurrent ) deactivate_objs();
```

If this is the case, the `deactivate_objs` function is invoked to toggle the object's status to inactive.

After ensuring that a valid object reference was passed to the `activate_objs` function

```
6:      if( obj ) {
```

the `gxObjCurrent` value is set to equal the newly found object. The object's `select` method is invoked to create the object's handles and set its status flag `selected` to `True`.

The user's selection of a graphic object from the Graphics Editor's canvas begins with the user's navigation of the interface. The following section demonstrates how the user's navigation actions are processed by the Graphics Editor application.

Processing User Navigation of Objects

Recall from Chapter 13, in the section “Setting Up a Canvas,” page 296, that when we established the layout for the Graphics Editor’s graphical user interface we assigned certain event handlers to the canvas area of the application.

Specifically, an event handler called `drawAreaEventProc` was assigned for the events `PointerMotion`, `ButtonPress`, and `ButtonRelease`. Listing 16.10 shows the source code for this event handler and its management of the different event types it receives.

Listing 16.10 Source Code for the `drawAreaEventProc` Function

```
1:    void drawAreaEventProc( Widget w, XtPointer cd,
2:                            XEvent *event, Boolean flag )
3:    {
4:      if( draw_mgr_func != NULL ) {
5:        (*draw_mgr_func)( event );
6:      } else {
7:        process_event( event );
8:      }
9:    }
```

The function is quite simple in that it only has to determine which branch of the application the event is targeted for.

If one of the drawing functions `gx_line`, `gx_pencil`, `gx_box`, `gx_arrow`, `gx_arc`, or `gx_text` is currently active, its value will be assigned to the function pointer `draw_mgr_func` as described in Chapter 13, in the section “Laying Out the User Interface,” page 265.

An active drawing function gets preferential treatment

```
4:      if( draw_mgr_func != NULL ) {
```

and the event received by the `drawAreaEventProc` is sent to this active function:

```
5:        (*draw_mgr_func)( event );
```

Otherwise, the `process_event` function that is found in Listing 16.11 is called to screen the event for the intended user action.

Listing 16.11 Source Code for the `process_event` Function

```
1:    static void process_event( XEvent *xe )
2:    {
3:      if( gxObjCurrent == NULL ) {
4:        if( xe && xe->type == ButtonPress ) {
5:          find_graphic( xe );
6:        }
7:      } else {
8:        /* update the object */
9:        update_obj( gxObjCurrent, xe );
10:     }
11:   }
```

The `process_event` function has two responsibilities. First, in the absence of a currently selected object, is to direct the event to the `find_graphic` function introduced in Listing 16.6:

```
5:          find_graphic( xe );
```

However, if an object is already selected, the event is targeted for updating the active object by a call to `update_obj`:

```
9:        update_obj( gxObjCurrent, xe );
```

The `update_obj` function must screen based on event type to determine the nature of object update required. Listing 16.12 shows the source code for the `update_obj` function.

Listing 16.12 Source Code for the `update_obj` Function

```
1:    static void update_obj( GXObjPtr obj, XEvent *event )
2:    {
3:      switch( event->type ) {
4:      case ButtonPress:
5:        buttonpress_update( obj, event );
6:        break;
7:
8:      case ButtonRelease:
9:        buttonrelease_update( obj, event );
10:       break;
11:
12:      case MotionNotify:
13:        motionnotify_update( obj, event );
14:        break;
15:     }
16:   }
```

The `update_obj` function only has to switch on the event type to determine which update function to call. Listing 16.13 shows the source code for the first case `buttonpress_update`.

Listing 16.13 Source Code for the `buttonpress_update` Function

```
1:    static void buttonpress_update( GXObjPtr obj,
2:                                    XEvent *event )
3:    {
4:      Boolean  found = False;
5:      GXObjPtr e_obj = NULL;
```

continues

Listing 16.13 Continued

```
6:      int      indx  = 0;
7:
8:      if( obj->action == NULL ) {
9:        FixedX = event->xbutton.x;
10:       FixedY = event->xbutton.y;
11:
12:       find_handle( obj, event, &found, &indx );
13:
14:       if( found ) {
15:         obj->action = obj->scale;
16:
17:         set_cursor( SCALE_MODE );
18:         GxActiveHandle = indx;
19:
20:       } else {
21:         parse_all_objects( obj, event, &e_obj );
22:         if( e_obj == obj ) {
23:           obj->action = obj->move;
24:           set_cursor( MOVE_MODE );
25:         } else {
26:           deactivate_objs();
27:         }
28:       }
29:     }
30:
31:     if( obj->action ) {
32:       (*obj->deselect)( obj );
33:       (*obj->action)( obj, event );
34:     }
35:   }
```

The `buttonpress_update` function has the ultimate purpose of correctly assigning an `action` to the current object.

Two actions are possible, moving or scaling. The determination of the correct `action` assignment is based on whether the event occurred on one of the handles assigned to the active object or, alternately, whether the event occurred on the object itself.

The `buttonpress_update` function begins by testing for a current action assigned to the active object:

```
8:      if( obj->action == NULL ) {
```

If a current `action` is assigned, the body of the `if` statement is skipped and immediately the `action` is invoked for the current event:

```
31:     if( obj->action ) {
32:       (*obj->deselect)( obj );
33:       (*obj->action)( obj, event );
34:     }
```

The call to deselect the object before invoking the action

```
(*obj->deselect)( obj );
```

is to ensure that the handles are not displayed while the object is either scaled or moved.

If an object's `action` field does not have a current function assigned, the `buttonpress_update` function determines what the value should be.

Before determining the correct action assignment, the global variables

```
FixedX = event->xbutton.x;
FixedY = event->xbutton.y;
```

are updated with the x, y coordinates of the current event. This will be important during the scale action covered in Chapter 20 in the section "Scaling a Line Object," page 398.

To determine the proper value of the action field for the current object, the `buttonpress_update` determines whether one of the object's handles was successfully selected by the event being processed:

```
find_handle( obj, event, &found, &indx );
```

The source code for the `find_handle` routine is provided in Listing 16.14.

The `find_handle` function returns two values in the fields passed as the second and third parameters: a status flag indicating whether a handle was found and, optionally if `found` reflects that a handle was found, the index of the handle.

The successful selection of the object's handle indicates that the user intended to scale the object. In this case, the object's `scale` method is used as the value for the `action` field:

```
obj->action = obj->scale;
```

Two chores are addressed in preparation for the scale action. The first changes the cursor to reflect the new state of the application:

```
set_cursor( SCALE_MODE );
```

➔ This function is provided in Chapter 17, "Utilities and Tools," page 343.

```
GxActiveHandle = indx;
```

The second chore required in preparation for the scale action is to set the value of `indx` (reflecting the specific handle selected by the user) to the global variable `GxActiveHandle`.

This information is necessary to ensure that the object is scaled in a direction appropriate to the direction of the arrow of the selected object's handle.

If a handle was not selected, the body of the else is entered

```
    } else {
```

and the buttonpress_update function parses all known objects to see whether the event landed on an object:

```
      parse_all_objects( obj, event, &e_obj );
```

An event directed to the update branch of the Graphics Editor application that is located directly on the graphic object implies that the user intended for the object to be moved.

If the parse_all_objects function as shown in Listing 16.4 determines that an object is selected by the specified event, the object is returned in the last parameter field of the function call.

If the event selected the same object as the currently active object

```
      if( e_obj == obj ) {
```

the user's intent to move the object is confirmed and the action field is updated accordingly:

```
        obj->action = obj->move;
```

The utility to update the applications cursor to reflect the new state of the application is called:

```
        set_cursor( MOVE_MODE );
```

If the object selected by the event does not match the active object, the buttonpress_update function deactivates the current object and ends the update process:

```
      } else {
        deactivate_objs();
```

Another update function used by the update_obj routine manages the ButtonRelease event type. The source code for the buttonrelease_update function is found in Listing 16.14.

Listing 16.14 Source Code for the buttonrelease_update Function

```
static void buttonrelease_update( GXObjPtr obj,
                                  XEvent *event )
{
  /*
   * reset the update function
   */
7:      if( obj->action ) {
8:        (*obj->action)( obj, event );
9:        (*obj->action)( obj, NULL );
10:
11:       obj->action = NULL;
12:       (*obj->select)( obj );
13:       gx_refresh();
14:     }
15:
16:     set_cursor( NORMAL_MODE );
17:   }
```

The task managed by the `buttonrelease_update` function depends entirely upon a valid assignment to the `action` field of the object.

If the object has a current action assigned, the action function is invoked with the event being processed:

```
8:        (*obj->action)( obj, event );
```

By design, the event type is intended to end the iterative actions that affect an object.

Therefore, after the `ButtonRelease` event is passed to the current action function, the function is invoked a second time with a `NULL` event to allow the object to perform any necessary cleanup

```
9:        (*obj->action)( obj, NULL );
```

and the current action is cleared:

```
11:       obj->action = NULL;
```

Finally, the object's `select` method is invoked to redisplay the handles and the screen is refreshed:

```
12:       (*obj->select)( obj );
13:       gx_refresh();
```

The application's cursor is updated

```
16:     set_cursor( NORMAL_MODE );
```

reflecting the new state of the application.

The final update function employed by the `update_obj` routine manages the `MotionNotify` event type. The source code for the `motionnotify_update` function is found in Listing 16.15.

Listing 16.15 Source Code for the `motionnotify_update` Function

```
static void motionnotify_update( GXObjPtr obj,
                                 XEvent *event )
{
  if( obj->action ) {
    (*obj->action)( obj, event );
  }
}
```

As with the `buttonrelease_update` function, the `motionnotify_update` routine depends on the action assigned to the current object.

The notification of the mouse in motion is the heart of the iterative processing used to alter the current object. For this reason, if a current action is assigned to the object, the event is passed wholesale.

The final piece to the management of user navigation is the introduction of the `find_handle` routine used by the `buttonpress_update` routine found in Listing 16.13.

Listing 16.16 Source Code for the `find_handle` Function

```
1:    static void find_handle( GXObjPtr gx_obj, XEvent *xe,
2:                             Boolean *found, int *indx )
3:    {
4:      int i;
5:
6:      *found = False;
7:
8:      if( gx_obj && gx_obj->handles && !(*found)) {
9:        for( i = 0; (i < gx_obj->num_handles) && !(*found); i++ ) {
10:         if( ( xe->xbutton.x > gx_obj->handles[i].x )           &&
11:             ( xe->xbutton.x < gx_obj->handles[i].x + HNDL_SIZE) &&
12:             ( xe->xbutton.y > gx_obj->handles[i].y )           &&
13:             ( xe->xbutton.y < gx_obj->handles[i].y + HNDL_SIZE)) {
14:           *indx  = i;
15:           *found = True;
16:         }
17:       }
18:     }
19:   }
```

The `find_handle` function starts by ensuring that handles for this object exist and that a handle has yet to be found:

```
8:      if( gx_obj && gx_obj->handles && !(*found)) {
```

Then for each handle known to the object, a series of greater-than/less-than tests are performed to see whether the coordinates of the `XEvent` lie within bounds of the object's handle.

If the event's x component is greater than the starting point of the handle

```
    if( ( xe->xbutton.x > gx_obj->handles[i].x )            &&
```

and the x component is less than the end of the handle

```
        ( xe->xbutton.x < gx_obj->handles[i].x + HNDL_SIZE) &&
```

and the event's y component is greater than the start and less than the end of the handle

```
        ( xe->xbutton.y > gx_obj->handles[i].y )            &&
        ( xe->xbutton.y < gx_obj->handles[i].y + HNDL_SIZE)) {
```

the handle is considered selected. The `found` flag is updated to inform the calling function that a handle was found and the index of the satisfying handle is stored in the `indx` field:

```
      *indx  = i;
      *found = True;
```

If none of the object's handles pass the test, the `found` flag is returned to the calling function as `False`, indicating no handle was found and the value of the `indx` field is immaterial.

This completes the discussion of routines in the Graphics Editor that manage objects based on common object data structure without regard to the specifics of the object's type.

Next Steps

This chapter covered in detail the management routines that act on objects generically, that is, despite their actual type.

In Chapter 17, more utilities and tools are introduced which address areas of the Graphics Editor application necessary for object creation, management, and housekeeping.

Chapter 17

In this chapter

Utilities and Tools

Chapter 16 presented functions used by the Graphics Editor that employ the common or generic features of the graphic objects.

In this chapter, I introduce and discuss utility routines necessary to manage the graphical user interface as well as behind-the-scenes tool functions.

Common Object Creation

As a software developer, code redundancy should set off sirens in your analytical mind. For that reason, when an object is created in the Graphics Editor, a central `gx_create_obj` routine is used to create the object and assign the default attributes of the object rather than allowing each of the object-specific creation routines to repeat these necessary steps.

The function also enables the assignment of default methods to prevent an oversight in the object-specific code from resulting in a fatal bus error.

The source code in Listing 17.1 introduces the `gx_create_obj` function.

Listing 17.1 Source Code for the `gx_create_obj` Function for Inclusion in `gxGx.c`

```
1:    GXObjPtr gx_create_obj( void )
2:    {
3:      GXObjPtr gx_obj = XtNew( GXObj );
4:
5:      gx_obj->fs = None;
6:      gx_obj->ls = LineSolid;
7:      gx_obj->lw = 1;
8:
9:      gx_obj->bg = WhitePixelOfScreen(XtScreen(GxDrawArea));
10:     gx_obj->fg = BlackPixelOfScreen(XtScreen(GxDrawArea));
```

continues

Listing 17.1 Continued

```
11:
12:     gx_obj->handles     = NULL;
13:     gx_obj->num_handles = 0;
14:
15:     gx_obj->data = NULL;
16:
17:     gx_obj->draw     = (void    (*)())null_func;
18:     gx_obj->erase    = (void    (*)())null_func;
19:     gx_obj->find     = (Boolean (*)())null_func;
20:     gx_obj->select   = (void    (*)())null_func;
21:     gx_obj->deselect = (void    (*)())null_func;
22:     gx_obj->move     = (void    (*)())null_func;
23:     gx_obj->scale    = (void    (*)())null_func;
24:     gx_obj->copy     = (void    (*)())null_func;
25:
26:     gx_obj->action   = NULL;
27:
28:     gx_obj->next = NULL;
29:
30:     /* reset the draw_mgr_func so   */
31:     /* future events are applied to */
32:     /* existing objects             */
33:     draw_mgr_func = NULL;
34:
35:     return gx_obj;
36:   }
```

The `gx_create_obj` routine begins by allocating the memory required for the new object:

```
3:      GXObjPtr gx_obj = XtNew( GXObj );
```

With memory for the object created, the default attributes are assigned:

```
5:      gx_obj->fs = None;
6:      gx_obj->ls = LineSolid;
7:      gx_obj->lw = 1;
8:
9:      gx_obj->bg = WhitePixelOfScreen(XtScreen(GxDrawArea));
10:     gx_obj->fg = BlackPixelOfScreen(XtScreen(GxDrawArea));
```

The next part of the `gx_create_obj` function initializes fields of the common object data structure to a safe state:

```
12:     gx_obj->handles     = NULL;
13:     gx_obj->num_handles = 0;
14:
15:     gx_obj->data = NULL;
```

Then a temporary function is assigned to the method fields of the object to prevent an erroneous invocation of an invalid function:

```
gx_obj->draw     = (void    (*)())null_func;
gx_obj->erase    = (void    (*)())null_func;
gx_obj->find     = (Boolean (*)())null_func;
gx_obj->select   = (void    (*)())null_func;
gx_obj->deselect = (void    (*)())null_func;
gx_obj->move     = (void    (*)())null_func;
gx_obj->scale    = (void    (*)())null_func;
gx_obj->copy     = (void    (*)())null_func;
```

The fail safe `null_func` routine is shown in Listing 17.1a.

Listing 17.1a Source Code for the `null_func` Routine for Inclusion in `gxGx.c`

```
static void null_func( void )
{
  printf( "Warning: null function called!\n" );
}
```

Because the object is newly created, there is not a current action to assign to it. Therefore, set the action field to reflect this:

```
gx_obj->action   = NULL;
```

It is necessary to initialize the `next` field of the object structure that links to other objects. This is critical to the proper management of the linked list in which this object will be placed:

```
gx_obj->next = NULL;
```

The global variable `draw_mgr_func` is reset to `NULL` to ensure that subsequent events generated in the drawing area canvas are sent to the object that is created and set as active:

```
draw_mgr_func = NULL;
```

Finally, the new object is returned to the calling function:

```
return gx_obj;
```

Though the creation routines for the object-specific data structures are not introduced until Chapter 20, "`Latex Line` Object," you should understand (extrapolate) that the graphic object's specific creation routines employ the `gx_create_obj` routine.

Further, upon receiving the object created by the `gx_create_obj` function, the object-specific data is assigned to the `data` field of the object and the appropriate object's methods are updated to properly manage that data field.

For the editor to retain and continually manage the object, it must be added to the linked list of objects under the editor's control. The following section describes the linked list management function that adds nodes to the editor's linked list.

Linked List Management

➔The Excursion “What is a Linked List?” found in Chapter 15, “Common Object Definition,” describes a *linked list* as a programming construct that enables *nodes* (data structures) to be linked together.

Designating a head node as the first element in the list and continually assigning consecutive node references to the `next` field of the last node in the list accomplishes this.

The `gx_add_obj` routine is responsible for finding the end of the list (last node in the list) and assigning the reference to the node being added to the `next` element of that object structure.

Listing 17.2 shows the source code for `gx_add_obj`.

Listing 17.2 Source Code for the `gx_add_obj` Function for Inclusion in `gxGx.c`

```
1:    void gx_add_obj( GXObjPtr obj )
2:    {
3:      GXObjPtr gx_obj;
4:
5:      if( gxObjHeader == NULL ) {
6:        gxObjHeader = obj;
7:      } else {
8:        gx_obj = gxObjHeader;
9:
10:       /* find the end of the object list */
11:       while( gx_obj->next != NULL ) {
12:         gx_obj = gx_obj->next;
13:       }
14:
15:       /*
16:        * add the new object to the end of our list
17:        */
18:       gx_obj->next = obj;
19:     }
20:   }
```

The `gx_add_obj` function begins by determining whether a list currently exists by testing the value of the `gxObjHeader` variable responsible for maintaining the list’s head:

```
5:      if( gxObjHeader == NULL ) {
```

If the `gxObjHeader` variable is `NULL`, this is the first object created by the Graphics Editor, and it therefore assumes the responsibility of being the head of the list:

```
6:        gxObjHeader = obj;
```

If a valid head already exists, however, the list must be traversed in search of its end.

To do this, a variable is assigned the value of the list head and a `while` loop is entered until the `next` element's value is `NULL`, as `NULL` marks the end of a linked list:

```
    gx_obj = gxObjHeader;

    /* find the end of the object list */
    while( gx_obj->next != NULL ) {
```

As long as there is a valid element assigned to the `next` field of the current node, traversal continues

```
      gx_obj = gx_obj->next;
```

When the end of the linked list is found, the element being added assumes a position as the new end of the list:

```
    gx_obj->next = obj;
```

The capability to create and retain graphic objects is made more meaningful when the Graphics Editor can draw those objects to the canvas.

In the next section I introduce a utility function used by the editor to create a Graphics Context based on the current attribute settings of an object.

Creating a Graphics Context

The `gx_allocate_gc` function introduced in Listing 17.3 is a cornerstone of the Graphics Editor's functionality and assigns the `XtGCValues` fields based on the attribute settings of the object specified as the function's first parameter. With a properly constructed `XtGCValues` structure, a Graphics Context is created to honor the object's settings.

Listing 17.3 Source Code for the `gx_allocate_gc` Function for Inclusion in `gxGx.c`

```
GC gx_allocate_gc( GXObjPtr obj, Boolean tile )
{
  GC gc;

  XGCValues values;
  XtGCMask  mask = 0L;

  values.foreground = obj->fg;
  mask |= GCForeground;

  values.background = obj->bg;
  mask |= GCBackground;

  values.line_width = obj->lw;
  mask |= GCLineWidth;
```

continues

Listing 17.3 Continued

```

    values.line_style = obj->ls;
    mask |= GCLineStyle;

    if( tile ) {
      values.tile = GxDrawAreaBG;
      mask |= GCTile;

      values.fill_style = FillTiled;
      mask |= GCFillStyle;
    }

    values.function = GXcopy;
    mask |= GCFunction;

    gc = XtAllocateGC(GxDrawArea, 0, mask, &values, mask, 0);
    return gc;
  }
```

The `gx_allocate_gc` function is straightforward in the creation of a Graphics Context except for the use of the second parameter called `tile`, described in the following section.

Graphics Context Tiling

The value of the `tile` flag passed as the second parameter to the `gx_allocate_gc` function is specified by the caller to indicate whether the `gc` created by this routine is intended for a draw or an erase action.

If the value of `tile` is specified as `True`, the Graphics Context is created to *tile* the background into the draw request. The effect of this is that the background of the canvas replaces the foreground pixels of the X graphic primitive request.

The net result of the Graphics Context *tile* behavior is to effectively *erase* the item rendered with this `gc` from the canvas. I say *effectively* because you've still drawn the item, albeit not using the foreground color, but rather by applying the background to the pixels that would normally have been set to the foreground color value.

For the tile behavior to work correctly, a tile `Pixmap` must first be created. Therefore, I've added the lines shown in Listing 17.4 to the beginning of the `create_canvas` routine introduced in Chapter 13, "Application Structure."

Listing 17.4 Changes to Be Made to the `create_canvas` Routine in `gxGraphics.c`

```
Widget create_canvas( Widget parent )
{
  GxDrawAreaBG =
    XCreatePixmapFromBitmapData(XtDisplay(parent),
                                DefaultRootWindow(XtDisplay(parent)),
                                grid_bits, grid_width, grid_height,
                                BlackPixelOfScreen(XtScreen(parent)),
                                WhitePixelOfScreen(XtScreen(parent)),
                                DefaultDepthOfScreen(XtScreen(parent)));
```

This code fragment creates a `Pixmap` using the character bitmap data `grid_bits` shown in Listing 17.5.

Listing 17.5 Contents of the `grid.xbm` File

```
#define grid_width 10
#define grid_height 10
static unsigned char grid_bits[] = {
   0x00,0x00,0x00,0x00,0x49,0x02,0x00,0x00,0x00,0x00,
   0x00,0x02,0x00,0x00,0x00,0x00,0x00,0x02,0x00,0x00};
```

To be syntactically correct and prevent compiler errors associated with the changes shown in Listing 17.4, you must create the `grid.xbm` file from Listing 17.5 and place this file in the `src/include` directory.

Further, an `include` directive must be added to the beginning of `gxGraphics.c` for the `grid.xbm` file as follows:

```
#include "gxGraphics.h"
#include "grid.xbm"
```

Finally, the following line must be added to the `GLOBAL` section of the `gxGraphic.h` header file:

```
GLOBAL Pixmap GxDrawAreaBG;
```

The next section introduces how to alter the appearance of the applications cursor to reflect the status or state of the application.

Using the Cursor as State Indicator

The default cursor for the X Window Environment is the left arrow cursor. This is fine for normal operation but can be changed at will to reflect any of the number of states the application might enter.

The cursors available to an X-based application are found in the header file `cursorfonts.h`. The code in Listing 17.6 shows the Graphics Editor utility for changing the cursor based on the cursor modes known to the application.

Listing 17.6 Source Code for the `set_cursor` Function to Be Added to `gxGraphics.c`

```
void set_cursor( CursorMode mode )
{
   Cursor new_cursor;

  switch( mode ) {
    case LINE_MODE:
      new_cursor = gxCrosshair;
      break;

    case PENCIL_MODE:
      new_cursor = gxPencil;
      break;

    case EDIT_MODE:
      new_cursor = gxEdit;
      break;

    case TEXT_MODE:
      new_cursor = gxTextI;
      break;

    case SCALE_MODE:
      new_cursor = gxScale;
      break;

    case MOVE_MODE:
      new_cursor = gxMove;
      break;

    case NORMAL_MODE:
    default:
      new_cursor = gxNormal;
      break;
  }
  XDefineCursor( XtDisplay(GxDrawArea),
                 XtWindow(GxDrawArea), new_cursor );
  XFlush( XtDisplay(GxDrawArea) );
}
```

The required parameter for the `set_cursor` function is of the data type `CursorMode`. This is a data type unique to the Graphics Editor and is defined as the enumeration shown in Listing 17.7.

Listing 17.7 The `CursorMode` Enumeration for Inclusion in the `gxGraphics.h` Header File

```
typedef enum _cursor_mode {
  NORMAL_MODE = 0,
  PENCIL_MODE,
  EDIT_MODE,
  TEXT_MODE,
  MOVE_MODE,
  SCALE_MODE,
  LINE_MODE
} CursorMode;
```

The `set_cursor` function switches on the value of the `CursorMode` specified to the function and sets the `new_cursor` value to one of the global variables shown in Listing 17.8.

Listing 17.8 Global Cursor References for Inclusion in the `gxGraphics.c` Source File

```
Cursor gxCrosshair;
Cursor gxPencil;
Cursor gxEdit;
Cursor gxNormal;
Cursor gxTextI;
Cursor gxMove;
Cursor gxScale;
```

After the `new_cursor` value is determined, the cursor's corresponding cursor value is defined for the `Window` of the `GxDrawArea` widget

```
XDefineCursor( XtDisplay(GxDrawArea),
               XtWindow(GxDrawArea), new_cursor );
```

and the `expose` event generated by the `XDefineCursor` function is flushed from the application's event queue to ensure that the new cursor appears immediately in the drawing area window:

```
XFlush( XtDisplay(GxDrawArea) );
```

The `Cursor` variables shown in Listing 17.8 require values well in advance of the application employing them, so a new function called `initializeGX` has been added to the `gxGraphics.c` source file.

The contents of the `initializeGX` function are found in Listing 17.9. The function's purpose is to initialize all global variables used by the application in addition to the cursor variables in Listing 17.8.

Listing 17.9 Source Code for the `initializeGX` Function for the `gxGraphics.c` File

```
1:    void initializeGX( void )
2:    {
3:      Pixel color;
4:      Display *dsp = XtDisplay(GxDrawArea);
5:
6:      XtVaGetValues( GxStatusBar, XtNbackground, &color, NULL );
7:
8:      gxObjHeader  = NULL;
9:      gxObjCurrent = NULL;
10:
11:     gxCrosshair = XCreateFontCursor(dsp, XC_crosshair);
12:     gxPencil    = XCreateFontCursor(dsp, XC_pencil);
13:     gxEdit      = XCreateFontCursor(dsp, XC_cross);
14:     gxTextI     = XCreateFontCursor(dsp, XC_xterm);
15:     gxScale     = XCreateFontCursor(dsp, XC_dotbox);
16:     gxMove      = XCreateFontCursor(dsp, XC_hand1);
17:     gxNormal    = XCreateFontCursor(dsp, XC_top_left_arrow);
18:
19:     rubberGC = XCreateGC(XtDisplay(GxDrawArea),
20:                DefaultRootWindow(XtDisplay(GxDrawArea)), 0, NULL );
21:
22:     XSetForeground( XtDisplay(GxDrawArea), rubberGC, color );
23:     XSetFunction( XtDisplay(GxDrawArea), rubberGC, GXxor );
24:
25:     XSetWindowBackgroundPixmap(XtDisplay(GxDrawArea),
26:                                XtWindow(GxDrawArea), GxDrawAreaBG);
27:
28:     FixedX = FixedY = OrigX = OrigY = ExntX = ExntY = 0;
29:   }
```

The function starts by extracting the background color of the status label for use later in the function when the global `rubberGC` is defined:

```
6:      XtVaGetValues( GxStatusBar, XtNbackground, &color, NULL );
```

The widget specified for determining the background color of the application is arbitrary.

The function continues by ensuring that the head of the linked list used to manage the objects known to the editor and the pointer to the currently active object are set to `NULL`:

```
8:      gxObjHeader  = NULL;
9:      gxObjCurrent = NULL;
```

Following this, the cursor values used to reflect the state of the application are created:

```
11:     gxCrosshair = XCreateFontCursor(dsp, XC_crosshair);
12:     gxPencil    = XCreateFontCursor(dsp, XC_pencil);
```

```
13:     gxEdit      = XCreateFontCursor(dsp, XC_cross);
14:     gxTextI     = XCreateFontCursor(dsp, XC_xterm);
15:     gxScale     = XCreateFontCursor(dsp, XC_dotbox);
16:     gxMove      = XCreateFontCursor(dsp, XC_hand1);
17:     gxNormal    = XCreateFontCursor(dsp, XC_top_left_arrow);
```

To provide the compiler with the definition of the cursors created in lines 11–17, the header file `cursorfonts.h` must be added to the beginning of `gxGraphics.c` source file:

```
    #include <X11/Xaw/Box.h>
    #include <X11/cursorfont.h>
```

Following the creation of these cursors, they are available for use during execution of the application.

The `initializeGX` function next creates the global `rubberGC` used by the object creation routines to provide a rubber-banding effect during object creation and update routines:

```
19:     rubberGC = XCreateGC(XtDisplay(GxDrawArea),
20:                DefaultRootWindow(XtDisplay(GxDrawArea)), 0, NULL );
```

The color extracted from the status label when the function began is assigned as the foreground color of the `rubberGC` Graphics Context

```
22:     XSetForeground( XtDisplay(GxDrawArea), rubberGC, color );
```

and the function of the Graphics Context is set to the `GXxor` value, which enables the rubber-banding behavior and ensures that items drawn using this `GC` do not affect the drawing area permanently:

```
23:     XSetFunction( XtDisplay(GxDrawArea), rubberGC, GXxor );
```

Finally, several global variables used to manage the object's scale, move, and rotate functions are initialized:

```
28:     FixedX = FixedY = OrigX = OrigY = ExntX = ExntY = 0;
```

Although use of these variables is not introduced until Chapter 20, their definitions can be added to the `GLOBAL` section of `gxGraphics.h`

```
GLOBAL Widget GxStatusBar;
GLOBAL Widget GxDrawArea;
GLOBAL Pixmap GxDrawAreaBG;
GLOBAL GC     rubberGC;
GLOBAL int    GxActiveHandle;

GLOBAL GXObjPtr gxObjHeader;
GLOBAL GXObjPtr gxObjCurrent;
GLOBAL int    FixedX, FixedY;
GLOBAL int    OrigX,  OrigY;
GLOBAL int    ExntX,  ExntY;
```

and they will be ready for use when the functions, which require them, are introduced.

Before completing the introduction of the `initializeGX` function, it is important that a call to it is added at the appropriate phase of the application startup. Listing 17.10 shows the proper placement for a call to `initializeGX` function.

Listing 17.10 Changes Necessary to the Function `Main` in `gxMain.c` for Invoking the Function `initalizeGX`

```
  .
  .
  .
    XtRealizeWidget( toplevel );
    initializeGX();

    XtAppMainLoop( appContext );
```

This completes a discussion of tools and utilities used by the Graphics Editor application. These routines work to create the common portion of the editor objects, to add objects to the management list, to create `GC`s to reflect the attributes of an object (and optionally prepare that `GC` for erasing), as well as using the program's cursor to reflect the state of the editor.

Next Steps

In the next chapter, I will introduce the concept of file formats in preparation for constructing the Save and Restore functions used by the Graphics Editor.

Chapter 18

In this chapter

- *Understanding Files*
- *Binary File Formatting*
- *ASCII File Formatting*
- *Magic Numbers*
- *Next Steps*

File Formats

As the Graphics Editor project advances in capability, it is increasingly more important for users to be able to save and subsequently restore the graphic objects they work so hard to create, place, and modify during a session of the editor application.

This chapter introduces the concepts and principles governing file formatting as a segue to adding the Save and Load functions used by the Graphics Editor.

Note

The most important concept to draw from this chapter is that the programmer who authors the application that will employ a file decides the format of the data placed in that file.

This can seem a hasty absolute as I point out that many applications understand many file formats. However, as I introduce the concepts relating to file formats, consider first that the decision of how to format a file starts with a programmer.

Several considerations guide the decision of how a file is formatted, or how the data targeted for a file is arranged.

For example, one consideration is *size*, dictating whether compression should be used in the file format. If a large amount of information is intended for the file contents and these files are to be transmitted across a network, you should make the file as small as possible.

Another consideration is *speed*. Files of considerable size that are formatted in a plain text or *ASCII* format take longer to parse and load than files formatted using a *binary* method.

These considerations direct the decision of the programmer in determining how the data in a file will be organized.

Note

As the save and load features of the Graphics Editor are implemented, be cognizant that only the Graphics Editor application will be able to read the files formatted for a task.

By read I mean, of course, create graphic objects from the data contained in the file that can be moved, scaled, rotated, and so forth.

However, as its popularity grows to worldwide renown, other graphic editors might want to adopt the file format. This probably won't occur for our application, but I need a humorous way to demonstrate that file formats are generally unique to the application programmer who conceives them.

Clearly, the more commonplace and popular the application becomes, the more likely that other applications will adopt some level of support for their file format. But it is not automatic.

In other words, not all graphic programs understand multiple graphic file formats, just as not all word processors understand documents formatted by other word processing applications.

For this reason, a unique suffix or file extension distinguishes the file type and the application meant to interpret it.

When all the decisions are made and the file format decided, it cannot be arbitrarily changed. Any changes to the placement of the data within the file or the characteristics of the data will have a direct and significant impact on the application that reads and writes it.

In other words, if a new data field is added to the file, the application's functions that write and read the file must be altered to support the new field.

Furthermore, a decision must be made as to whether the previous file format (before the addition of the data field) will still be supported.

If support for the original format is continued, the application's author must account programmatically for determining which format a file passed to the application is in and the action required to read it.

Emphasizing the impact that even a small change to the format of a file has on the application leads me back to the statement made earlier: The programmer who authors the application that will employ a file decides the format of the data placed in that file.

Acknowledging the impact on the application for which the file is targeted, consider for a moment the significance of a format change if that file format is understood by multiple applications.

The following sections will address creating files using both ASCII and binary formatting techniques. First, however, let us be sure of what a file is.

Understanding Files

In data processing, a file is a collection of *records*. For example, all the information one might have on his customers would be ideal for storing in a file.

Each customer record would consist of *fields* for individual data elements. These data elements could include items such as the customer's name, identification number, address, and so forth.

By placing all information within a record in exactly the same location for all fields for each of the customer records, the organization of the file will be uniform and easily manipulated by a computer program.

This example can be instantly paralleled to the data requirements of the Graphics Editor project. Each object created in the editor will have common data fields, which must be included in the data set saved to the file created for the purpose. However, the task is complicated in that there are non-uniform data elements that also must be included in the file. These non-uniform elements are the data-specific fields of the various unique objects supported by the editor.

The last point to be made is the capability to distinguish file formats. Some applications describe files with given formats by assigning them a particular filename suffix. (The filename suffix is also known as a *filename extension*.)

For example, a program or executable file is sometimes given or required to have an `.exe` suffix. In general, the suffixes tend to be as descriptive of the formats as they can be within the limits of the number of characters allowed for suffixes by the operating system.

Binary File Formatting

Writing a file using a binary file format refers to writing the file using the binary numbering system, which uses only 0 and 1.

This is significant because the conventions used to examine the contents of a file (text editors, word processors, and so forth) cannot understand the file's contents. This is true because *character encoding* (meaning the arrangement of characters) is typically based on 8-bit groupings.

A binary data file, however, will not (necessarily) write 8 bits for each data element it places in the file. Consider the PCX file header format shown in Listing 18.1.

Listing 18.1 PCX File Header Format

```
Byte     Item          Size     Description/Comments
0        Manufacturer   1       Constant Flag 10 = ZSoft .PCX1 Version 1
                                Version information:
                                0 = Version 2.5
                                2 = Version 2.8 w/palette information
                                3 = Version 2.8 w/o palette information
                                5 = Version 3.0
2        Encoding       1       1 = .PCX run length encoding
3        Bits per pixel 1       Number of bits/pixel per plane
4        Window         8       Picture Dimensions
                                (Xmin, Ymin) - (Xmax - Ymax)
                                in pixels, inclusive
12       HRes           2       Horizontal Resolution of creating
                                device
14       VRes           2       Vertical Resolution of creating device
16       Colormap       48      Color palette setting, see text
64       Reserved       1
65       NPlanes        1       Number of color planes
66       Bytes per Line 2       Number of bytes per scan line per color
                                plane (always even for .PCX files)
68       Palette Info   2       How to interpret palette -
                                1 = color/BW,
                                2 = grayscale
70       Filler         58      blank to fill out 128 byte header
```

What is most interesting about Listing 18.1 is the `SIZE` column reflecting the size of the data field written for the data entities.

The first three fields written to a `.pcx` file are the Manufacturer, Encoding, and Bits per pixel fields necessary to interpret the image data contained in the file. Each of these data fields is only 1 bit in size. The fourth field placed in the file is the Window element and is 8 bits in size.

Therefore, if a `.pcx` file were loaded into a text editor, the first 3 bits (corresponding to the first three data elements) and the first 5 bits of the fourth data element would be interpreted by the editor as a character.

Depending on the values of these elements, the bits can map to a printable character. If not, the text editor will attempt to display a non-printable encoding of these 8 bits.

A discussion of the attempt to encode binary data into character data does not introduce a file format consideration, but rather draws a contrast in data organization between ASCII and binary formats.

What does serve as a consideration when choosing a binary file format is the complexity required to parse the file because of its cryptic format: not that it will necessarily be cryptic to the developer, but rather to those who adopt support of it later.

An advantage of the binary file format is that the size of the file is minimized as the absolute required data elements (down to the single bit value) are written to the file. Generally, there is very little padding, as shown in the last field of the `.pcx` file header.

As a caution when choosing the binary file format, the mechanism used to create the file can be governed by the same limitations as binary object files introduced in Chapter 1, "UNIX for Developers," in the section "Object Files," page 19.

For example, if the common object data structure introduced in Chapter 15, "Common Object Definition," in the section "Common Object Data Structure," page 315, were saved to a file using a *dump* as illustrated here

```
fwrite( gxObj, sizeof( GXObj), 1, fp );
```

the data would depend on the byte ordering and word groupings of the machine architecture that performed the `fwrite`.

The data could only be successfully read on a machine of the same architecture.

The binary file formatting method is (generally) more efficient both in the size of the files it generates and in the speed at which the files are parsed.

Files saved using ASCII file formatting (as introduced in the next section) are not as efficient but are easily maintained.

ASCII File Formatting

ASCII file formats, as implied by the title, are based on ASCII and use only the printable characters from the ASCII table.

Contrasted to binary file formats for saving application data, ASCII files are inherently larger than necessary because the data is written using 8-bit encoding even when the data requirements are less.

Because the files are based on the ASCII table, text editors and word processors can read plain-text fields.

The following sections discuss two common methods of writing ASCII-based files: tagged and position-specific formatting.

Tagged File Formats

One method of providing extensibility and robustness to your application's data file is to employ a tagged file format.

Within a tagged file format, keywords are grouped closely to a value. Keyword value pairs can be one per line or separated using a token known not to exist in either any of the keyword specifications or possible values that will satisfy the keywords.

An example of a tagged file format is shown in Listing 18.2.

Listing 18.2 Sample Line-Separated Tagged File Format

```
FS=None
LS=Solid
LW=3
FG=Red
BG=None
```

Optionally, all tag value pairs could appear separated by a token on the same line, as shown in Listing 18.3.

Listing 18.3 Sample Token-Separated Tagged File Format

```
FS=None,LS=Solid,LW=3,FG=Red,BG=None
```

The flexibility afforded by tagged file formats is the capability to add and remove keys with minimal impact on the owning application.

In the absence of a keyword value pair, the application can assign a default value to the destination field.

Further, if an unrecognized keyword value pair is found in the file, it could simply be ignored, enabling new data elements to be added to the file with no impact on older versions of the application loading the new file format.

The tagged ASCII file format is clearly the slowest to parse and load because of the level of processing that must occur for the keyword value pairs found in the data file.

Not as robust as the tagged ASCII file but offering some level of convenience are the position-specific ASCII file formats introduced in the next section.

Position-Specific File Formats

Position-specific ASCII file formats are the easiest to create programmatically because the developer can create format strings that are used for both the write and read functions of the application.

To implement a position-specific file format, the data elements targeted for the application's data file are formatted using the syntax of the format string used by the C language `fprintf` function.

Listing 18.4 demonstrates this using the same data fields from previous listings.

Listing 18.4 Writing a Position-Specific ASCII Data File Entry

```
fprintf( fp, "%s %s %s %s\n", fs, ls, lw, fg, bg );
```

This same format string is subsequently used to read the data with the C `fscanf` function, as shown in Listing 18.5.

Listing 18.5 Reading an Entry From a Position-Specific ASCII Data File

```
fscanf( fp, "%s %s %s %s\n", &fs, &ls, &lw, &fg, &bg );
```

The position-specific mechanism, though simple to implement, does not offer the resiliency or robustness of the tagged file method. A change to the file format introduces a complication that is not trivial to overcome. However, by placing a control tag or header element known as a *magic number* at the start of the file, the application will know the expected data and the file characteristics.

Magic Numbers

The general method of determining the data elements written to an application's data file is through use of a *magic number*, which serves as a control or version number for the application's restore function.

The best illustration of a magic number is its use in the graphic interchange file (GIF) format. This graphic image file uses two slightly different formats.

A string found at the very beginning of the file distinguishes the file format as either GIF87a or GIF89a.

Based on the value of the magic number, an application's load function can account for version differences corresponding to the format of the file being loaded.

The magic number, as illustrated with the strings GIF87a and GIF89, need not be an actual number. Literally, the term refers to a position-specific entry in the data file that can subsequently be used by the program to predict the elements contained in the file. As described with the Tagged Line format that uses a tag line to determine the data fields to follow, a magic number applies to the expectations of the contents of the entire file.

Next Steps

With a clearer understanding of the decisions and options you face when determining the best way to format a data file to meet the needs of the save-and-restore feature of an application, you are ready to face the challenge posed in the next chapter.

Chapter 19, "Save and Restore," satisfies the requirement of adding a save-and-restore capability to the Graphics Editor..

In this chapter

- *File Format Strategy*
- *Save and Restore Program Hooks*
- *Common-Object Save and Restore*
- *Object-Specific Save and Restore*
- *Next Steps*

Chapter 19

Save and Restore

The ability for users to continue their work in subsequent sessions of the Graphics Editor marks it as a mature and professional-level application.

Borrowing from the discussion of Binary, ASCII, and ASCII-tagged file formats of Chapter 18, "File Formats," the following sections satisfy the requirement of saving and restoring the graphics objects created by the Graphics Editor.

This chapter introduces a file format strategy and shows how the common and specific data elements of the editor objects will easily comply.

File Format Strategy

Familiar with the concepts of formatting application data files from Chapter 18, it is important that we identify a strategy for the Graphics Editor's generation of files that provides a blend of simplicity, resiliency, and robustness.

As was made clear from the editor object definitions introduced in Chapter 15, "Common Object Definition," not all the data elements that compose an editor object are homogenous. In other words, beyond the common elements controlling the attributes of the objects, each object type has unique data fields.

The `Arc` object retains the contents of the `XArc` structure to define data specific to the object:

```
typdef struct XArc {
    int x, y;
    Deminsion width, height;
    int angle1, angle2;
};
```

The Text object employs the GXText structure

```
typedef struct _gxtext {
    int x, y;     /* top-left */

    int dx, dy;
    char *text;
    int  len;

    GXFont  vpts;
    GXFont  font;   /* segment definitions */
    GXFontP fontp;  /* num segs per char   */

} GXText, *GXTextPtr;
```

and the standard point-array–based objects LatexLine, PolyLine, Box, and Arrow use the GXLine data structure:

```
typedef struct _gxline {

    XPoint *pts;
    int    num_pts;

} GXLine, *GXLinePtr;
```

Because the data requirements for the objects of the Graphics Editor are not identical, the file format strategy must account for the nesting of data for objects of different types.

To accomplish this, a combination of the Position-Specific and Tagged-File Formatting methods is used. The merging of the two techniques enables the restore method of the editor to distinguish between object data for the various object types as well as enabling the editor objects to expand the fields they save in future versions without affecting existing data files.

Listing 19.1 shows the data file generated by procedures introduced later in this chapter when the objects shown in Figure 19.1 are saved. The data file includes an example of the point-array–based Line object, an Arc object, and the Text object.

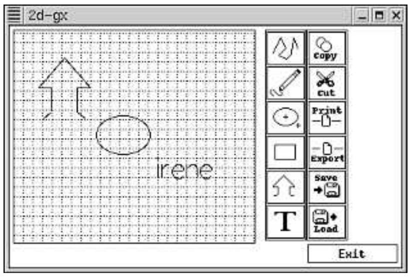


Figure 19.1

A sample of the objects saved by the editor.

Listing 19.1 Graphics Editor Data File Format

```
1:  OBJ - fg bg ls lw
2:  0 65535 0 1
3:  LINE [numpts x y x y ...]
4:  9
5:  30 91
6:  38 84
7:  38 59
8:  25 59
9:  52 28
10: 79 59
11: 65 59
12: 65 84
13: 74 91
14: OBJ - fg bg ls lw
15:  0 65535 0 1
16: ARC [x y width height angle1 angle2]
17: 85 88 56 40 0 23040
18: OBJ - fg bg ls lw
20:  0 65535 0 1
21: TEXT [ str x y ]
22: irene
23:  148 131
```

The pattern of the file is to first save a tagged line indicating the format of the data that follows on the next line. For instance,

```
1:  OBJ - fg bg ls lw
```

shows that the data to follow will be specific to the common object portion of an object and include the field's foreground, background, line style, and line width.

```
2:  0 65535 0 1
```

The second line is a position-specific formatted line containing the data fields promised by the previous tagged line.

Consider the tagged line produced by the `Line` object:

```
3: LINE [numpts x y x y ...]
```

This line indicates the form of the Line data that follows the tagged line. Specifically, saving first the number of points contained in the object

```
4: 9
```

followed by the appropriate pairs of points:

```
5:  30 91
6:  38 84
7:  38 59
8:  25 59
```

```
9:  52 28
10: 79 59
11: 65 59
12: 65 84
13: 74 91
```

The same format is applied to the `Arc` object

```
14:  OBJ - fg bg ls lw
15:   0 65535 0 1
16:  ARC [x y width height angle1 angle2]
17:  85 88 56 40 0 23040
```

and the `Text` object:

```
21:  TEXT [ str x y ]
22:  irene
23:   148 131
```

Notice that the fields stored for the `Text` object are considerably less than those contained in the `GXText` structure. Only the fields required to reproduce the object when loaded into the editor are placed in the data file. For instance, because the editor supports only one vector text font, the font data need not be placed in the data file. If, however, in the future you expand the editor to enable the user to select from many vector fonts, the font assigned to the saved object must be included in the data file. Fortunately, the data file's format strategy accounts for content expansion.

The strength of combining position-specific and tagged-file format strategies is that the application can parse the tagged line to know how to interpret the position-specific line.

In the future, when the application is advanced and functionality is added, the tagged line is updated to reflect the additional data field(s) saved to the file. The newer versions of the application will include the evaluation of the tagged line to determine whether an older file that doesn't contain the additional data or a newer file that does is being loaded.

The following sections show the necessary modifications to the Graphics Editor project to support the save and restore functionality using a combined position-specific and tagged-file format.

Save and Restore Program Hooks

The definition of the control menu for the Graphics Editor provides a point of entry for the save and restore functions:

```
static GxIconData gxCntrlIcons[] = {
    .
    .
    .
{ &save_icon,   gx_save,   "Save current drawing..." },
{ &load_icon,   gx_load,   "Load saved drawing..."   },
    .
    .
    .
{ NULL },
};
```

The functions `gx_save` and `gx_load`, however, have heretofore been empty (stub) functions. The next section introduces their contents, but first, we must modify the Common Object Structure to include an object method for saving the object-specific data contained in each graphics object.

Add the following line to the `GXObj` data structure located in `gxGraphics.h` header file:

```
void (*move)  ( struct _gx_obj *, XEvent * );
void (*scale) ( struct _gx_obj *, XEvent * );

void (*action)  ( struct _gx_obj *, XEvent * );

void (*save) ( FILE *, struct _gx_obj * );
```

Further, this object method should be initialized in the function `gx_create_obj` located in the `gxGx.c` source file:

```
gx_obj->scale    = (void    (*)())null_func;
gx_obj->copy     = (void    (*)())null_func;

gx_obj->save     = (void    (*)())null_func;
```

With the proper object method defined and initialized, we can now look at the contents of the `gx_save` and `gx_load` functions.

Common-Object Save and Restore

When the user clicks the Save icon located in the Control button panel of the Graphics Editor application, the `gx_save` function is invoked.

This function is responsible for prompting the user for a filename, opening the file, and traversing all objects contained in the application, saving first their common data elements and then invoking the objects' `save` method to write the object-specific data to the file.

The contents of the `gx_save` function accomplishing the steps described are shown in Listing 19.2.

Listing 19.2 The gx_save Function

```
1:  void gx_save( void )
2:  {
3:      FILE *fp;
4:
5:      char *filename = gxGetFileName();
6:
7:      setStatus( "Saving objects..." );
8:      if( filename ) {
9:          fp = fopen( filename, "w+" );
10:
11:         if( fp == NULL ) {
12:           perror( "Failed to open file: " );
13:         } else {
14:             gxSaveObjs( fp, gxObjHeader );
15:             fclose( fp );
16:
17:        }
18:     }
19:     setStatus( "Saving objects... done!" );
20: }
```

The gx_save function begins by invoking a function to prompt the user for a filename:

```
5:     char *filename = gxGetFileName();
```

When the filename is returned, a file of the specified name is opened for writing:

```
9:         fp = fopen( filename, "w+" );
```

After ensuring that the file opened successfully in lines 11–13, the function gxSaveObjs is called

```
14:                gxSaveObjs( fp, gxObjHeader );
```

specifying the beginning of the list of objects and a pointer for the opened data file.

Listing 19.3 shows the gxGetFileName function and Listing 19.4 shows the gxSaveObjs function.

EXCURSION

Error Reporting in the C Programming Language

Notice the perror command, used for the first time in this text in Listing 19.2.

```
12:            perror( "Failed to open file: " );
```

The C programming language provides the perror function to print to stderr the programmer's specified message ("Failed to open file: ") followed by the error reported by the system when the offending call failed.

System errors such as failing to open a file are made available to routines in C by merit of a global variable named `errno`.

The value of `errno` is set when an error occurs, and its value corresponds to an array of error strings maintained by C. For instance, if you wanted the Graphics Editor to report the error to the status bar, replace the `perror` call with the following:

```
{
    char msg[128];
    sprintf( msg, "Failed to open file %s : %s\n",
             filename, strerror( errno ) );

    setStatus( msg );
}
```

The `strerror` function, also provided by C, will find the error string corresponding to the value of `errno`. In order for these functions to be prototyped for use and `errno` to be visible in the source file, you must include the `error.h` header file and `extern` the `errno` variable. This is accomplished with the following lines:

```
#include <error.h>
extern in errno;
```

Listing 19.3 Retrieving a Filename from User Input

```
static char *gxGetFileName( void )
{
  XtAppContext app;
  XEvent       event;

  Widget dialog;
  char *str = NULL;

  dialog = XtVaCreateManagedWidget( "Filename...",
                                    dialogWidgetClass,
                                    GxDrawArea,
                                    XtNwidth,  115,
                                    XtNheight, 70,
                                    XtNlabel,  "Enter File:",
                                    XtNvalue,  "",
                                    NULL );

  XawDialogAddButton( dialog, " Ok ", close_dialog, dialog );

  app = XtWidgetToApplicationContext( GxDrawArea );

  while( XtIsManaged(dialog)) {
    XtAppNextEvent( app, &event );
    XtDispatchEvent( &event );
```

continues

Listing 19.3 Continued

```
25:   }
26:
27:   str = XawDialogGetValueString( dialog );
28:   XtDestroyWidget( dialog );
29:
30:   /*
31:    * look for 'illegal' characters
32:    */
33:   {
34:     int c, indx = 0;
35:     char illegal_chars[] = { '\n', 'z' };
36:
37:     while( (c = (int)str[indx]) != '\0' ) {
38:       if( strchr( illegal_chars, c ) != NULL ) {
39:         str[indx] = '\0';
40:         break;
41:       }
42:       indx++;
43:     }
44:
45:     /*
46:      * remove leading zeros
47:      */
48:     while( *str && *str == ' ' ) str++;
49:   }
50:
51:   if( str && *str )
52:     return XtNewString(str);
53:   else
54:     return NULL;
55: }
56: static void close_dialog( Widget w, XtPointer cdata,
57:                           XtPointer cbs )
58: {
59:   Widget dialog = (Widget)cdata;
60:
61:   if( dialog ) XtUnmanageChild( dialog );
62: }
```

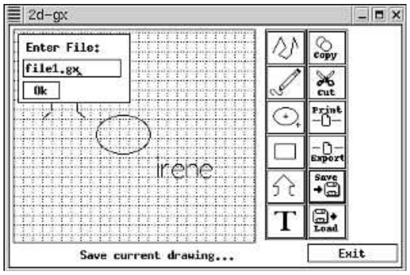
The `gxGetFileName` function that is found in Listing 19.3 creates a dialog widget with the single OK button (lines 9–18). An illustration of `dialogWidget` created by this function is found in Figure 19.2.

Note

Inclusion of the header files `X11/Xaw/Dialog.h` and `X11/Xaw/StringDefs.h` as well as a prototype for `close_dialog` must appear at the beginning of the file where you place the functions introduced in Listing 19.2 in order for the sample to compile without error.

Figure 19.2

Prompting the user for a filename.



The function then extracts the `XtApplicationContext` and uses it to intercept events from the `XtAppMainLoop` by extracting and dispatching events until the user closes the dialog prompting for input:

```
app = XtWidgetToApplicationContext( GxDrawArea );

while( XtIsManaged(dialog)) {
  XtAppNextEvent( app, &event );
  XtDispatchEvent( &event );
}
```

A widget is considered managed as long as it is visible on the screen. When the user presses the OK button, the dialog is removed from the screen by the `close_dialog` function registered as the OK button's callback action

```
XawDialogAddButton( dialog, " Ok ", close_dialog, dialog );
```

and the test `XtIsManaged` returns `False`, ending the `while` loop.

The `close_dialog` function simply "unmanages" the `dialogWidget` when the button is pressed:

```
if( dialog ) XtUnmanageChild( dialog );
```

The value entered into the text field inherent to the `dialogWidget` is consulted for the user's input

```
str = XawDialogGetValueString( dialog );
```

and the widget is destroyed:

```
XtDestroyWidget( dialog );
```

The remainder of the `gxGetFileName` function ensures the integrity of the string for use as a filename. Specifically, a filename cannot contain newlines, spaces, or, solely for the purpose of illustration, lowercase *z* characters.

When the function is sure of the contents of the string, a copy is returned to the user:

```
if( str && *str )
  return XtNewString(str);
```

Otherwise, if the removal of illegal characters has consumed the string, `NULL` is returned:

```
53:   else
54:     return NULL;
```

The `gxSaveObjs` routine introduced in Listing 19.4 satisfies the final function introduced in Listing 19.2

```
14:             gxSaveObjs( fp, gxObjHeader );
```

for accomplishing the saving of objects created by the editor application.

Listing 19.4 Traversing the Editor Objects for Saving

```
1:  static void gxSaveCommon( FILE *fp, GXObjPtr obj )
2:  {
3:     fprintf( fp, "OBJ - fg bg ls lw\n" );
4:     fprintf( fp, " %ld %ld %d %d\n",
5:              obj->fg, obj->bg, obj->ls, obj->lw );
6:  }
7:
8:  static void gxSaveObjs( FILE *fp, GXObjPtr obj )
9:  {
10:     if( obj ) {
11:         gxSaveCommon( fp, obj );
12:         (*obj->save)( fp, obj );
13:
14:         gxSaveObjs( fp, obj->next );
15:     }
16: }
```

The `gxSaveObjs` function in Listing 19.4 is a recursive function called repetitively until the end of the list of objects is found. The two lines

```
10:     if( obj ) {
```

and

```
14:         gxSaveObjs( fp, obj->next );
```

are both important to the recursive nature of the function. Line 10 ensures that a valid object reference was provided as the end of the list of objects is marked with `NULL`, whereas line 14 passes the next possible object.

For each object contained in the list, two functions are invoked. The first is to save the common object data fields

```
11:         gxSaveCommon( fp, obj );
```

and the second is to save the object-specific data:

```
12:         (*obj->save)( fp, obj );
```

The `gxSaveCommon` found simply writes the tagged line and the position-specific data line to the file referenced by the file pointer:

```
    fprintf( fp, "OBJ - fg bg ls lw\n" );
    fprintf( fp, " %ld %ld %d %d\n",
             obj->fg, obj->bg, obj->ls, obj->lw );
```

The graphic objects' save methods are introduced in Chapters 20–24 with the definition of the object internals.

The following section demonstrates how the data file generated by the save action of the Graphics Editor is restored in the editor.

Object-Specific Save and Restore

When the user clicks the Load icon located in the control button panel of the Graphics Editor application, the `gx_load` function is invoked.

This function is responsible for prompting the user for a filename, opening the file, traversing the file contents, and creating an object specific to the tags located in the file.

Listing 19.5 shows the contents of the `gx_load` function.

Listing 19.5 The `gx_load` Function

```
void gx_load( void )
{
    FILE *fp;

    char *filename = gxGetFileName();

    setStatus( "Loading objects..." );
    if( filename ) {
      fp = fopen( filename, "r" );

      if( fp == NULL ) {
        perror( "Failed to open file: " );
      } else {
          gxLoadObjs( fp );
          fclose( fp );
      }
    }
    setStatus( "Loading object... done!" );
}
```

After prompting for and validating the filename, the file is opened read-only:

```
    if( filename ) {
      fp = fopen( filename, "r" );
```

If the file is opened successfully, the function `gxLoadObjs` is called with the file reference:

```
14:           gxLoadObjs( fp );
```

Listing 19.6 introduces the `gxLoadObjs` function, which traverses the file referenced by `fp` and invokes the necessary `create` functions to restore the objects to the editor.

Listing 19.6 Loading Objects into the Editor

```
1:  static void gxLoadCommon( FILE *fp, GXObjPtr obj )
2:  {
3:     fscanf( fp, " %ld %ld %d %d\n",
4:              &obj->fg, &obj->bg, &obj->ls, &obj->lw );
5:  }
6:
7:  static void gxLoadObjs( FILE *fp )
8:  {
9:      char objForm[128];
10:     GXObjPtr obj;
11:
12:     while( fgets( objForm, 128, fp ) != NULL ) {
13:         obj = gx_create_obj();
14:         gxLoadCommon( fp, obj );
15:
16:         fgets( objForm, 128, fp );
17:         switch( *objForm ) {
18:           case 'A':
19:             gxArcLoad( fp, obj );
20:             break;
21:
22:           case 'L':
23:             gxLineLoad( fp, obj );
24:             break;
25:
26:           case 'T':
27:             gxTextLoad( fp, obj );
28:             break;
29:         }
30:     }
31: }
```

The `gxLoadObjs` function retrieves a line from the file reference by the file pointer `fp`

```
12:     while( fgets( objForm, 128, fp ) != NULL ) {
```

using the C `fgets` function, which fills the buffer `objForm` up to the occurrence of a newline character in the file.

Note

Knowing the format of the file, you are aware that this line contains the tagged field line specifying the data elements provided on the following line.

Because this is version 1.0, there is no need to parse the tagged line to determine the fields specified. If, however, in future versions of the application, data fields are added to the save format, this is when the line would be parsed.

Following the extraction of the tagged line, the data for the object-specific values is read:

```
        gxLoadCommon( fp, obj );
```

Notice that in the `gxLoadCommon` function, the syntax for reading the data written by `gxSaveCommon` is very similar. Specifically, the `fwritef` has been renamed `fscanf` and addresses of the structure elements are provided so that the values read from the file can be assigned to them:

```
    fscanf( fp, " %ld %ld %d %d\n",
              &obj->fg, &obj->bg, &obj->ls, &obj->lw );
```

With the common object elements restored, the next line of the file is the tagged line of the object-specific data. By reading this line into the `objForm` array

```
        fgets( objForm, 128, fp );
```

you can determine which load routine to call because the first character is sufficient for determining the object type:

```
        switch( *objForm ) {
```

Note

The syntax `*objForm` is identical to specifying `objForm[0]`. Both references equate to the first character of the array.

Based on the value of the first character, one of the methods `gxArcLoad`, `gxLineLoad`, or `gxTextLoad` is invoked to read the data specific to an object of that type.

These functions are introduced in the following chapters when (*finally* you must be saying) the internals of the various editor objects are introduced.

Next Steps

The addition of functionality to support the save and restore capability of the editor is necessary to produce a mature product. Relative to the flow of this text, this chapter completes the evolution of the graphics editor objects and enables us to address the important task of defining each object in full.

The following five chapters in Part V, “Adding Objects to the Editor,” provide the definition of internal object methods and functions required to create a functional graphics editor.

Part V

Adding Objects to the Editor

Chapter 20

In this chapter

Latex Line Object

By choosing the icon shown in Figure 20.1, a user begins the `Latex Line` object creation process.

Figure 20.1

Choose the Latex Line creation icon to begin the `Latex Line` *object creation process.*

During the creation of a `Latex Line` object, points are selected from the canvas to define the vertices of the object. There is no limit to the number of points or the direction of the segments created during this process.

When all the points are defined for the new `Latex Line` object, the user double-clicks the mouse cursor to indicate the end of the creation. Upon completing the interactive mode of the `Latex Line` object creation, a plethora of events occur.

This chapter presents the internal methods and supporting functions defined for the `Latex Line` object. These routines control all aspects of managing the different features of the object.

Note

The code listings introduced in this chapter are targeted for placement in the `gxLine.c` source file. Additionally, functions presented in the listings that are not defined `static` should have a corresponding prototype placed in the `gxProtos.h` header file.

Creating a `Latex Line` Object

The definition of the `gxDrawIcons` array introduced in Chapter 13, “Application Structure”

```
{ &line_icon, (void (*)(void))gx_line,
  "Draw an elastic line..."       },
```

assigned the Latex Line icon the `gx_line` function for invocation when selected by the user. This function is defined in Listing 20.1, and it serves as the entry point to the creation process of the `Latex Line` object.

Listing 20.1 The `gx_line` Function

```
void gx_line( XEvent *event )
{
  static GXLine *rubber_line = NULL;

  if( event == NULL ) {
    rubber_line = NULL;
  } else {
    /* remove the current rubber line (if there is one) */
    GXRubberLine( rubber_line );

    switch( event->type ) {
      case ButtonPress:
        if( rubber_line == NULL ) set_cursor( LINE_MODE );

        /*
         * See if we 'double clicked' & are done selecting points
         */
        if( point_equal_event( rubber_line, event ) == True ) {

          /* erase our temp line */
          XDrawLines( XtDisplay(GxDrawArea), XtWindow(GxDrawArea),
                      rubberGC, rubber_line->pts,
                      rubber_line->num_pts, CoordModeOrigin );

          create_line( NULL, rubber_line );
          gx_refresh();

          set_cursor( NORMAL_MODE );
          rubber_line = NULL;
        } else {
          /*
           * Initialize a GXLine struture to manage our creation
           */
          GXRubberLine( rubber_line );
          rubber_line = update_line( event, rubber_line );
        }
      break;

      case ButtonRelease:
      case MotionNotify:
        /*
         * update the GXLine structure based on the
         * new point and current location of the mouse
         */
        if( rubber_line ) {
          /*
           * replace the last point with the current event location
           * IF we have more than one point
           */
          if( rubber_line->num_pts > 1 ) {

            rubber_line->pts[rubber_line->num_pts-1].x =
                                        event->xbutton.x;
            rubber_line->pts[rubber_line->num_pts-1].y =
                                        event->xbutton.y;

          } else {
            (void)update_line( event, rubber_line);
          }
          /*
           * redraw the rubberbanding line
           */
          GXRubberLine( rubber_line );
        }
      break;
    }
  }
}
```

In reviewing the definition of the `gx_line` function, you instantly see that the function is entirely event driven—meaning, based on the event specified, the `gx_line` function takes an appropriate action.

The actions for each of the events anticipated can be generalized as follows:

- Upon receiving a `ButtonPress` event, begin the creation process.
- Subsequent `ButtonPress` events cause the addition of the event point to an array of points contained in the `rubber_line` structure.
- A `ButtonPress` event point that equals the previous point inserted into the `rubber_line` structure indicates a double-click and ends the creation.
- At the occurrence of either the `ButtonRelease` or `MotionNotify` event, the event point is connected to the most recent point added to the `point` array and drawn using a rubber-banding line to provide the interactive creation of the Latex Line.

The `gx_line` function begins by testing the validity of the event reference specified as the sole parameter to the function:

```
5:    if( event == NULL ) {
6:      rubber_line = NULL;
7:    } else {
```

Passing a NULL event pointer to the gx_line function enables the local variable rubber_line to be reset, thereby aborting the current creation mode. This is necessary to eliminate the possibility of the user failing to end one creation process before starting another. When a new creation process is started for any type of object, a NULL event is passed first to ensure the initial state.

When, however, a valid event pointer is specified, the rubber_line variable is passed to the GXRubberLine function, which is responsible for updating the line using the interactive rubber-banding GC as follows:

```
9:     GXRubberLine( rubber_line );
```

Then the event type is checked to determine which action body the function should execute:

```
11:     switch( event->type ) {
```

If the current event is a ButtonPress event, many things can happen because the action body associated with this event is the busiest. If the rubber_line variable is still NULL, the action body knows that this is the first iteration and sets the application cursor to reflect the new mode:

```
13:         if( rubber_line == NULL ) set_cursor( LINE_MODE );
```

Note

As you might recall, the cursor control functions were introduced in Chapter 17, "Utilities and Tools," in "Using the Cursor as a State Indicator," page 349.

When a ButtonPress event occurs, the gx_line action for it must determine whether this event's location coincides with the pervious ButtonPress event point passed to the function:

```
18:         if( point_equal_event( rubber_line, event ) == True ) {
```

If the point_equal_event function indicates that the points are equal, the creation process ends by erasing the interactive rubber-banding line and calling the create_line function (lines 21–25).

The ButtonPress case then returns the cursor to normal and clears the rubber_line variable:

```
28:           set_cursor( NORMAL_MODE );
29:           rubber_line = NULL;
```

However, if the point_equal_ function returned False, indicating that the current event point does not equal the previously added point (no double-click), the else body is entered, the interactive line is updated

```
            GXRubberLine( rubber_line );
```

and the current event point is added to the rubber_line->pts array:

```
            rubber_line = update_line( event, rubber_line );
```

The processing of the ButtonRelease or MotionNotify events is a little simpler:

```
        case ButtonRelease:
        case MotionNotify:
```

If a valid rubber_line structure reference exists, replace the last point in the array with the current event point:

```
              rubber_line->pts[rubber_line->num_pts-1].x =
                                            event->xbutton.x;
              rubber_line->pts[rubber_line->num_pts-1].y =
                                            event->xbutton.y;
```

If the rubber_line->pts array contains only one point, the array must be expanded to accommodate the point addition:

```
              (void)update_line( event, rubber_line);
```

Note that the last point added to the pts array by the ButtonRelease or MotionNotify actions is not committed to the list of vertices that are specified by the user. Instead, this last point simply makes it easier for the GXRubberLine to do its job because it only must update the last segment defined by the array. This segment consists of one point selected by the user (second to the last point of the array) and one point stored temporarily by ButtonRelease or MotionNotify (last point). In other words, one more point is maintained in the rubber_line->pts array than requested by the user. This last point stored by the ButtonRelease and MotionNotify is replaced with the next ButtonPress event.

The last thing the gx_line function is responsible for is ensuring that the interactive rubber_line is again updated to the screen:

```
            GXRubberLine( rubber_line );
```

The GXRubberLine function defined in Listing 20.2 updates segments of the line being created, providing a visual interactive creation for the user by enabling him to see where the next segment endpoint will be placed when the mouse button is pressed.

Listing 20.2 The `GXRubberLine` Function

```
1:  static void GXRubberLine( GXLine *line )
2:  {
3:    int indx;
4:    if( line && line->pts && (line->num_pts > 1)) {
5:      indx = line->num_pts - 1;
6:      XDrawLine( XtDisplay(GxDrawArea),
7:                 XtWindow(GxDrawArea), rubberGC,
8:                 line->pts[indx  ].x,
9:                 line->pts[indx  ].y,
10:                line->pts[indx-1].x,
11:                line->pts[indx-1].y );
12:   }
13: }
```

The `GXRubberLine` function begins by ensuring enough points are defined to form a line segment

```
4:   if( line && line->pts && (line->num_pts > 1)) {
```

and then it assigns the variable `indx` the value of one less than the number of points, which is the last point of the array:

```
5:     indx = line->num_pts - 1;
```

Note

Remember from our discussions of arrays in Chapter 2, "Programming Constructs," that an array with two elements employs the indices 0 and 1 to access them.

Then the last segment of the line defined by the `pts` array is drawn using `indx` (the last point) and `indx - 1` (the second to the last point):

```
6:      XDrawLine( XtDisplay(GxDrawArea),
7:                 XtWindow(GxDrawArea), rubberGC,
8:                 line->pts[indx  ].x,
9:                 line->pts[indx  ].y,
10:                line->pts[indx-1].x,
11:                line->pts[indx-1].y );
```

Look now at the definition of the `point_equal_point` function defined in Listing 20.3, which is used by the `gx_line` function to determine whether the user double-clicked the mouse cursor.

Listing 20.3 The `point_equal_point` Function

```
1:  static Boolean point_equal_event( GXLine *line, XEvent *event )
2:  {
3:    Boolean pts_equal = False;
4:    int xe_x, xe_y;
5:
6:    xe_x = event->xbutton.x;
7:    xe_y = event->xbutton.y;
8:
9:    /*
10:    * the last point will always be the current motion event
11:    * so check the one before for redundancy (equates to a
12:    * double click to end the action )
13:    */
14:   if( line && (line->num_pts > 2) ) {
15:     int num = line->num_pts - 2;
16:
17:     if( (abs(line->pts[num].x - xe_x) <= TOLERANCE ) &&
18:         (abs(line->pts[num].y - xe_y) <= TOLERANCE )) {
19:
20:      pts_equal = True;
21:    }
22:   }
23:   return pts_equal;
24: }
```

The `point_equal_point` function must ensure that there are at least three points in the `pts` array before determining whether a double-click occurred:

```
14:   if( line && (line->num_pts > 2) ) {
```

This is necessary because the `ButtonRelease` or `MotionNotify` and not the user placed the last point of the array. In order for the line creation to be valid, at least two user-specified points must be in the array.

When the correct number of points exists in the array, the last user-defined point is compared to the event point. If the absolute value of their differences is less than the `TOLERANCE` (3) defined in the `gxGraphics.h` header file, they are considered equal.

Note

The margin of error allowed by `TOLERANCE` accounts for the user having Monday morning shakes.

Furthermore, even on a good day, it is difficult to click the exact same point on the screen consecutive times. It is expected that the mouse can move ever so slightly from the subtle movement of hand muscles required to double-click the mouse.

If the points are deemed equal (or close enough), the interactive portion of the creation process ends and an actual line object is created with a call to the `create_line` function found in Listing 20.4.

Listing 20.4 The `create_line` Function

```
1:  static void create_line( GXObjPtr _obj, GXLine *line )
2:  {
3:    GXLinePtr line_data;
4:    GXObjPtr  obj = _obj;
5:
6:    if( obj == NULL ) {
7:        obj = gx_create_obj();
8:    }
9:
10:   line_data = (GXLinePtr)XtNew(GXLine);
11:   memcpy( (char *)line_data, (char *)line, sizeof(GXLine));
12:
13:   obj->data = line_data;
14:
15:   obj->draw     = line_draw;
16:   obj->erase    = line_erase;
17:   obj->find     = line_find;
18:   obj->move     = line_move;
19:   obj->scale    = line_scale;
20:   obj->copy     = line_copy;
21:   obj->select   = line_select;
22:   obj->deselect = line_deselect;
23:
24:   obj->save = line_save;
25:
26:   gx_add_obj( obj );
27: }
```

The parameter list of the `create_line` function

```
1: static void create_line( GXObjPtr _obj, GXLine *line )
```

enables an invocation of the procedure with the caller having already created the common object to contain the line data. This supports the restoring of the object from a saved file as is seen later in the chapter.

During interactive creation, however, the function `create_line` is called with a `NULL` value as the first parameter forcing the function to create the common object portion through a call to `gx_create_obj` introduced in Chapter 17, “Utilities and Tools,” section “Common Object Creation,” page 343.

```
7:        obj = gx_create_obj();
```

A `GXLine` data structure is then created

```
10:   line_data = (GXLinePtr)XtNew(GXLine);
```

and the contents of the second argument to the function is copied into the new structure:

```
11:   memcpy( (char *)line_data, (char *)line, sizeof(GXLine));
```

Finally, the new data structure is assigned as the unique data value for the new object:

```
13:   obj->data = line_data;
```

Consistent with objects of type Line, the methods for the new object are assigned the line manipulation functions for controlling the object-specific data structure in lines 15–24.

Finally, the `create_line` function must ensure the newly created object is retained by the application by adding the object to the linked list used to manage editor objects:

```
26:   gx_add_obj( obj );
```

Note

To review the `gx_add_obj` function, refer to Chapter 17, section “Linked List Management,” page 346.

The final function for review is the `update_line` function found in Listing 20.5, which is employed by `gx_line` to commit user selected points to the `rubber_line->pts` array.

Listing 20.5 The `update_line` Function

```
1:  static GXLinePtr update_line( XEvent *event, GXLine *line )
2:  {
3:    static GXLine xline;
4:    GXLinePtr     xlinePtr = &xline;
5:
6:    if( line == NULL ) {
7:      xline.pts = (XPoint *)XtMalloc( sizeof(XPoint) );
8:      xline.num_pts = 0;
9:    } else {
10:     xline.pts = (XPoint *)XtRealloc( (char *)xline.pts,
11:                          sizeof(XPoint) * (xline.num_pts + 1));
12:   }
13:
14:   xline.pts[xline.num_pts].x = event->xbutton.x;
15:   xline.pts[xline.num_pts].y = event->xbutton.y;
16:
17:   xline.num_pts++;
18:
19:   return xlinePtr;
20: }
```

The `update_line` function must manage the memory associated with the `points` array, holding the vertices added to the `Line` object by the user. If the `line` pointer is `NULL`, an `XtMalloc` is called to assign the initial storage space for the `pts` array

```
7:     xline.pts = (XPoint *)XtMalloc( sizeof(XPoint) );
```

and the number of points tracked by `num_pts` is initialized to 0

```
8:     xline.num_pts = 0;
```

otherwise, memory previously assigned to the `xline` `GXLine` structure referenced is re-allocated to include room for the point being added:

```
10:     xline.pts = (XPoint *)XtRealloc( (char *)xline.pts,
11:                          sizeof(XPoint) * (xline.num_pts + 1));
```

After sufficient room is obtained to contain the new point, it is added to the array

```
14:   xline.pts[xline.num_pts].x = event->xbutton.x;
15:   xline.pts[xline.num_pts].y = event->xbutton.y;
```

and the total number of points is incremented to reflected the entry:

```
17:   xline.num_pts++;
```

As the user interactively creates the Latex Line, points are added to the `rubber_line->pts` array and the number of points tracked in the `num_pts` field. At some point the user will double-click the mouse cursor, which is caught by the `point_equal_point` function, and the `Line` object will be created by the call to `create_line`. A created object is no longer drawn using the interactive rubber-banding `GC`, but by invoking the `draw` method of the object.

The following section introduces the methods that draw and erase the `Line` object from the drawing area canvas.

Drawing and Erasing a `Line` Object

The act of drawing or erasing an object in the Graphics Editor differs only in the treatment of the `tile` field of the `GC` created for the request.

Notice in Listing 20.6, lines 18 and 23, that the `draw` and `erase` methods differ only in the value passed for the `tile` flag required as the second parameter to the `draw_erase` function. (Note as well that the `erase` method is responsible for removing the object's handles if present on the screen.)

Listing 20.6 Drawing and Erasing a `Line` Object

```
1:  static void draw_erase( GXObjPtr line, Boolean tile )
2:  {
3:    GC gc;
4:    GXLinePtr line_data = (GXLinePtr)line->data;
5:
6:    gc = gx_allocate_gc( line, tile );
7:
```

```
8:    XDrawLines(XtDisplay(GxDrawArea),
9:              XtWindow(GxDrawArea), gc,
10:             line_data->pts, line_data->num_pts,
11:             CoordModeOrigin );
12:
13:   XtReleaseGC( GxDrawArea, gc );
14: }
15:
16: static void line_draw( GXObjPtr line )
17: {
18:   draw_erase( line, False );
19: }
20:
21: static void line_erase( GXObjPtr line )
22: {
23:   draw_erase( line, True );
24: }
```

The `draw_erase` function begins by extracting the `GXLine` data structure from the common object's data field

```
4:   GXLinePtr line_data = (GXLinePtr)line->data;
```

and creating a `GC` from the attribute setting of the object:

```
6:   gc = gx_allocate_gc( line, tile );
```

Important to the creation of the Graphic Context is the specification of the tile flag to the `gx_allocate_gc` function introduced in Chapter 17, section "Creating a Graphics Context," page 347.

Note

If you recall, this flag indicated whether the background `Pixmap` of the `GxDrawArea` was assigned as the value to the `tile` field of the `GC` created.

A `tile` value assigned to a Graphic Context causes an effective erase action to occur because the pixels from the background `Pixmap` are placed where otherwise the foreground value of the `GC` would be placed, making the underlying background the result of the `XDrawLines` (or any X Graphic Primitive) request.

An appropriately created `GC` for the current draw or erase action, the `draw_erase` function can request the object be updated in the canvas window:

```
8:    XDrawLines(XtDisplay(GxDrawArea),
9:              XtWindow(GxDrawArea), gc,
10:             line_data->pts, line_data->num_pts,
11:             CoordModeOrigin );
```

Because the X Server will attempt to cache the `GC` for future requests, it is important to specify the complete use of it with a call to `XtReleaseGC`:

```
13:   XtReleaseGC( GxDrawArea, gc );
```

With the object visible on the screen, it is now eligible for manipulation by the user. However, before it can be moved, scaled, or deleted, it must be selected by the user.

The next section introduces the `Line` object's `find` method used to determine whether an event has successfully located the object on the drawing area.

Finding a `Line` Object

The `Line` object's `find` method, shown in Listing 20.7, considers the object selected if the event point contained in the `XEvent` structure referenced in the function's second parameter coincides with one of the object's vertices or intersects one of the object's line segments.

Listing 20.7 The `line_find` Function

```
1:  static Boolean line_find(GXObjPtr line, XEvent *event)
2:  {
3:    Boolean found = False;
4:    XPoint p;
5:
6:    p.x = event->xbutton.x;
7:    p.y = event->xbutton.y;
8:
9:
10:    found = point_selected( (GXLinePtr)line->data, &p );
11:    if( found == False )
12:      found = segment_selected((GXLinePtr)line->data,&p);
13:
14:    return found;
15:  }
```

The `line_find` function extracts the event point from the XEvent structure

```
6:    p.x = event->xbutton.x;
7:    p.y = event->xbutton.y;
```

and tests first the vertices of the object

```
10:   found = point_selected( (GXLinePtr)line->data, &p );
```

and then the intersection of the point with the segments forming the object:

```
12:     found = segment_selected((GXLinePtr)line->data,&p);
```

If a point is determined selected by the `point_selected` function, the work is done. Otherwise, the `segment_selected` is invoked and the resulting value found is returned to the caller.

Listing 20.8 introduces the `segment_selected` and `point_selected` functions in turn.

Listing 20.8 Determining Whether a Point or Segment Is Selected

```
static
Boolean segment_selected( GXLinePtr data, XPoint *pt )
{
  Boolean found = False;
  int i;

  for(i=0; i<data->num_pts-1 && found == False; i++) {
    found = near_segment(data->pts[i].x,data->pts[i].y,
                    data->pts[i+1].x, data->pts[i+1].y,
                    pt->x, pt->y );
  }
  return found;
}

static
Boolean point_selected( GXLinePtr line, XPoint *pt )
{
  int i, x, y, found = False;

  for( i = 0; (i < line->num_pts) && !found; i++ ) {

    x = line->pts[i].x;
    y = line->pts[i].y;

    if( abs(pt->x - x) <= TOLERANCE  &&
        abs(pt->y - y) <= TOLERANCE ) {

      found = True;
    }
  }
  return found;
}
```

The `segment_selected` function, found at the beginning of Listing 20.8, breaks the `Line` object into point pairs, defining each of the line segments that comprise the object. Each of these segments is passed to the `near_segment` function introduced in Chapter 10, "Trigonometric and Geometric Functions," in the section "Calculating Point and Line Intersection," page 211.

If for any segment the `near_segment` function returns `True`, the `for` loop ends and the value is returned to indicate that the object has been found.

The `point_selected` function, found in Listing 20.8, parses all the points defining the vertices (segment endpoints) of the object looking for a point within `TOLERANCE` of the event point.

If an appropriate point is found, `True` is returned to the `line_find` method, indicating that the current `Line` object has been successfully located by the event.

The `line_find` method serves two purposes in the management of the `Line` object. One purpose of the `line_find` method is to enable the user to select the object for manipulation. A second purpose determines whether the manipulation requested by the user is the `move` action.

The follow section introduces the step stemming from the `line_find` method resulting in the selection (or implicit deselection) of the `Line` object.

Selecting and Deselecting a `Line` Object

If the event passed to the `line_find` method introduced in the previous section results in the location of an object that is not currently selected, the object is made active by invoking its `select` method.

The `select` method for the `Line` object, as seen in Listing 20.9, ensures that handles are created and drawn for the object.

Listing 20.9 Selecting a `Line` Object

```
static void line_select( GXObjPtr line )
{
  line_bounding_handles( line );
  gx_draw_handles( line );
}
```

Note

The function `gx_draw_handles` was introduced in Chapter 16, "Object Manipulation," in the section "Managing Object Handles," page 327.

The process of creating the handles for the `Line` object begins with a call to `line_bounding_handles` found in Listing 20.10.

Listing 20.10 Creating Handles for the `Line` Object

```
static void line_bounding_handles( GXObjPtr gx_line )
{
  int  i, x1, y1, x2, y2, width, height;

  GXLine *line_data = gx_line->data;

  gx_line->handles =
    (XRectangle *)XtMalloc( sizeof(XRectangle) * 8 );
  gx_line->num_handles = 8;

   if( gx_line->handles == NULL ) {
     perror( "Alloc failed for line handles" );
     gx_line->num_handles = 0;
14:      return;
15:    }
16:
17:    for( i = 0; i < 8; i++ ) {
18:      gx_line->handles[i].width  = HNDL_SIZE;
19:      gx_line->handles[i].height = HNDL_SIZE;
20:    }
21:
22:    get_bounds( line_data->pts, line_data->num_pts,
23:                &x1, &y1, &x2, &y2 );
24:    width  = x2 - x1;
25:    height = y2 - y1;
26:
27:    gx_line->handles[0].x = x1 - HNDL_OFFSET - HNDL_SIZE;
28:    gx_line->handles[0].y = y1 - HNDL_OFFSET - HNDL_SIZE;
29:
30:    gx_line->handles[1].x = x1 + (width/2) - HNDL_OFFSET;
31:    gx_line->handles[1].y = y1 - HNDL_SIZE - HNDL_OFFSET;
32:
33:    gx_line->handles[2].x = x2 + HNDL_OFFSET;
34:    gx_line->handles[2].y = y1 - HNDL_SIZE - HNDL_OFFSET;
35:
36:    gx_line->handles[3].x = x2 + HNDL_OFFSET;
37:    gx_line->handles[3].y = y1+(height/2)-HNDL_OFFSET;
38:    gx_line->handles[4].x = x2 + HNDL_OFFSET;
39:    gx_line->handles[4].y = y2 + HNDL_OFFSET;
40:
41:    gx_line->handles[5].x = x1 + (width/2) - HNDL_OFFSET;
42:    gx_line->handles[5].y = y2 + HNDL_OFFSET;
43:
44:    gx_line->handles[6].x = x1 - HNDL_OFFSET - HNDL_SIZE;
45:    gx_line->handles[6].y = y2 + HNDL_OFFSET;
46:
47:    gx_line->handles[7].x = x1 - HNDL_OFFSET - HNDL_SIZE;
48:    gx_line->handles[7].y = y1+(height/2)-HNDL_OFFSET;
49:  }
```

The creation of the `Line` object handles in `line_bounding_handles` begins by ensuring that the correct number of elements are created in the `handles` array and the correct handle count is assigned the `num_handles` field of the common object structure:

```
7:    gx_line->handles =
8:      (XRectangle *)XtMalloc( sizeof(XRectangle) * 8 );
9:    gx_line->num_handles = 8;
```

After testing for a failure of the allocation routine, the widths of all of the handles for the `Line` object are assigned:

```
17:   for( i = 0; i < 8; i++ ) {
18:     gx_line->handles[i].width  = HNDL_SIZE;
19:     gx_line->handles[i].height = HNDL_SIZE;
20:   }
```

Then, in keeping with the name of the function, the bounds of the `Line` object are obtained in order to determine the placement of the handles:

```
22:   get_bounds( line_data->pts, line_data->num_pts,
23:               &x1, &y1, &x2, &y2 );
```

Note

The `line_bounding_handles` assigns handles at each corner and on every side of the object to mark the bounding box that is required to contain the object.

For a review of object handles see Chapter 16, section "Managing Object Handles," page 327.

Note

The `get_bounds` function is borrowed from the `Text` object and is introduced with the `Text` object internal methods and functions in Chapter 24, "Vector Text Object."

The `get_bounds` function finds the minimum and maximum points of the segment endpoints contained in the `pts` array.

Using these extents, the `line_bounding_handles` determines the width and the height of the object:

```
24:   width  = x2 - x1;
25:   height = y2 - y1;
```

Last comes the tedious task of placing each of the eight handles at the appropriate location around the object in lines 27–48.

The position of the handles runs clock-wise, starting with `handles[0]` located in the upper-right corner of the object's bounds.

This discussion addressed the for-instance of the `line_find` method resulting in the location of an object leading to its selection. If, however, the find methods of the objects drawn on the editor's canvas are invoked with no resulting object found, any previously selected object must be deselected.

The `Line` object's `deselect` method introduced in Listing 20.11 shows the steps necessary to remove the handles indicating the active state of the `Line` object.

Listing 20.11 Deselecting a `Line` Object

```
1:  static void line_deselect( GXObjPtr line )
2:  {
3:    if( line->handles && line->num_handles > 0 ) {
```

```
4:      gx_erase_handles( line );
5:
6:      XtFree((char *)line->handles );
7:
8:      line->handles = NULL;
9:      line->num_handles = 0;
10:    }
11:  }
```

As you can see when reviewing Listing 20.11, it is easier to deselect a `Line` object than it is to select one.

The `deselect` method ensure that `handles` exist for the `Line` object and erases them from the screen

```
4:     gx_erase_handles( line );
```

frees the memory associated with them

```
6:     XtFree((char *)line->handles );
```

and returns the `handles` and `num_handles` fields to their initial values:

```
8:     line->handles = NULL;
9:     line->num_handles = 0;
```

In terms of processing, it can seem expensive to destroy the `handles` completely when an object is deselected. However, the placement of the `handles` was very sensitive to the location of the `Line` object on the screen.

As stated earlier, the second purpose of the `line_find` method (beyond locating an object for selection) is to determine the manipulation action requested by the user. If the object is in motion (being actively moved by the user), the handle placement will no longer be correct at the end of the action.

Therefore, the destruction of the `handles` and recreation requires much less overhead to ensure the correctness of the object's `handles`.

This leads to the next section, which introduces the `move` method that controls the repositioning of the `Line` object.

Moving a `Line` Object

The steps required to relocate a `Line` object on the canvas are provided by the `line_move` method seen in Listing 20.12.

Listing 20.12 Moving a `Line` Object

```
1:  static void line_move( GXObjPtr line, XEvent *event )
2:  {
3:    static int x = 0, y = 0;
```

continues

Listing 20.12 Continued

```
4:
5:    GXLinePtr line_data = (GXLinePtr)line->data;
6:    int i;
7:
8:    if( x && y ) {
9:      XDrawLines( XtDisplay(GxDrawArea),
10:                 XtWindow(GxDrawArea), rubberGC,
11:                 line_data->pts, line_data->num_pts,
12:                 CoordModeOrigin );
13:   } else {
14:     /* our first time through */
15:     (*line->erase)( line );
16:
17:     x = event ? event->xbutton.x : 0;
18:     y = event ? event->xbutton.y : 0;
19:   }
20:
21:   if( event ) {
22:     for( i = 0; i < line_data->num_pts; i++ ) {
23:       line_data->pts[i].x += (event->xbutton.x - x);
24:       line_data->pts[i].y += (event->xbutton.y - y);
25:     }
26:
27:     /*
28:      * draw rubberband line
29:      */
30:     XDrawLines( XtDisplay(GxDrawArea),
31:                 XtWindow(GxDrawArea), rubberGC,
32:                 line_data->pts, line_data->num_pts,
33:                 CoordModeOrigin );
34:
35:     x = event->xbutton.x;
36:     y = event->xbutton.y;
37:   } else {
38:     x = 0;
39:     y = 0;
40:   }
41: }
```

Note As you might recall, the `move` and `scale` actions are assigned and invoked from the `process_event` function introduced in Chapter 16, section "Processing User Navigation of Objects," page 334.

The `line_move` function uses the `static` variables `x` and `y` to determine whether this invocation is the first time the function has been called:

```
8:   if( x && y ) {
```

If it is not the first time, lines 9–12 erase the rubber-banding copy of the `Line` object drawn to reflect the object's new location.

However, if `x` and `y` are still 0, the `else` is entered and the `Line` object's `erase` method is called to remove it from the screen so that it can be replaced with the interactive version:

```
15:     (*line->erase)( line );
```

It is important also for the first iteration to save the points associated with the current event:

```
17:     x = event ? event->xbutton.x : 0;
18:     y = event ? event->xbutton.y : 0;
```

Note

Notice that the if-then-else syntax of lines 17–18 enables the absence of a valid event reference for resetting `x` and `y` values to `0` to cancel the `move` action.

If there is a valid event reference, all points that compose the `Line` object are moved the difference of the `static` `x` and `y` values and the current event `x` and `y` components:

```
22:     for( i = 0; i < line_data->num_pts; i++ ) {
23:       line_data->pts[i].x += (event->xbutton.x - x);
24:       line_data->pts[i].y += (event->xbutton.y - y);
25:     }
```

Assuming that every `MotionNotify` event successfully reaches the `line_move` function, the object is relocated one pixel at a time.

The interactive depiction of the object is redrawn to the screen for the new point values and the current event point is saved to the `static` `x` and `y` variables in lines 30–36.

Notice again that lines 22–25 directly apply the distance the cursor has moved during the action to the points contained in the `line_data->pts` array. This is possible because the move transformation does not risk any rounding error.

Whole pixel values are represented by the `x` and `y` components of the event structure and their difference from the `static` `x`, `y` are applied directly to the whole pixel values contained in the object. In other words, it is an entirely integer-based calculation.

As demonstrated in the following section, the `scale` method of the `Line` object has the potential of suffering data loss because of rounding errors and requires a slightly more intense management of the action.

Scaling a `Line` Object

The `scale` method of an object, as introduced in Chapter 16, in the section "Processing User Navigation of Objects," page 334, is assigned as the `active` action when the user selects one of the active objects handles.

The `line_scale` method introduced in Listing 20.13 manages the scale action for the Graphics Editor `Line` object.

Listing 20.13 Scaling a `Line` Object

```
static void line_scale( GXObjPtr line, XEvent *event )
{
  static GXLinePtr tmp_data = NULL;
  GXLinePtr line_data = (GXLinePtr)line->data;

  if( tmp_data ) {
    XDrawLines( XtDisplay(GxDrawArea),
                XtWindow(GxDrawArea), rubberGC,
                tmp_data->pts, tmp_data->num_pts,
                CoordModeOrigin );
   } else {
     /* our first time... */
     (*line->erase)( line );

     tmp_data = (GXLinePtr)XtNew(GXLine);
     tmp_data->num_pts = line_data->num_pts;
     tmp_data->pts = (XPoint *)
        XtMalloc( sizeof(XPoint) * tmp_data->num_pts );

     get_bounds( line_data->pts, line_data->num_pts,
                 &OrigX, &OrigY, &ExntX, &ExntY );
   }

   if( event ) {
     memcpy( (char *)tmp_data->pts, (char *)line_data->pts,
             sizeof(XPoint) * tmp_data->num_pts );

     apply_delta( tmp_data->pts, tmp_data->num_pts,
                  FixedX - event->xbutton.x,
                  FixedY - event->xbutton.y );

      XDrawLines( XtDisplay(GxDrawArea),
                  XtWindow(GxDrawArea), rubberGC,
                  tmp_data->pts, tmp_data->num_pts,
                  CoordModeOrigin );
   } else {
     if( tmp_data ) {
       memcpy( (char *)line_data->pts,
               (char *)tmp_data->pts,
               sizeof(XPoint) * line_data->num_pts );
```

```

            XtFree((char *)tmp_data->pts);
            XtFree((char *)tmp_data);

            tmp_data = NULL;
          }
        }
      }
```

The `line_scale` method, concerned with compounding rounding errors prone to the floating point calculations of scaling the points contained in the `line_data->pts` array, makes a copy of the points each time the function is invoked. Then the delta of a `FixedX` and `FixedY` and current event point components are applied to the original `line_data->pts`. This management of data points ensures that the margin of error introduced by the floating-point calculations is absolutely minimized.

Note

For a review of the calculations performed by the `apply_delta` function as well as the declaration and assignment of the `FixedX` and `FixedY` variables, see Chapter 11, section "Scaling a Line," page 234.

Similar in structure to the `line_move` function, the `line_scale` function uses the `static tmp_data` to determine whether the function is called for the first time

```
  if( tmp_data ) {
```

as a means of updating the interactive `Line` object reflecting the user's scale actions.

Otherwise, the `Line` object is erased

```
    (*line->erase)( line );
```

and the `tmp_data` structure is created:

```
    tmp_data = (GXLinePtr)XtNew(GXLine);
    tmp_data->num_pts = line_data->num_pts;
    tmp_data->pts = (XPoint *)
       XtMalloc( sizeof(XPoint) * tmp_data->num_pts );
```

Another responsibility of the `line_scale` action's first invocation is correctly setting the global variables `OrigX`, `OrigY`, `ExntX`, and `ExntY` to reflect the origin and extents of the object being scaled:

```
    get_bounds( line_data->pts, line_data->num_pts,
                &OrigX, &OrigY, &ExntX, &ExntY );
```

These global variables are used by the scale support function called by the `apply_delta` function introduced in Listing 11.4.

Then, so long as a valid event reference is passed to `line_scale`, the original object points are copied to the temporary structure

```
memcpy( (char *)tmp_data->pts, (char *)line_data->pts,
        sizeof(XPoint) * tmp_data->num_pts );
```

and the delta created by the movement of the mouse cursor is applied to the temporary points:

```
apply_delta( tmp_data->pts, tmp_data->num_pts,
             FixedX - event->xbutton.x,
             FixedY - event->xbutton.y );
```

At some point, the user will release the mouse cursor, ceasing the move action, and the temporary points replace the `Line` object's original points:

```
  memcpy( (char *)line_data->pts,
          (char *)tmp_data->pts,
          sizeof(XPoint) * line_data->num_pts );
```

With no further need for the temporary structure, it is removed from memory

```
  XtFree((char *)tmp_data->pts);
  XtFree((char *)tmp_data);
```

and reset to `NULL` preparing for the next invocation:

```
  tmp_data = NULL;
```

The points now contained in the `line_data->pts` array are a scaled version of the original points with minimal loss of line integrity because of floating-point rounding errors.

Only two more `Line` object methods await discovery before completing the addition of this object to the Graphics Editor.

The next section introduces the `copy` method invoked by selecting the copy control icon from the menu panel.

Copying a `Line` Object

The capability to replicate objects in the Graphics Editor is necessary in order to have an equal number of control functions as draw functions appear in the button panels. (I did not want the menus to be different lengths.)

Truly one of the easiest methods to support, Listing 20.14 introduces the `line_copy` function for reproducing the currently active object on the editor's canvas.

Listing 20.14 Copying a `Line` Object

```
static void line_copy( GXObjPtr line )
{
```

```
3:    int i;
4:    GXLinePtr temp_data;
5:    GXLinePtr line_data = (GXLinePtr)line->data;
6:
7:    (*line->deselect)( line );
8:
9:    temp_data = (GXLine *)XtNew(GXLine);
10:   temp_data->num_pts = line_data->num_pts;
11:
12:   temp_data->pts = (XPoint *)
13:       XtMalloc(sizeof(XPoint) * temp_data->num_pts );
14:   for( i = 0; i < temp_data->num_pts; i++ ) {
15:     temp_data->pts[i].x = line_data->pts[i].x + OFFSET;
16:     temp_data->pts[i].y = line_data->pts[i].y + OFFSET;
17:   }
18:
19:   create_line( NULL, temp_data );
20:   XtFree((char *)temp_data );
21: }
```

Comfortable with the `line_scale` function's requirement to create a temporary copy of the `GXLine` structure, the `line_move` function requires a small step to understanding.

Specifically, after ensuring that the active object's handles are removed from the screen

```
7:   (*line->deselect)( line );
```

a temporary `GXLine` structure is created in lines 9–13 and the original object's points are copied to the new structure, but incremented by `OFFSET` so the new object does not exactly overlay the original:

```
15:     temp_data->pts[i].x = line_data->pts[i].x + OFFSET;
16:     temp_data->pts[i].y = line_data->pts[i].y + OFFSET;
```

This temporary `GXLine` structure is used to create a brand new `Line` object containing the offset copy of the original object's points:

```
19:   create_line( NULL, temp_data );
```

The following section presents the last area of functionality required by the `Line` object, Saving and Restoring the object-specific data.

Saving and Restoring a `Line` Object

The capability to save and restore objects contained in the Graphics Editor, as introduced in Chapter 19, provides a mature level of capability to our application.

The `line_save` method found in Listing 20.15 adheres to the save strategy introduced earlier.

Listing 20.15 Saving a `Line` to a `File` Object

```
1:  static void line_save( FILE *fp, GXObjPtr obj )
2:  {
3:      int i;
4:      GXLinePtr line = (GXLinePtr)obj->data;
5:
6:      fprintf( fp, "LINE [numpts x y x y ...]\n" );
7:      fprintf( fp, "%d\n", line->num_pts );
8:
9:      for( i = 0; i < line->num_pts; i++ ) {
10:         fprintf( fp, "%d %d\n",
11:                  line->pts[i].x, line->pts[i].y );
12:     }
13: }
```

The `line_save` function extracts the `GXLine` data structure from the common portion of the object

```
4:     GXLinePtr line = (GXLinePtr)obj->data;
```

and prepares for saving the data it contains by writing a tagged line reflecting the format of the data to follow

```
6:     fprintf( fp, "LINE [numpts x y x y ...]\n" );
```

and the number of points that will be written:

```
7:     fprintf( fp, "%d\n", line->num_pts );
```

Finally, it writes point pairs one per line to the destination file referenced by `fp`:

```
9:      for( i = 0; i < line->num_pts; i++ ) {
10:        fprintf( fp, "%d %d\n",
11:                 line->pts[i].x, line->pts[i].y );
12:     }
```

The `gxLineLoad` function found in Listing 20.16 shows how the data written by the `line_save` method is restored to the editor.

Listing 20.16 Restoring a `Line` from a `File` Object

```
1:  void gxLineLoad( FILE *fp, GXObjPtr obj )
2:  {
3:      int i;
4:      GXLine line;
5:
6:      fscanf( fp, "%d\n", &line.num_pts );
7:
8:      line.pts = (XPoint *)
9:         XtMalloc(sizeof(XPoint) * line.num_pts);
10:
11:     for( i = 0; i < line.num_pts; i++ ) {
        fscanf( fp, "%hd %hd\n",
                &line.pts[i].x, &line.pts[i].y );
    }

    create_line( obj, &line );
}
```

The `gxLineLoad` function reverses the steps of the `line_save` method by retrieving first the number of points saved to the file

```
    fscanf( fp, "%d\n", &line.num_pts );
```

creating sufficient memory to store them

```
    line.pts = (XPoint *)
       XtMalloc(sizeof(XPoint) * line.num_pts);
```

and then restoring the point pairs from the file:

```
    for( i = 0; i < line.num_pts; i++ ) {
        fscanf( fp, "%hd %hd\n",
                &line.pts[i].x, &line.pts[i].y );
```

The introduction of the method and function required to save and restore the `Line` object completes the introduction of this object into the Graphics Editor.

Next Steps

The `Latex Line` object introduced in this chapter is only one of four point-array–based objects that share the `GXLine` object-specific data structure.

In the next chapter I will introduce the `Pencil` object, which enables a user to create a free-style line using an unlimited number of points.

In this chapter

Chapter 21

`Pencil` Line Object

By choosing the Pencil creation icon seen in Figure 21.1, the user enters the interactive creation mode for a freestyle line.

Figure 21.1

Selecting the Pencil creation icon starts the process for a freestyle line.

The creation of the `Pencil` object begins with the first `ButtonPress` and ends with the next.

During the interim `MotionNotify` events separating the `ButtonPress` events starting and ending the creation, points are added to a points array defining the object under construction.

However, not *every* point is put into the array. This approach would require larger point array storage with no added value because only points that alter the slope of the implied line segment for the `Pencil` object being created are important enough for storage.

This as well as other aspects of managing the `Pencil` object creation are introduced in the sections that follow.

Note

The code listings introduced in this chapter are targeted for placement in the `gxLine.c` source file. Additionally, functions presented in the listings that are not defined `static` should have a corresponding prototype placed in the `gxProtos.h` header file.

Creating a `Pencil` Object

The `gx_pencil` function introduced in Listing 21.1 is assigned as the function to invoke when the Pencil creation icon is selected from the menu panel through the definition of the `gxDrawIcons` array found in Chapter 13:

```
{ &pen_icon,  (void (*)(void))gx_pencil,
 "Draw a freestyle line..."      },
```

Listing 21.1 The `gx_pencil` Function

```
void gx_pencil( XEvent *event )
{
  static GXLinePtr rubber_pencil = NULL;
  static int       ptCnt         = 0;

  if( event == NULL ) {
    rubber_pencil = NULL;
  } else {
   /*
    * remove any current rubber banding...
    */
    if( rubber_pencil ) {
      XDrawLines( XtDisplay(GxDrawArea),
                  XtWindow(GxDrawArea), rubberGC,
                  rubber_pencil->pts,
                  rubber_pencil->num_pts,
                  CoordModeOrigin );
    }

    switch( event->type ) {
      case ButtonPress:
        if( rubber_pencil == NULL ) {
          rubber_pencil =
             update_pencil( event, rubber_pencil );
          set_cursor( PENCIL_MODE );
        } else {
          create_line( NULL, rubber_pencil );
          gx_refresh();

          set_cursor( NORMAL_MODE );
          rubber_pencil = 0;
          ptCnt = 0;
        }
      break;

      case ButtonRelease:
      case MotionNotify:
        /*
         * update the GXLine structure based on the
         * new point and current location of the mouse
         */
        if( rubber_pencil ) {
          (void)update_pencil( event, rubber_pencil );
          XDrawLines( XtDisplay(GxDrawArea),
                      XtWindow(GxDrawArea), rubberGC,
                      rubber_pencil->pts,
                      rubber_pencil->num_pts,
                      CoordModeOrigin );
        }
      break;
    }
  }
}
```

The `gx_pencil` function is structured identically to the `gx_line` function controlling the creation of the `Latex Line` object, with one exception. Where the `gx_line` function invoked the `update_line` function to record the event point in the temporary `pts` array being constructed by the user, the `gx_pencil` function employs the `update_pencil` function introduced in Listing 21.2.

Listing 21.2 The `update_pencil` Function

```
static GXLinePtr update_pencil( XEvent *event, GXLinePtr pencil )
{
  static GXLine    pen = {NULL, 0};
  GXLinePtr penPtr = &pen;

  int x = -1, y = -1;

  gx_new_vertex( event, pencil, &x, &y );

  /* start over if the user is being silly... */
  if( pen.num_pts > 1024 ) {
    pen.num_pts = 0;
  }

  if( (x > 0) && (y > 0) ) {
    if( !pencil ) {
      pen.pts = (XPoint *)XtMalloc( sizeof(XPoint) );
      pen.num_pts = 0;
    } else {
      pen.pts = (XPoint *)
        XtRealloc( (char *)pen.pts,
                   sizeof(XPoint) * (pen.num_pts + 1));
    }

    pen.pts[pen.num_pts].x = x;
    pen.pts[pen.num_pts].y = y;

    pen.num_pts++;
  }

  return penPtr;
}
```

The `update_pencil` function begins by immediately calling the function `gx_new_vertex` to determine whether the current event point should be added to the `pts` array being constructed by the `Pencil` object creation process:

```
gx_new_vertex( event, pencil, &x, &y );
```

Consideration of this function will follow in a moment; however, whether a point is added to the array or not, the function continues by ensuring that some ridiculously large number of points had not been requested

```
if( pen.num_pts > 1024 ) {
  pen.num_pts = 0;
}
```

resetting to the beginning if necessary.

Following this validation, the function sees whether a valid point was returned from the `gx_new_vertex` function:

```
if( (x > 0) && (y > 0) ) {
```

Because the initial `x` and `y` values were set to –1, any value returned from `gx_new_vertex` indicates the need to process them.

The `update_pencil` function then determines, based on the value of the `GXLine` reference `pen`, whether this is the first time through the function

```
if( !pencil ) {
```

to determine which allocation routine to invoke to create space for the point being added. On the first pass through this function, the `pen` value is `NULL` and the `XtMalloc` function is used to gain the initial memory assignment:

```
pen.pts = (XPoint *)XtMalloc( sizeof(XPoint) );
pen.num_pts = 0;
```

On subsequent invocations the `XtRealloc` function is used to expand the memory currently assigned the `pen` structure reference:

```
XtRealloc( (char *)pen.pts,
           sizeof(XPoint) * (pen.num_pts + 1));
```

Finally, the new point is added to the array

```
pen.pts[pen.num_pts].x = x;
pen.pts[pen.num_pts].y = y;
```

and the running total of the point count incremented accordingly:

```
pen.num_pts++;
```

Look again at the `gx_new_vertex` function introduced in Listing 21.3 and used by `update_pencil` to determine whether the point contained in the event structure is worthy of being added to the points array for the `Pencil` object under creation.

Listing 21.3 Determine Whether a New Vertex Should Be Stored

```
1:  static void gx_new_vertex( XEvent *xe,
2:                              GXLinePtr upd, int *x, int *y )
3:  {
4:    float a = -1.0, b = -1.0;
5:
6:    if( upd &&
7:        (upd->num_pts > 0) && (xe->xbutton.x > 0) ) {
8:
9:      /*
10:      *  see if the slope has changed
11:      */
12:     a = (float)upd->pts[upd->num_pts - 1].y /
13:         (float)upd->pts[upd->num_pts - 1].x;
14:
15:     b = (float)xe->xbutton.y / (float)xe->xbutton.x;
16:
17:     if( a != b ) {
18:       *x = xe->xbutton.x;
19:       *y = xe->xbutton.y;
20:     }
21:   }
22: }
```

First ensuring that there is a valid point for comparison contained in the `upd` structure passed as the second parameter to the function

```
6:    if( upd &&
7:        (upd->num_pts > 0) && (xe->xbutton.x > 0) ) {
```

the `gx_new_vertex` also ensures that the value of the `x` component of the event point is not 0 because this would result in a fatal application error when used as the denominator of a divide calculation later in the function.

After the validation step of lines 6–7, two slope values are calculated. The first

```
12:     a = (float)upd->pts[upd->num_pts - 1].y /
13:         (float)upd->pts[upd->num_pts - 1].x;
```

is for the previous point added to the update structure. The second

```
15:     b = (float)xe->xbutton.y / (float)xe->xbutton.x;
```

is for the current event point.

Note

To review slope calculations for a line, return to Chapter 10, “Trigonometric and Geometric Functions,” in the section “Calculating Slope,” page 216.

The function then compares the two slope calculations to determine whether they are equal:

```
    if( a != b ) {
```

Only if they are not equal are they returned to the calling function for addition into the `pts` array of the object being created:

```
      *x = xe->xbutton.x;
      *y = xe->xbutton.y;
```

No value is gained by adding points with like slopes to the array, because the `XDrawLines` function will ensure that points connecting endpoints are drawn to the screen.

This method of evaluating points before adding them to the object minimizes the space required to represent them by not including needless information.

When the `ButtonPress` event marking the end of the creation process for the `Pencil` object is received, the `create_line` function introduced in the previous chapter is invoked to create a `Line` object:

```
          create_line( NULL, rubber_pencil );
```

As described in the next section, no other functions are unique to the management of the `Pencil` object.

`Pencil` Object Management

The `Pencil` object, through requiring a creation process unique to the object, ends up being a point-array–based object indistinguishable from the `Latex Line` object.

The `Pencil` object exists (along with the `Latex Line`) as an array of points contained within a `GXLine` structure. This enables all the methods assigned by the `create_line` function to adequately manage the `Pencil` object as it does the `Latex Line`, introduced in Chapter 20.

Next Steps

Two other point-array–based objects exist that employ the `GXLine` data structure to represent their object-specific data. These, too, will end up as point-array–based objects using the same methods for management as the `Latex Line` and `Pencil` objects. However, like the `Pencil` object, they require a unique creation process.

The following chapter introduces the creation process for two other point-array–based objects supported by the Graphics Editor: the `Box` object and the `Arrow` object.

In this chapter

- *The* `Box` *Object*
- *The* `Arrow` *Object*
- *Next Steps*

Chapter 22

Object Templates

The `Box` and `Arrow` objects supported by the editor are considered template objects because the interactive process of creating these objects enforces the shape appropriate to the object.

The following sections introduce the `Box` and `Arrow` objects. You will notice many similarities between these objects and the `Latex Line` and `Pencil` objects introduced in previous chapters.

Note

The code listings introduced in this chapter are targeted for placement in the `gxLine.c` source file. Additionally, functions presented in the listings that are not defined `static` should have a corresponding prototype placed in the `gxProtos.h` header file.

The `Box` Object

By choosing the icon shown in Figure 22.1, a user begins the `Box` object creation process.

Figure 22.1

The `Box` *object creation icon.*

During the creation of the `Box` object, the user moves the mouse cursor to the canvas window and presses and holds the mouse button while dragging to form the box dimensions desired.

Releasing the mouse button ends the creation process, and a Box object consistent with the size specified during the interactive phase of the creation is added to the list of objects known to the editor.

The gx_box function introduced in Listing 22.1 satisfies the assignment made to the gxDrawIcons array in Chapter 13, “Application Structure,” for the function to invoke when the gx_box icon is selected:

```
{ &box_icon,  (void (*)(void))gx_box,
  "Draw a square or rectangle..." },
```

Listing 22.1 The gx_box Function

```
void gx_box( XEvent *event )
{
  static XRectangle *rubber_box = NULL;

  if( event == NULL ) {
    rubber_box = NULL;
  } else {
    /*
     * remove the current rubberband if there is one
     */
    if( rubber_box ) {
      XDrawRectangle(XtDisplay(GxDrawArea),
                     XtWindow(GxDrawArea), rubberGC,
                     rubber_box->x, rubber_box->y,
                     rubber_box->width,
                     rubber_box->height );
    }

    switch( event->type ) {
      case ButtonPress:
        rubber_box = update_box( event, NULL );
        set_cursor( EDIT_MODE );
      break;

      case ButtonRelease:
        if( rubber_box ) {
          create_line(NULL, line_from_box(rubber_box));
          gx_refresh();
        }
        set_cursor( NORMAL_MODE );
        rubber_box = NULL;
      break;

      case MotionNotify:
        if( rubber_box ) {
          (void)update_box( event, rubber_box );
          XDrawRectangle(XtDisplay(GxDrawArea),
                         XtWindow(GxDrawArea),
                         rubberGC,
                         rubber_box->x, rubber_box->y,
                         rubber_box->width,
                         rubber_box->height );
        }
      break;
    }
  }
}
```

The gx_box function differs from the creation routine gx_line and gx_pencil only in the form of the temporary XRectangle rubber_box data structure employed, the update applied to it

```
36:           (void)update_box( event, rubber_box );
```

and the processing of the XRectangle to convert it to a GXLine structure:

```
27:           create_line(NULL, line_from_box(rubber_box));
```

The update_box function is introduced in Listing 22.2 and the line_from_box conversion routine is introduced in Listing 22.3.

Listing 22.2 Updating the Temporary Box Points

```
static
XRectangle *update_box( XEvent *event, XRectangle *upd )
{
  static int fix_x = 0, fix_y = 0;
  static XRectangle box;

  XRectangle *boxPtr = &box;

  if( upd == NULL ) {
    fix_x = event->xbutton.x;
    fix_y = event->xbutton.y;
  }

  box.x = min( fix_x, event->xbutton.x );
  box.y = min( fix_y, event->xbutton.y );

  box.width  = max( fix_x, event->xbutton.x )-box.x;
  box.height = max( fix_y, event->xbutton.y )-box.y;

  return boxPtr;
}
```

The challenge of the update_box function is to determine when the user flips the box over one of the x- or y-axis during creation.

To understand this, first consider that the upper-left corner of the box is placed at the ButtonPress event location indicating the start of the box creation, and the lower corner of the box is placed at the ButtonRelease event location ending the creation.

When the event location marking the end of the creation process is less than the event location starting the creation, the object has flipped. This is important to catch because we don't want to specify a negative width or height value for the rectangle defining the `Box` object. It is the responsibility of the `update_box` function to account for a flipped condition.

The `update_box` function begins by determining whether this is the first call to this function for the current creation as indicated by the value of `upd` being `NULL`:

```
if( upd == NULL ) {
   fix_x = event->xbutton.x;
   fix_y = event->xbutton.y;
 }
```

At the first invocation for a creation, the `static` variables `fix_x` and `fix_y` must be assigned the corresponding components of the event point.

This fixed point will enable the function to prevent subsequent events resulting in the box being flipped over either the x- or the y-axis.

The `XRectangle` reference by `box` is then assigned an `x`, `y` location that is the smaller of the values `fix_x`, `fix_y`, and the corresponding event point components:

```
box.x = min( fix_x, event->xbutton.x );
box.y = min( fix_y, event->xbutton.y );
```

From the location assignment of `box`, the `width` and `height` are set as the larger of `fix_x`, `fix_y`, and the corresponding event point components less whatever `x`, `y` values were assigned to `box`:

```
box.width  = max( fix_x, event->xbutton.x )-box.x;
box.height = max( fix_y, event->xbutton.y )-box.y;
```

Following the successful update of the interactive box defining the wishes of the user, the creation process will end when she releases the mouse button. When the creation ends, the `XRectangle` used during the creation process must be converted to a `GXLine` structure for a point-array–based object to result from this process.

Listing 22.3 introduces the `line_from_box` function for converting between the two structures.

Listing 22.3 Converting a `Box` Object to a `Line` Object

```
static GXLinePtr line_from_box( XRectangle *box )
{
  static GXLine line;

  line.pts = (XPoint *)XtMalloc( sizeof(XPoint) * 5);
  line.num_pts = 5;

8:    line.pts[0].x = box->x;
9:    line.pts[0].y = box->y;
10:
11:   line.pts[1].x = box->x + box->width;
12:   line.pts[1].y = box->y;
13:
14:   line.pts[2].x = box->x + box->width;
15:   line.pts[2].y = box->y + box->height;
16:
17:   line.pts[3].x = box->x;
18:   line.pts[3].y = box->y + box->height;
19:
20:   line.pts[4].x = box->x;
21:   line.pts[4].y = box->y;
22:
23:   return &line;
24: }
```

The steps required to convert an `XRectangle` to a `GXLine` structure are very simple and only require that we ensure the resulting `GXLine` object is closed by repeating the first point as the last point.

The `line_from_box` function creates space enough for five points in the `GXLine pts` array

```
5:    line.pts = (XPoint *)XtMalloc( sizeof(XPoint) * 5);
6:    line.num_pts = 5;
```

and then it assigns each corner of the rectangle to a corresponding element of the `pts` array in lines 8–18, repeating the first point in the last element:

```
20:   line.pts[4].x = box->x;
21:   line.pts[4].y = box->y;
```

Without this last step, the resulting line object would appear as a sideways U with the open end facing to the left.

With a `GXLine` structure representing the box created by the user, the `create_line` function is called and management continues for this object as it does for both the `Latex Line` and `Pencil` objects.

Considerations similar to those for the `Box` object exist for the `Arrow` object introduced in the next section.

The `Arrow` Object

By choosing the Arrow creation icon shown in Figure 22.2 from the Graphics Editor menu panel, a user is able to begin the creation process for the `Arrow` object.

Figure 22.2

The Arrow creation icon.

The `Arrow` object, like the `Box` object, is considered a template object because the Arrow shape is enforced by the creation process. Also similar to the `Box` object is the fact that the `ButtonPress` event begins the creation process and the `ButtonRelease` event ends it.

The event point associated with the `ButtonPress` determines the upper-left corner of the object being created and the `ButtonRelease` event point marks the lower-right corner of the object.

Listing 22.4 introduces the `gx_arrow` function assigned in the `gxDrawIcons` array in Chapter 13.

Listing 22.4 The `gx_arrow` Function

```
void gx_arrow( XEvent *event )
{
  static XRectangle *rubber_box = NULL;
  GXLinePtr arrow;

  if( event == NULL ) {
    rubber_box = NULL;
  } else {
    if( rubber_box ) {
      arrow = arrow_from_box( rubber_box );
      XDrawLines(XtDisplay(GxDrawArea),
                 XtWindow(GxDrawArea), rubberGC,
                 arrow->pts, arrow->num_pts,
                 CoordModeOrigin );
    }

    switch( event->type ) {
      case ButtonPress:
        rubber_box = update_box( event, NULL );
        set_cursor( EDIT_MODE );

        break;

      case ButtonRelease:
        if( rubber_box ) {
          create_line(NULL,arrow_from_box(rubber_box));
          gx_refresh();
        }
        set_cursor( NORMAL_MODE );
        rubber_box = NULL;

      break;

34:       case MotionNotify:
35:         if( rubber_box ) {
36:           (void)update_box( event, rubber_box );
37:
38:           arrow = arrow_from_box( rubber_box );
39:           XDrawLines(XtDisplay(GxDrawArea),
40:                      XtWindow(GxDrawArea), rubberGC,
41:                      arrow->pts, arrow->num_pts,
42:                      CoordModeOrigin );
43:
44:         }
45:       break;
46:     }
47:   }
48: }
```

The gx_arrow function should be immediately familiar because it follows the same structure as the gx_box (and its predecessors).

The only difference, in fact, between the gx_box function and the gx_arrow function is the arrow_from_box function used to convert from the XRectangle structure to the GXLine structure in the following lines:

```
26:           create_line(NULL,arrow_from_box(rubber_box));
38:           arrow = arrow_from_box( rubber_box );
```

The arrow_from_box function is defined in Listing 22.5.

Listing 22.5 Creating an Arrow from a Box

```
static GXLinePtr arrow_from_box( XRectangle *box )
{
  static GXLine arrow;
  static XPoint pts[10];

  pts[0].x = box->x + (box->width/10);
  pts[0].y = box->y + box->height;

  pts[1].x = box->x + (box->width  / 4);
  pts[1].y = box->y + box->height - (box->height / 8);

  pts[2].x = box->x + (box->width  / 4);
  pts[2].y = box->y + (box->height / 2);

  pts[3].x = box->x;
  pts[3].y = box->y + (box->height / 2);

  pts[4].x = box->x + (box->width  / 2);
  pts[4].y = box->y;

  pts[5].x = box->x + box->width;
  pts[5].y = box->y + (box->height / 2);
```

continues

```
23:
24:   pts[6].x = box->x + ((box->width * 3)  / 4);
25:   pts[6].y = box->y + (box->height / 2);
26:
27:   pts[7].x = box->x + ((box->width * 3)  / 4);
28:   pts[7].y = box->y + box->height - (box->height / 8);
29:
30:   pts[8].x = box->x + box->width - (box->width / 10);
31:   pts[8].y = box->y + box->height;
32:
33:   arrow.pts     = pts;
34:   arrow.num_pts = 9;
35:
36:   return &arrow;
37: }
```

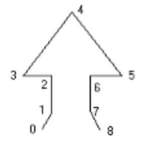
The `arrow_from_box` function defines enough room to hold 10 `XPoint` structures:

```
4:   static XPoint pts[10];
```

The arrangement of points in the array is illustrated in Figure 22.3.

Figure 22.3

An illustration of the arrangement of points in an Arrow.



Lines 6–31 meticulously assign the point values based on the proportions and ratios consistent with point positions shown in Figure 22.3.

Note

Nothing is truly scientific about dividing a box into a series of points to achieve the shape of an arrow.

I accomplished it through trial and error until the results were aesthetically pleasing.

It is important that ratios of the box be used to create the points defining the arrow, however, in order to enable the aspect ratio (width to height proportion) to be honored in the resulting `GXLine` structure.

The point array is then assigned as the `pts` value of the `GXLine` structure along with the correct number of points

```
33:   arrow.pts     = pts;
34:   arrow.num_pts = 9;
```

and the address of the GXLine structure arrow is returned for use in either drawing the interactive object being navigated by the user or in the creating the final point-array–based object:

```
36:   return &arrow;
```

This completes the introduction of the Box and Arrow template objects supported by the Graphics Editor.

Next Steps

With the Box and Arrow objects comfortably at rest in the Graphics Editor project, you have successfully added a total of four objects. Because they are all point-array–based objects they are able to borrow heavily from a core structure.

In Chapter 23, the Arc object is added to the list of objects supported by the editor. Because the representation of an arc is not similar to the GXLine objects seen so far, there will be differences to point out. However, the basic event-driven creation, moving, and scaling processes introduced for the Latex Line and inherited by the Pencil, Box, and Arrow objects will remain the same.

Chapter 23

In this chapter

Arc Object

By choosing the Arc creation icon seen in Figure 23.1, the user begins the process of creating an `Arc` object.

Figure 23.1

Choosing the Arc creation icon begins the process of creating an `Arc` *object.*

The interactive process of creating an `Arc` object is similar to creating the `Box` or `Arrow` objects discussed in Chapter 22, "Object Templates."

Specifically, a `ButtonPress` event begins the creation process and a `ButtonRelease` event ends it. The event point corresponding to the `ButtonPress` assigns the location of the upper-left corner of the bounding box that is to contain the `Arc` object, whereas the event point for the `ButtonRelease` determines the lower-right corner of this box.

This chapter presents the internal methods and supporting functions defined for the `Arc` object. These routines control all aspects of managing the different features of the object.

Note

The code listings introduced in this chapter are targeted for placement in the `gxArc.c` source file. Additionally, functions presented in the listings that are not defined `static` should have a corresponding prototype placed in the `gxProtos.h` header file.

Creating an Arc Object

The definition of the gxDrawIcons array introduced in Chapter 13

```
{ &arc_icon,  (void (*)(void))gx_arc,
  "Draw a circle..."              },
```

assigned the Arc icon the gx_arc function for invocation when selected by the user. This function is defined in Listing 23.1 and serves as the entry point to the creation process of the Arc object.

Listing 23.1 The gx_arc Function

```
void gx_arc( XEvent *event )
{
  static XArc *rubber_arc = NULL;

  if( event == NULL ) {
    rubber_arc = NULL;
  } else {
    if( rubber_arc &&
        rubber_arc->width > 0 &&
        rubber_arc->height > 0) {
      XDrawArc( XtDisplay(GxDrawArea),
               XtWindow(GxDrawArea), rubberGC,
               rubber_arc->x, rubber_arc->y,
               rubber_arc->width, rubber_arc->height,
               rubber_arc->angle1, rubber_arc->angle2);
    }

    switch( event->type ) {
      case ButtonPress:
        /*
         * initailize an XArc structure to retain
         * the arc info during rubber banding
         */
        rubber_arc = update_arc( event, NULL );
        set_cursor( EDIT_MODE );
      break;

    case ButtonRelease:
      /*
       * create the arc, and mark it selected as
       * the current object
       */
      if( rubber_arc &&
          rubber_arc->width && rubber_arc->height ) {
        create_arc( NULL, rubber_arc );
        gx_refresh();
      }
      set_cursor( NORMAL_MODE );

41:       break;
42:     case MotionNotify:
43:       /*
44:        * update the XArc structure based on the
45:        * new location of the mouse pointer
46:        */
47:       if( rubber_arc ) {
48:         (void)update_arc( event, rubber_arc );
49:
50:         /*
51:          * redraw the rubberbanding arc
52:          */
53:         if( rubber_arc->width && rubber_arc->height ) {
54:           XDrawArc( XtDisplay(GxDrawArea),
55:                XtWindow(GxDrawArea), rubberGC,
56:                rubber_arc->x, rubber_arc->y,
57:                rubber_arc->width, rubber_arc->height,
58:                rubber_arc->angle1, rubber_arc->angle2);
59:         }
60:       }
61:       break;
39:       rubber_arc = NULL;
62:     }
63:   }
64: }
```

Structured similarly to the creation functions reviewed in previous chapters, the `gx_arc` function is driven by the events it receives informing of user navigation.

The function begins by testing for a valid event reference passed by the caller:

```
5:    if( event == NULL ) {
6:      rubber_arc = NULL;
```

Remember that a `NULL` event enables the function to reset and cancel any pending creation process.

If a valid event has been sent, the `gx_arc` function tests the presence of a valid Arc reference stored in the `rubber_arc` variable:

```
8:      if( rubber_arc &&
9:          rubber_arc->width > 0 &&
10:         rubber_arc->height > 0) {
```

If a valid Arc definition exists, the function erases the previously drawn interactive `Arc` object pending the update of the `rubber_arc` structure:

```
11:       XDrawArc( XtDisplay(GxDrawArea),
12:                XtWindow(GxDrawArea), rubberGC,
13:                rubber_arc->x, rubber_arc->y,
14:                rubber_arc->width, rubber_arc->height,
15:                rubber_arc->angle1, rubber_arc->angle2);
```

The gx_arc function then switches, based on the type of event sent, and takes the appropriate action:

```
    switch( event->type ) {
```

When the user presses the mouse button, the creation process begins and the rubber_arc structure is assigned its initial values:

```
        rubber_arc = update_arc( event, NULL );
```

The movement of the cursor further updates the rubber_arc structure to reflect the user's navigation:

```
        (void)update_arc( event, rubber_arc );
```

However, when the mouse button is released, an actual Arc object is created:

```
        create_arc( NULL, rubber_arc );
```

Because the structure of the gx_arc function has been thoroughly established with the introduction of similar functions previously, I won't go into great detail. Interesting, however, are the functions unique to the creation of the Arc object, specifically update_arc, introduced in Listing 23.2, and create_arc found in Listing 23.3.

Listing 23.2 The `update_arc` Function

```
static XArc *update_arc( XEvent *event, XArc *upd )
{
  static int fix_x = 0, fix_y = 0;
  static XArc arc;

  XArc *arcPtr = &arc;
  int x1, y1, x2, y2, rx, ry;

  if( upd == NULL ) {
    arc.angle1 = 0*64;
    arc.angle2 = 360*64;

    fix_x = event->xbutton.x;
    fix_y = event->xbutton.y;
  }

  rx = event->xbutton.x - fix_x;
  ry = event->xbutton.y - fix_y;

  x1 = fix_x + rx;
  x2 = fix_x - rx;
  y1 = fix_y + ry;
  y2 = fix_y - ry;

  arc.x = min(x1, x2);
  arc.y = min(y1, y2);

  arc.width  = max(x1, x2) - arc.x;
  arc.height = max(y1, y2) - arc.y;

  return arcPtr;
}
```

The `update_arc` function has similar responsibilities to the `update_box` function introduced in Chapter 22. The `update_arc` function must manage when the distance the cursor has traveled since the start of the creation could result in negative values for the width and height fields of the `XArc` structure.

Starting with a test for a `NULL upd` reference to mark the beginning of the creation process

```
  if( upd == NULL ) {
```

the `update_arc` function makes the necessary initializations for processing subsequent events correctly:

```
    arc.angle1 = 0*64;
    arc.angle2 = 360*64;
    fix_x = event->xbutton.x;
    fix_y = event->xbutton.y;
```

Thinking ahead to consecutive calls to the `update_arc` function during the creation process, the deltas (or radii) for the x and y-axis are calculated from the fixed points and the current event point:

```
  rx = event->xbutton.x - fix_x;
  ry = event->xbutton.y - fix_y;
```

From the x and y-axis radius values `rx` and `ry`, the point defining the corners of the bounding box can be found:

```
  x1 = fix_x + rx;
  x2 = fix_x - rx;
  y1 = fix_y + ry;
  y2 = fix_y - ry;
```

Using these points, you assign the smaller of the pairs as the x and y origin in the `XArc` structure

```
  arc.x = min(x1, x2);
  arc.y = min(y1, y2);
```

and the greater of them as the horizontal and vertical extent:

```
  arc.width  = max(x1, x2) - arc.x;
  arc.height = max(y1, y2) - arc.y;
```

When the interactive mode of the Arc creation yields the object desired by the user, she releases the mouse button and ends the creation process. When this happens, the actual `Arc` object is created by a call to the `create_arc` function introduced in Listing 23.3.

Listing 23.3 The `create_arc` Function

```
1:  static void create_arc( GXObjPtr _obj, XArc *arc )
2:  {
3:    XArc *arc_data;
4:    GXObjPtr obj = _obj;
5:
6:    if( obj == NULL ) {
7:        obj = gx_create_obj();
8:    }
9:
10:   arc_data  = (XArc *)XtNew( XArc );
11:   obj->data = (void *)arc_data;
12:
13:   arc_data->width  = arc->width;
14:   arc_data->height = arc->height;
15:
16:   arc_data->x = arc->x;
17:   arc_data->y = arc->y;
18:
19:   arc_data->angle1 = arc->angle1;
20:   arc_data->angle2 = arc->angle2;
21:
22:   obj->draw  = arc_draw;
23:   obj->erase = arc_erase;
24:   obj->find  = arc_find;
25:   obj->move  = arc_move;
26:   obj->scale = arc_scale;
27:   obj->copy  = arc_copy;
28:
29:   obj->save  = arc_save;
30:
31:   obj->select   = arc_select;
32:   obj->deselect = arc_deselect;
33:
34:   gx_add_obj( obj );
35: }
```

As with the `create_line` function exhausted in earlier chapters, the `create_arc` function accounts for being called with the common object portion already created in order to support the restore function introduced later in this chapter.

Barring the presence of a `GXObj` reference, the `create_arc` function must request one:

```
7:       obj = gx_create_obj();
```

The function creates a new reference to an XArc structure and assigns it to the data field of the GXObj:

```
10:   arc_data  = (XArc *)XtNew( XArc );
11:   obj->data = (void *)arc_data;
```

The values of the XArc defined by the user during creation are transferred to the new reference:

```
13:   arc_data->width  = arc->width;
14:   arc_data->height = arc->height;
```

and

```
16:   arc_data->x = arc->x;
17:   arc_data->y = arc->y;
```

and

```
19:   arc_data->angle1 = arc->angle1;
20:   arc_data->angle2 = arc->angle2;
```

Finally, the function is ready to assign the methods to the GXObj that will properly manage and control the Arc data portion of this object (lines 22–32).

The last thing the create_arc function does is ensure that the newly created object is appended to the list of objects currently managed by the editor application:

```
34:   gx_add_obj( obj );
```

As was true with the Line objects in Chapters 20–22, for an object to be visible on the canvas it must be drawn. The following section introduces the methods that draw and erase the Arc object.

Drawing and Erasing an Arc Object

Again following the form of the line_draw and line_erase methods assigned to Line objects, except for the arc_erase being responsible for erasing the object's handles, the arc_draw and arc_erase methods differ only in the value passed for the tile flag as seen in Listing 23.4.

Listing 23.4 Drawing and Erasing an Arc Object

```
1:  static void draw_erase( GXObjPtr arc, Boolean tile )
2:  {
3:    GC gc;
4:    XArc *arc_data;
5:
6:    gc = gx_allocate_gc( arc, tile );
7:    arc_data = (XArc *)arc->data;
```

continues

Listing 23.4 Continued

```

  XDrawArc( XtDisplay(GxDrawArea),
            XtWindow(GxDrawArea), gc,
            arc_data->x,      arc_data->y,
            arc_data->width,  arc_data->height,
            arc_data->angle1, arc_data->angle2 );

  XtReleaseGC( GxDrawArea, gc );
}

static void arc_draw( GXObjPtr obj )
{
  draw_erase( obj, False );
}

static void arc_erase( GXObjPtr obj )
{
  gx_erase_handles( obj );
  draw_erase( obj, True );
}
```

True for any of the objects added to the editor thus far, if the object is visible it can be selected by the user. The following section introduces the method for determining whether an event successfully selected an `Arc` object.

Finding an `Arc` Object

The work for finding an `Arc` object was done in Chapter 10, “Trigonometric and Geometric Functions.” The `arc_find` method was introduced in Listing 10.2 in the section “Calculating Point and Arc Intersections,” page 218, which explains the mathematics required to perform the necessary calculations.

If successfully located by the event passed to the `arc_find` method, the object must be updated as selected. The following section provides the `select` and `deselect` methods assigned to the `Arc` object.

Selecting and Deselecting an `Arc` Object

The `select` method of the `arc_object` introduced in Listing 23.5 is responsible for creating and displaying the bounding handles associated with the object.

Listing 23.5 Selecting an `Arc` Object

```
static void arc_select( GXObjPtr arc )
{
  arc_bounding_handles( arc );
  gx_draw_handles( arc );
}
```

The arc_select method is a simple process until we introduce the arc_bounding_handles function in Listing 23.6, which creates the handles assigned to the Arc object.

Listing 23.6 Creating Arc Handles

```
static void arc_bounding_handles( GXObjPtr gx_arc )
{
  int i;
  XArc *arc = gx_arc->data;

  gx_arc->handles = (XRectangle *)XtMalloc(sizeof(XRectangle) * 8);
  gx_arc->num_handles = 8;

  if( gx_arc->handles == NULL ) {
    fprintf(stderr, "Alloc failed for arc handles" );
    gx_arc->num_handles = 0;
    return;
  }

  for( i = 0; i < 8; i++ ) {
    gx_arc->handles[i].x =
    gx_arc->handles[i].y = 0;

    gx_arc->handles[i].width  = HNDL_SIZE;
    gx_arc->handles[i].height = HNDL_SIZE;
  }

  gx_arc->handles[0].x = arc->x - HNDL_SIZE - HNDL_OFFSET;
  gx_arc->handles[0].y = arc->y - HNDL_SIZE - HNDL_OFFSET;

  gx_arc->handles[1].x = arc->x + (arc->width/2) - HNDL_OFFSET;
  gx_arc->handles[1].y = arc->y - HNDL_SIZE - HNDL_OFFSET;

  gx_arc->handles[2].x = arc->x + arc->width + HNDL_OFFSET;
  gx_arc->handles[2].y = arc->y - HNDL_SIZE - HNDL_OFFSET;

  gx_arc->handles[3].x = arc->x + arc->width  + HNDL_OFFSET;
  gx_arc->handles[3].y = arc->y +(arc->height/2) - HNDL_OFFSET;

  gx_arc->handles[4].x =
                   arc->x + arc->width  + HNDL_OFFSET;
  gx_arc->handles[4].y =
                   arc->y + arc->height + HNDL_OFFSET;

  gx_arc->handles[5].x =
                   arc->x + (arc->width/2) - HNDL_OFFSET;
  gx_arc->handles[5].y =
                   arc->y + arc->height + HNDL_OFFSET;

  gx_arc->handles[6].x = arc->x - HNDL_SIZE - HNDL_OFFSET;
```

continues

Listing 23.6 Continued

```
  gx_arc->handles[6].y =
                   arc->y + arc->height + HNDL_OFFSET;

  gx_arc->handles[7].x = arc->x - HNDL_SIZE - HNDL_OFFSET;
  gx_arc->handles[7].y =
                   arc->y + (arc->height/2) - HNDL_OFFSET;
}
```

The `arc_bounding_handles` function creates eight handles for positioning at every corner and every side of the object:

```
  gx_arc->handles = (XRectangle *)XtMalloc(sizeof(XRectangle) * 8);
  gx_arc->num_handles = 8;
```

Following steps to ensure that the allocation succeeds for the handles, a `for` loop is entered to initialize the width and height fields:

```
    gx_arc->handles[i].width  = HNDL_SIZE;
    gx_arc->handles[i].height = HNDL_SIZE;
```

Finally, a process identical to assigning the positions of the `Line` object handles introduced in Chapter 20, "`Latex Line` Object," in the section "Selecting and Deselecting a `Line` Object," page 392, is used to place the arc bounding handles. One difference, however, is that the `XArc` structure stored in the object inherently provides the bounds of the object, eliminating the need to calculate them.

A selected object is a candidate for deselecting; therefore the `arc_deselect` method is defined in Listing 23.7 to support this requirement.

Listing 23.7 Deselecting an `Arc` Object

```
static void arc_deselect( GXObjPtr arc )
{
  if( arc->handles && arc->num_handles > 0 ) {
    gx_erase_handles( arc );

    XtFree((char *)arc->handles );

    arc->handles = NULL;
    arc->num_handles = 0;
  }
}
```

The `deselect` method ensures that the handles exist for this object and takes steps to remove them from the screen and release the memory they occupy, resetting the object fields to reflect their defunct state:

```
    arc->handles = NULL;
    arc->num_handles = 0;
```

Because the user has selected an Arc object for the purpose of manipulating it in some way, the following sections present the arc_move and arc_scale functions.

Moving an Arc Object

The steps required to relocate an Arc object on the screen are provided by the arc_move function, seen in Listing 23.8.

Listing 23.8 The arc_move Function

```
static void arc_move( GXObjPtr arc, XEvent *event )
{
  static int x = 0, y = 0;
  XArc *arc_data = (XArc *) arc->data;

  if( x && y ) {
    /*
     * erase the rubberband arc
     */
    XDrawArc( XtDisplay(GxDrawArea),
              XtWindow(GxDrawArea), rubberGC,
              arc_data->x, arc_data->y,
              arc_data->width, arc_data->height,
              arc_data->angle1, arc_data->angle2 );

  } else {
    /*
     * our first time through - erase the actual arc...
     */
    (*arc->erase)( arc );

    /*
     * ...store the current event location
     */
    x = event ? event->xbutton.x : 0;
    y = event ? event->xbutton.y : 0;
  }

  if( event ) {
    /*
     *  get the x,y delta
     */
    arc_data->x += (event->xbutton.x - x);
    arc_data->y += (event->xbutton.y - y);

    /*
     * draw a rubberband arc
     */
    XDrawArc( XtDisplay(GxDrawArea),
```

continues

Listing 23.8 Continued

```
              XtWindow(GxDrawArea), rubberGC,
              arc_data->x, arc_data->y,
              arc_data->width, arc_data->height,
              arc_data->angle1, arc_data->angle2 );

    x = event->xbutton.x;
    y = event->xbutton.y;
  } else {
    x = 0;
    y = 0;
  }
}
```

As you might recall, the `move` and `scale` actions are assigned and invoked from the `process_event` function introduced in Chapter 16, "Object Manipulation," in the section "Processing User Navigation of Objects," page 334.

The `arc_move` function follows the same form as the `line_move` introduced in Chapter 20, in the section "Moving a Line Object," page 395. The only difference to note is that by simply updating the x and y values of the `XArc` structure, the repositioning of the `Arc` object is accomplished:

```
33:     arc_data->x += (event->xbutton.x - x);
34:     arc_data->y += (event->xbutton.y - y);
```

The following section introduces the `arc_scale` method assigned to `Arc` objects.

Scaling an `Arc` Object

The `scale` method of an object, as introduced in Chapter 16, in the section "Processing User Navigation of Objects," page 334, is assigned as the `active` action when the user selects one of the active objects handles.

The `arc_scale` function introduced in Listing 23.9 manages the scale action for the Graphics Editor `Arc` object.

Listing 23.9 The `arc_scale` Function

```
static void arc_scale( GXObjPtr arc, XEvent *event )
{
  static XArc *tmp_data = NULL;

  if( tmp_data ) {
    /*
     * erase the rubberband arc
     */
    XDrawArc( XtDisplay(GxDrawArea),
             XtWindow(GxDrawArea), rubberGC,
              tmp_data->x, tmp_data->y,
              tmp_data->width, tmp_data->height,
              tmp_data->angle1, tmp_data->angle2 );

  } else {
    /*
     * our first time through - erase the actual arc
     */
    (*arc->erase)( arc );

    tmp_data = (XArc *)XtNew( XArc );
  }

  if( event ) {
    memcpy( (char *)tmp_data,
            (char *)arc->data, sizeof(XArc));
    apply_delta( tmp_data,
                 FixedX - event->xbutton.x,
                 FixedY - event->xbutton.y );

    /*
     * draw a rubberband arc
     */
    XDrawArc( XtDisplay(GxDrawArea),
              XtWindow(GxDrawArea), rubberGC,
              tmp_data->x, tmp_data->y,
              tmp_data->width, tmp_data->height,
              tmp_data->angle1, tmp_data->angle2 );

  } else {
    if( tmp_data ) {
       memcpy( (char *)arc->data, (char *)tmp_data,
              sizeof(XArc));

      XtFree( (char *)tmp_data );
      tmp_data = NULL;
    }
  }
}
```

➔The steps required to scale the `Arc` object follow those required to scale a `Line` object as discussed in Chapter 20, in the section "Scaling a Line Object," page 398.

Note

For a review of the calculations preformed by the `apply_delta` function as well as the declaration and assignment of the `FixedX` and `FixedY` variables, see Chapter 11, "Graphic Transformations," in the section "Scaling an Arc," page 243, and Listing 11.7 "The `apply_delta` Function for Arcs," page 245.

Copying an `Arc` Object

The `arc_copy` function, introduced in Listing 23.10, is simple to implement and convenient for users.

Listing 23.10 The `arc_copy` Function

```
1:  static void arc_copy( GXObjPtr arc )
2:  {
3:    XArc *temp_data;
4:    (*arc->deselect)(arc);
5:
6:    temp_data = (XArc *)XtNew(XArc);
7:    memcpy( (char *)temp_data, (char *)arc->data, sizeof(XArc));
8:
9:    temp_data->x += OFFSET;
10:   temp_data->y += OFFSET;
11:   create_arc( NULL, temp_data );
12:
13:   XtFree( (char *)temp_data );
14: }
```

The `arc_copy` function deselects the active object's handles

```
4:   (*arc->deselect)(arc);
```

creates memory to hold the `XArc` data

```
6:   temp_data = (XArc *)XtNew(XArc);
```

copies the data associated with the previously active object

```
7:   memcpy( (char *)temp_data, (char *)arc->data, sizeof(XArc));
```

increments the location of the copied data by an OFFSET

```
9:    temp_data->x += OFFSET;
10:   temp_data->y += OFFSET;
```

and employs the `create_arc` function to create a new object

```
11:   create_arc( NULL, temp_data );
```

followed by clearing the temporary `XArc` structure:

```
13:   XtFree( (char *)temp_data );
```

The last elements required for satisfying the creation of the `Arc` object is to add support for saving and restoring the Arc data.

Saving and Restoring an `Arc` Object

The capability to save and restore objects contained in the Graphics Editor, as introduced in Chapter 19, "Save and Restore," provides a mature level of capability to our application.

The `arc_save` function found in Listing 23.11 adheres to the save strategy introduced earlier.

Listing 23.11 The `arc_save` Function

```
1:  static void arc_save( FILE *fp, GXObjPtr arc )
2:  {
3:      XArc *arc_data = (XArc *)arc->data;
4:
5:      fprintf( fp, "ARC [x y width height angle1 angle2]\n");
6:
7:      fprintf( fp, "%d %d %d %d %d %d\n",
8:                      arc_data->x, arc_data->y,
9:                      arc_data->width, arc_data->height,
10:                     arc_data->angle1, arc_data->angle2 );
11: }
```

Extracting the `XArc` data from the object

```
3:     XArc *arc_data = (XArc *)arc->data;
```

writing the tagged line to indicate the format of the data to follow

```
5:     fprintf( fp, "ARC [x y width height angle1 angle2]\n");
```

and finally, writing the data

```
7:      fprintf( fp, "%d %d %d %d %d %d\n",
8:                      arc_data->x, arc_data->y,
9:                      arc_data->width, arc_data->height,
10:                     arc_data->angle1, arc_data->angle2 );
```

are the steps required to save the data associated with the `Arc` object.

The steps to restore the data to the editor are introduced in Listing 23.12.

Listing 23.12 The `gxArcLoad` Function

```
1:  void gxArcLoad( FILE *fp, GXObjPtr obj )
2:  {
3:      XArc arc;
4:
5:      fscanf( fp, "%hd %hd %hd %hd %hd %hd\n",
6:                      &arc.x, &arc.y, &arc.width,
7:                      &arc.height, &arc.angle1, &arc.angle2 );
8:
9:      create_arc( obj, &arc );
10: }
```

The data line is the only thing of import to the `gxArcLoad` function. Defining a local `XArc` structure and loading the saved data into the appropriate fields

```
fscanf( fp, "%hd %hd %hd %hd %hd %hd\n",
             &arc.x, &arc.y, &arc.width,
             &arc.height, &arc.angle1, &arc.angle2 );
```

and passing the address of the structure to the `create_arc` function

```
create_arc( obj, &arc );
```

retrieves the previously saved data and creates an object from it.

This completes the all aspects of introducing the `Arc` object to the Graphics Editor.

Next Steps

One object remains for introduction, integration, and investigation: namely, the `Text` object.

The `Text` object by far is the most complex of all the objects supported by the Graphics Editor. However, upon successful completion of the following chapter, you will have a functional editor capable of drawing, moving, scaling, saving, and restoring a variety of graphics objects.

In this chapter

Chapter 24

Vector `Text` Object

By choosing the Text creation icon shown in Figure 24.1, the user begins the creation process for the `Text` object.

Figure 24.1

The Text creation icon begins the `Text` *object creation process.*

The creation process for the `Text` object begins with prompting the user for a text string from which to create the Text object.

With the string successfully entered by the user, a value reflecting the interactive text object follows the movement of the cursor. The user places the text on the canvas window by pressing the mouse button. The resulting `ButtonPress` event location is used as the origin of the new `Text` object.

This chapter presents the internal methods and supporting functions defined for the `Text` object. These routines control all aspects of managing the different features of the object.

Note

The code listings introduced in this chapter are targeted for placement in the `gxText.c` source file. Additionally, functions presented in the listings that are not defined `static` should have a corresponding prototype placed in the `gxProtos.h` header file.

Creating a `Text` Object

The definition of the `gxDrawIcons` array introduced in Chapter 13

```
{ &text_icon, (void (*)(void))gx_text,
  "Draw dynamic text..."          },
```

assigned the Text icon the `gx_text` function for invocation when selected by the user. This function is defined in Listing 24.1, and it serves as the entry point to the interactive creation process of the `Text` object.

Listing 24.1 The `gx_text` Function

```
1:  void gx_text( XEvent *event )
2:  {
3:    static char *creation_text = NULL;
4:
5:    if( event == NULL ) {
6:      creation_text = NULL;
7:      place_creation_text( NULL, NULL );
8:
9:      /*
10:      * we have to prompt for a string
11:      */
12:     creation_text = get_creation_text();
13:
14:     if( creation_text != NULL )
15:       set_cursor( TEXT_MODE );
16:
17:   } else {
18:
19:     /*
20:      * If we have a string, place it!
21:      */
22:     if( creation_text ) {
23:       /* adjust for the hotspot in our cursor */
24:       event->xbutton.y -= 10;
25:       event->xbutton.x += 10;
26:
27:       place_creation_text( event, &creation_text );
28:     }
29:   }
30: }
```

At the specification of a `NULL` event reference, the `gx_text` function initializes local variables to a known state both in the `gx_text` function and the `place_creation_text` function:

```
5:    if( event == NULL ) {
6:      creation_text = NULL;
7:      place_creation_text( NULL, NULL );
```

Following this, the user is prompted for input with a call to the `get_creation_text` function:

```
12:     creation_text = get_creation_text();
```

If a string is successfully gained from the user, the cursor mode is updated to reflect the state of the application and

```
15:       set_cursor( TEXT_MODE );
```

the function finishes for the current iteration. The function is called again when the user moves the cursor into the canvas window and begins to generate `MotionNotify` events. Because the `create_text` variable is static, its value is available for the successive calls:

```
22:     if( creation_text ) {
```

With the presence of the `creation_text` string value, the event points are adjusted to account for the hotspot of the cursor

```
24:       event->xbutton.y -= 10;
25:       event->xbutton.x += 10;
```

and then used to locate the interactive text string for the placement:

```
27:       place_creation_text( event, &creation_text );
```

Listing 24.2 introduces the `place_creation_text` function used to indicate to the user the future location of the text object based on their navigation.

Listing 24.2 The `place_creation_text` Function

```
static void
place_creation_text( XEvent *event, char **_text )
{
  static GXTextPtr rubber_text = NULL;
  char *text = NULL;

  if( event == NULL ) {
    rubber_text = NULL;
  } else {
    if( _text ) text = *_text;

    if( rubber_text ) {
      GXDrawText( rubber_text, rubberGC );
    }

    switch( event->type ) {
      case ButtonPress:
        if( rubber_text && text ) {
          create_text( NULL, rubber_text );
```

continues

Listing 24.2 Continued

```
20:           gx_refresh();
21:
22:           freeGXText( rubber_text );
23:           rubber_text = NULL;
24:
25:           free( text );
26:           *_text = NULL;
27:
28:           set_cursor( NORMAL_MODE );
29:         }
30:         break;
31:
32:       case MotionNotify:
33:         if( text ) {
34:           if( rubber_text == NULL ) {
35:             rubber_text = update_gxtext( event,
36:                                          text, NULL );
37:           } else {
38:             (void)update_gxtext( event,
39:                                  text, rubber_text );
40:
41:           }
42:           GXDrawText( rubber_text, rubberGC );
43:         }
44:         break;
45:     }
46:   }
47: }
```

The `place_creation_text` function determines its actions based on the current event type.

Notice that the parameters required for the `place_creation_text` function include not only a reference to the event being processed but also a pointer to the character pointer containing the `creation_text`:

```
2: place_creation_text( XEvent *event, char **_text )
```

By passing the address to the creation text, its contents can be cleared when the user completes the placement process as indicated by a `ButtonPress` event.

Beyond validating the presence of valid event and copying the text variable to a local reference in lines 7–10, the `place_creation_text` function determines whether the `rubber_text` variable was previously initialized. A valid `rubber_text` value indicates that the function must erase the current interactive text string:

```
12:     if( rubber_text ) {
13:       GXDrawText( rubber_text, rubberGC );
14:     }
```

The function is now ready to determine the event type and execute the appropriate action body:

```
16:     switch( event->type ) {
```

Consider first the `MotionNotify` action body in lines 32–44.

The action for `MotionNotify` must invoke the `update_gxtext` to either establish the initial value of the `rubber_text` variable (lines 35–36) or to update the variable for subsequent notifications of mouse motion and draw the temporary `GXText` structure referenced by `rubber_text`.

The action appropriate for the `ButtonPress`

```
17:       case ButtonPress:
```

must ensure that a valid `rubber_text` value is assigned as well as current `text` string

```
18:         if( rubber_text && text ) {
```

and invokes the `create_text` routine to create the `Text` object:

```
19:           create_text( NULL, rubber_text );
```

After updating the canvas window with a call to `gx_refresh` on line 20, the function frees the memory associated with the `rubber_text` variable used during the interactive creation:

```
22:           freeGXText( rubber_text );
```

Several functions referenced during the course of this discussion have yet to be introduced. Review first the function to retrieve a string by prompting the user, introduced in Listing 24.3.

Listing 24.3 Prompting the User for a String

```
static void
close_dialog(Widget w, XtPointer cdata, XtPointer cbs)
{
  Widget dialog = (Widget)cdata;

  if( dialog ) XtUnmanageChild( dialog );
}

static char *get_creation_text( void )
{
  XtAppContext app;
  XEvent       event;

  Widget dialog;
  char *str = NULL;
```

continues

Listing 24.3 Continued

```

  dialog = XtVaCreateManagedWidget( "Text Entry Box",
                                    dialogWidgetClass,
                                    GxDrawArea,
                                    XtNwidth,  115,
                                    XtNheight, 70,
                                    XtNlabel,
                                       "Enter Text:",
                                    XtNvalue,  "",
                                    NULL );

  XawDialogAddButton( dialog, " Ok ",
                      close_dialog, dialog );

  app = XtWidgetToApplicationContext( GxDrawArea );

  while( XtIsManaged(dialog)) {
    XtAppNextEvent( app, &event );
    XtDispatchEvent( &event );
  }

  str = XawDialogGetValueString( dialog );
  XtDestroyWidget( dialog );

  /*
   * look for 'illegal' characters
   */
  {
    int c, indx = 0;
    char illegal_chars[] = { '\n', '\r' };

    while( (c = (int)str[indx]) != '\0' ) {
      if( strchr( illegal_chars, c ) != NULL )
          str[indx] = ' ';
      indx++;
    }

    /*
     * remove leading zeros
     */
    while( *str && *str == ' ' ) str++;
  }

  if( str && *str )
    return XtNewString(str);
  else
    return NULL;
}
```

The `get_creation_text` function is identical to the function used to retrieve a filename from the user for the `gx_save` and `gx_load` functions.

Review the discussion of the `GxGetFileName` function in Chapter 19, in the section "Common Object Save and Restore," page 367, for how the `get_creation_text` function works.

Listing 24.4 reviews the `GxDrawText` function originally introduced in Chapter 15, in the section "Text Object Data Structure," page 307.

Listing 24.4 The `GXDrawText` Function

```
1:  static void GXDrawText( GXTextPtr text, GC gc )
2:  {
3:    char *txt = text->text;
4:    int  c, chr, nsegs, num_pts;
5:
6:    for( c = 0; c < text->len; c++, txt++ ) {
7:      chr = *txt - ' ';
8:      nsegs = 0;
9:
10:     while( text->font[chr][nsegs] != NULL ) {
11:       num_pts = text->fontp[chr][nsegs];
12:
13:       if( num_pts > 0 ) {
14:         XDrawLines( XtDisplay(GxDrawArea),
15:                     XtWindow(GxDrawArea), gc,
16:                     text->vpts[c][nsegs], num_pts,
17:                     CoordModeOrigin );
18:       }
19:       nsegs++;
20:     }
21:   }
22: }
```

The `GxDrawText` function is responsible for traversing the characters comprising the `Text` object

```
6:    for( c = 0; c < text->len; c++, txt++ ) {
7:      chr = *txt - ' ';
```

then traversing the segments defining the character

```
10:     while( text->font[chr][nsegs] != NULL ) {
```

and for each set of points defining the segment:

```
11:       num_pts = text->fontp[chr][nsegs];
```

This function also invokes the XDrawLines Graphic Primitive to draw that portion of the character

```
13:      if( num_pts > 0 ) {
14:        XDrawLines( XtDisplay(GxDrawArea),
15:                    XtWindow(GxDrawArea), gc,
16:                    text->vpts[c][nsegs], num_pts,
17:                    CoordModeOrigin );
18:      }
```

before proceeding to the next segment:

```
19:      nsegs++;
```

The update_gxtext function introduced in Listing 24.5 is a complex function used to create a GXText structure from the character string and event location taken from the parameters passed to the function.

Listing 24.5 The update_gxtext Function

```
1:  static GXTextPtr
2:  update_gxtext( XEvent *xe, char *str, GXTextPtr upd )
3:  {
4:     GXTextPtr text = NULL;
5:
6:     if( upd ) {
7:       reset_pts( upd, xe->xbutton.x, xe->xbutton.y, 0, 0 );
8:     } else {
9:       text = create_gxtext( str, xe->xbutton.x,
10:                            xe->xbutton.y,
11:                            plain_simplex,
12:                            plain_simplex_p );
13:    }
14:
15:    return text;
16: }
```

The update_gxtext function branches in one of two ways. If a GXText structure is already referenced by upd, the reset_pts is invoked to update the structure for the new location specified by the event point:

```
7:      reset_pts( upd, xe->xbutton.x, xe->xbutton.y, 0, 0 );
```

However, if a GXText structure is not referenced currently by upd, the create_gxtext function is invoked to create one:

```
9:      text = create_gxtext( str, xe->xbutton.x,
10:                            xe->xbutton.y,
11:                            plain_simplex,
12:                            plain_simplex_p );
```

Notice the last two parameters passed to the create_gxtext function, plain_simplex and plain_simplex_p.

The plain_simplex array is a vector font set. More literally, it contains the definitions of all characters that can be represented by the font set. The plain_simplex_p is a control array containing the number of points contained in each segment of the corresponding character found in the plain_simplex array.

The definition of these arrays is shown in Listing 24.6; however, to begin understanding it, refer to the Excursion "An Introduction to Vector Fonts," in Chapter 15, in the section "Text Object Data Structure," page 307.

Listing 24.6 The vfonts/simplex.h Header File

```
#ifndef SIM_INC_H
#define SIM_INC_H

#include "vfonts/vector_chars.h"

XPoint **plain_simplex[] = {
 char699,char714,char717,char733,char719,char2271,
 char734,char731,char721,char722,char2219,char725,
 char711,char724,char710,char720,char700,char701,
 char702,char703,char704,char705,char706,char707,
 char708,char709,char712,char713,char2241,char726,
 char2242,char715,char2273,char501,char502,char503,
 char504,char505,char506,char507,char508,char509,
 char510,char511,char512,char513,char514,char515,
 char516,char517,char518,char519,char520,char521,
 char522,char523,char524,char525,char526,char2223,
 char804,char2224,char2262,char999,char730,char601,
 char602,char603,char604,char605,char606,char607,
 char608,char609,char610,char611,char612,char613,
 char614,char615,char616,char617,char618,char619,
 char620,char621,char622,char623,char624,char625,
 char626,char2225,char723,char2226,char2246,char718,
};

int *plain_simplex_p[] = {
 char_p699,char_p714,char_p717,char_p733,char_p719,
 char_p2271,char_p734,char_p731,char_p721,char_p722,
 char_p2219,char_p725,char_p711,char_p724,char_p710,
 char_p720,char_p700,char_p701,char_p702,char_p703,
 char_p704,char_p705,char_p706,char_p707,char_p708,
 char_p709,char_p712,char_p713,char_p2241,char_p726,
 char_p2242,char_p715,char_p2273,char_p501,char_p502,
 char_p503,char_p504,char_p505,char_p506,char_p507,
 char_p508,char_p509,char_p510,char_p511,char_p512,
 char_p513,char_p514,char_p515,char_p516,char_p517,
 char_p518,char_p519,char_p520,char_p521,char_p522,
 char_p523,char_p524,char_p525,char_p526,char_p2223,
 char_p804,char_p2224,char_p2262,char_p999,char_p730,
 char_p601,char_p602,char_p603,char_p604,char_p605,
 char_p606,char_p607,char_p608,char_p609,char_p610,
```

continues

Listing 24.6 Continued

```
   char_p611,char_p612,char_p613,char_p614,char_p615,
   char_p616,char_p617,char_p618,char_p619,char_p620,
   char_p621,char_p622,char_p623,char_p624,char_p625,
   char_p626,char_p2225,char_p723,char_p2226,char_p2246,
   char_p718,
 };

 #endif /* SIM_INC_H */
```

The `plain_simplex` array contains the character definitions for all characters that can be represented by this vector font set.

Note

You've probably noticed that the names of the characters in the `plain_simplex` array have no relationship to the actual characters defined.

The piece that makes these arrays meaningful is the `vector_chars.h` header file. This file defines several vector font sets available to the Graphics Editor project. For simplicity, however, only the `plain_simplex` font is employed.

The `vector_chars.h` file contains hundreds of character definitions. These characters, when constructed, were numbered sequentially. Later, as I cleaned up the definitions, removing those not referenced and grouping others into sets, it was necessary to repeat some characters representing simple punctuation and, in the end, the sets had a sort of mismatched appearance.

Appendix C, "Additional Vector Font Sets and `vector_chars.h`," presents all the vector fonts sets available to the Graphics Editor project as well as the contents of the `vector_chars.h` file. Because the file contains the definitions of all characters from all font sets (which consist of multiple segment definitions for each character), the file is literally thousands of lines long. Therefore, you probably do not want to type it in. In this case, I encourage to you reference the pertinent files on the CD-ROM accompanying this book.

Note

Chapter 28, "Extending the Graphics Editor," outlines ways to incorporate all available vector font sets into the Graphics Editor application concurrently.

A sample of the character for the exclamation point is presented in Listing 24.7.

Listing 24.7 Definition of Exclamation Point (`char714`)

```
static XPoint seg0_714[] = {
     (0,-12}, {0,2},
};

static XPoint seg1_714[] = {
  {0,7}, {-1,8}, {0,9}, {1,8}, {0,7},
};

static XPoint *char714[] = {
  seg0_714, seg1_714,
  NULL,
};

static int char_p714[] = {
  XtNumber(seg0_714), XtNumber(seg1_714),
};
```

A second example is provided in Listing 24.8, which contains the definition of the double quote character.

Listing 24.8 Definition of Double Quote (`char717`)

```
static XPoint seg0_717[] = {
  {-4,-12}, {-4,-5},
};

static XPoint seg1_717[] = {
  {4,-12}, {4,-5},
};

static XPoint *char717[] = {
  seg0_717, seg1_717,
  NULL,
};

static int char_p717[] = {
  XtNumber(seg0_717), XtNumber(seg1_717),
};
```

The relationship between a font set such as `plain_simplex` (definition of all characters) and a single character contained in the array (defined by a series of segments), and each of the segments comprising a character (arrays of `XPoints`), should be clear from these examples.

Further, the `plain_simplex_p` array serves the purpose of providing the number of points available for each segment defining a character.

With an understanding of the structure of a vector font set well in hand, turn your attention to the `create_gxtext` function introduced in Listing 24.9.

Listing 24.9 The `create_gxtext` Function

```
GXTextPtr create_gxtext( char *text,
                         int x, int y,
                         GXFont fnt, GXFontP fntp )
{
  GXTextPtr text_data;
  int c, chr, s, num_pts, nsegs;

  text_data = (void *)XtNew( GXText );

  text_data->x = x;
  text_data->y = y;

  text_data->dx = 0;
  text_data->dy = 0;

  text_data->text = XtNewString( text );
  text_data->len  = strlen( text );

  text_data->font  = fnt;
  text_data->fontp = fntp;

  text_data->vpts = (XPoint ***)
       XtMalloc(sizeof(XPoint **) * text_data->len);

  for( c = 0; c < text_data->len; c++, text++ ) {
    chr = *text - ' ';

    nsegs = 0;
    while( fnt[chr][nsegs] != NULL ) {
      nsegs++;
    }

    text_data->vpts[c] = (XPoint **)
      XtMalloc( sizeof( XPoint * ) * nsegs );

    for( s = 0; s < nsegs; s++ ) {
      num_pts = fntp[chr][s];

      text_data->vpts[c][s] = (XPoint *)
        XtMalloc( sizeof( XPoint ) * num_pts );
    }
  }

  reset_pts( text_data,
             text_data->x, text_data->y,
             text_data->dx, text_data->dy );

  return text_data;
}
```

The create_gxtext function begins by allocating a new instance of the GXText structure

```
text_data = (void *)XtNew( GXText );
```

followed by initializing the static fields of the GXText structure. The x and y fields track the origin of the Text object and are applied to all points of the font set as they are loaded into the GXText structure. The initial value for these fields will be the coordinates of the event point:

```
text_data->x = x;
text_data->y = y;
```

The dx and dy fields manage the scale factors for the x and y-axis applied to this Text object since creation. Their usefulness will be demonstrated shortly, but upon creation no scale deltas have been requested, so their initial values are 0:

```
text_data->dx = 0;
text_data->dy = 0;
```

A copy of the character string comprising the new object is stored in the GXText structure along with the string's width:

```
text_data->text = XtNewString( text );
text_data->len  = strlen( text );
```

Finally, to support future expansion of dynamically specifying the font set to apply for individual Text objects, the font particular to this object is assigned to the GXText structure. Also assigned is the control array which tracks the number of points for the segments of the various characters understood by the font set:

```
text_data->font  = fnt;
text_data->fontp = fntp;
```

The function then creates space to store the vector characters for the text string comprising the Text object

```
text_data->vpts = (XPoint ***)
     XtMalloc(sizeof(XPoint **) * text_data->len);
```

and proceeds to traverse the text string one character at a time:

```
for( c = 0; c < text_data->len; c++, text++ ) {
```

Because the first character understood by any font set is the space, all characters are subtracted from the decimal value of the space to ensure the index into the font set is correctly aligned for the current character:

```
  chr = *text - ' ';
```

Using this character, the number of segments needed to define it using the font set is determined with a `while` loop:

```
    nsegs = 0;
    while( fnt[chr][nsegs] != NULL ) {
      nsegs++;
    }
```

Knowing the required number of segments allows us to allocate sufficient space to store them:

```
    text_data->vpts[c] = (XPoint **)
      XtMalloc( sizeof( XPoint * ) * nsegs );
```

Finally, the control array indicating the number of points for each segment is employed to allocate sufficient space to store the points after they are transposed to be relative to the origin:

```
    for( s = 0; s < nsegs; s++ ) {
      num_pts = fntp[chr][s];

      text_data->vpts[c][s] = (XPoint *)
        XtMalloc( sizeof( XPoint ) * num_pts );
```

With the proper space reserved for the string defining the new `GXText` structure, the `reset_pts` function is used to transpose the points defining the segments of the characters in the text string:

```
  reset_pts( text_data,
             text_data->x, text_data->y,
             text_data->dx, text_data->dy );
```

Before presenting the `reset_pts` function, note the number of allocations performed for each character of the string assigned to the `GXText` structure.

The `GXText` structure is a memory-intensive construct because it requires memory allocation for the transformed points within the various segments. The actual number of allocations depends on the number of segments, which is influenced by the complexity of the font.

Therefore it is important that proper management be applied to this memory and that it be returned to the heap when no longer used by the application. For this reason, the `freeGXText` function found in Listing 24.10 has been defined.

Listing 24.10 Freeing the Memory Allocated by the `create_gxtext` Function

```
static void freeGXText( GXTextPtr text_data )
{
  int       c, chr, nsegs;
  char      *text = text_data->text;

  for( c = 0; c < text_data->len; c++, text++ ) {
    chr = *text - ' ';

    nsegs = 0;
    while( text_data->font[chr][nsegs] != NULL ) {

      XtFree( (char *)text_data->vpts[c][nsegs] );
      nsegs++;
    }
    XtFree( (char *)text_data->vpts[c] );
  }
  XtFree( (char *)text_data->vpts );
  XtFree( (char *)text_data->text );
}
```

The `freeGXText` function effectively works backward to perform frees where the `create_gxtext` performs allocations.

Look now at the `reset_pts` function used by the `create_gxtext` function and defined in Listing 24.11.

Listing 24.11 Resetting Vector Points to Maintain Text Quality

```
static void reset_pts( GXTextPtr text,
                       int x, int y,
                       int dx, int dy )
{
  reset_font_pts( text, x, y );
  apply_scale( text, dx, dy );
}
```

The `reset_pts` function is responsible for first applying the `x`, `y` origin specified in the parameter list to the points defined in the font set and then applying the `dx` and `dy` scale factors to the transformed points.

The function `reset_font_pts` shown in Listing 24.12 accomplishes the task of transforming the points in the font set to the specified origin.

Listing 24.12 The `reset_font_pts` Function

```
static void reset_font_pts( GXTextPtr text,
                            int x, int y )
{
  char *txt = text->text;
  int i, c,                /* loop counters        */
      chr,                 /* index to chr in font */
      nsegs,               /* num segs for chr     */
      num_pts,             /* num pts for seg      */
      maxx,                /* max extent for a chr */
      orig_x = x,          /* where chr begins     */
      c_off  = 0;          /* offset from prev chr */
```

continues

Listing 24.12 Continued

```
12:
13:   text->x = x;
14:   text->y = y;
15:
16:   maxx = -9999;
17:   for( c = 0; c < text->len; c++, txt++ ) {
18:     chr = *txt - ' ';
19:
20:     nsegs = 0;
21:     while( text->font[chr][nsegs] != NULL ) {
22:       num_pts = text->fontp[chr][nsegs];
23:
24:       for( i = 0; i < num_pts; i++ ) {
25:         /*
26:          * reset vpts from font relative origin of this
27:          * character applying screen scale
28:          */
29:         text->vpts[c][nsegs][i].x =
30:           (text->font[chr][nsegs][i].x + c_off) +
31:                                                   orig_x;
32:         text->vpts[c][nsegs][i].y =
33:           text->font[chr][nsegs][i].y + y;
34:
35:         maxx = max( maxx, text->vpts[c][nsegs][i].x );
36:       }
37:       nsegs++;
38:     }
39:
40:     /*
41:      * look ahead at width of next char setting orig_x
42:      * to ensure no overlap with the current character
43:      */
44:     if( (c + 1) < text->len )
45:       c_off = next_char_min( text, *(txt + 1) );
46:
47:     orig_x  = SPC(maxx);
48:   }
49: }
```

Following the declaration of the variables that will control the multiple levels of looping necessary to traverse the characters, segments, and points defining the object, the function resets the `x`, `y` origin fields maintained in the `GXText` structure:

```
13:   text->x = x;
14:   text->y = y;
```

The mechanism for using each character of the text string contained in the `GXText` structure to traverse the segments defined by the font set is consistent throughout the support functions used by the `Text` object. Therefore focus here is on the purpose of the function.

In other words, lines 16–22 are identical to those used previously to access the points which comprise the current character's definition by the font set.

Unique functionality begins when the `vpts` structure created at the beginning of the `create_gxtext` function is assigned the transformed points taken from the font set.

```
29:         text->vpts[c][nsegs][i].x =
30:           (text->font[chr][nsegs][i].x + c_off) +
31:                                               orig_x;
32:         text->vpts[c][nsegs][i].y =
33:           text->font[chr][nsegs][i].y + y;
```

The transformation applied to the `x` points contained in the font set

```
text->font[chr][nsegs][i].x
```

consists of applying a character offset `c_off` and the value of the `x` component of the origin to determine the value of the corresponding element in the `vpts` array,

```
29:         text->vpts[c][nsegs][i].x =
```

which will later be used to actually draw the object.

The transformation for the `y` component is a little more direct:

```
32:         text->vpts[c][nsegs][i].y =
33:           text->font[chr][nsegs][i].y + y;
```

The next step taken by the `reset_font_pts` function is to track the furthest `x` point for this character:

```
35:         maxx = max( maxx, text->vpts[c][nsegs][i].x );
```

The `maxx` value will be important for ensuring that subsequent characters are properly spaced to the right of the current one.

When all the points for all the segments for the current character are transformed and placed in the appropriate `vpts` element, the `while` loop ends

```
38:     }
```

and the offset for the next character is calculated:

```
44:     if( (c + 1) < text->len )
45:       c_off = next_char_min( text, *(txt + 1) );
```

Finally, the origin of the `Text` object is advanced by the SPC macro

```
47:     orig_x  = SPC(maxx);
```

and the `for` loop advances to the next character for the text string of this object. The transformation begins for the next character.

The macro definition should be placed at the beginning of the `gxText.c` file:

```
#define SPC(w) (w + 3) /* just a wee gap between chars */
```

The function `next_char_min` used to calculate the offset of the character to follow is introduced in Listing 24.13.

Listing 24.13 The `next_char_min` Function

```
1:  static int next_char_min( GXTextPtr text, char c )
2:  {
3:    int i, minx = 9999, num_pts, nsegs = 0, coff = 0;
4:
5:    int _c = c - ' ';
6:
7:    nsegs = 0;
8:
9:    while( text->font[(int)_c][nsegs] != NULL ) {
10:     num_pts = text->fontp[(int)_c][nsegs];
11:
12:     for( i = 0; i < num_pts; i++ ) {
13:       minx = min(text->font[(int)_c][nsegs][i].x, minx);
14:     }
15:     nsegs++;
16:   }
17:
18:   /* approx fixed with of a character */
19:   /* will be used to represent space  */
20:   if( c == ' ' )
21:     coff = 12;
22:   else if( minx < 0 )
23:     coff = abs(minx);
24:
25:   return coff;
26: }
```

The `next_char_min` function looks at the points defining a character within the font set and determines the minimum `x` point for the character cell:

```
13:       minx = min(text->font[(int)_c][nsegs][i].x, minx);
```

Of course, the points for all segments defining the character are considered

```
15:     nsegs++;
```

to determine the amount to advance the transformation of points placed into the `vpts` array to avoid character overlap.

Two special cases are considered by the `next_char_min` function. First is the occurrence of a space character

```
20:   if( c == ' ' )
```

because a space should be the same width despite the character following. And, second, the formation of the character definitions centered on origin resulting in a negative minimum value:

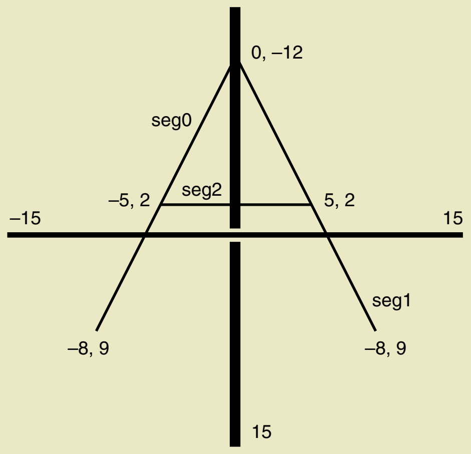
```
22:   else if( minx < 0 )
```

Note

Figure 24.2 is a graphical depiction of character definitions being centered on the origin and clearly demonstrates the need to account for a negative minimum value.

Figure 24.2

The coordinate relationships for the points defining each character of a vector font set.



The second function employed by the `reset_pts` function in Listing 24.11 is the `apply_scale` function

```
apply_scale( text, dx, dy );
```

which can be found in Listing 24.14.

Listing 24.14 The `apply_scale` Function

```
static void apply_scale( GXTextPtr text,
                         int dx, int dy )
{
  char *txt = text->text;

  int x, y;                  /* for transformation */
  int i, c, chr, nsegs, num_pts; /* for traversing */
  int minx, miny, maxx, maxy, width, height;

  get_extents( text, &minx, &miny, &maxx, &maxy );

  width  = maxx - minx;
  height = maxy - miny;

  x = text->x - minx;
  y = text->y - miny;
```

continues

Listing 24.14 Continued

```
17:
18:   for( c = 0; c < text->len; c++, txt++ ) {
19:     chr = *txt - ' ';
20:
21:     nsegs = 0;
22:     while( text->font[chr][nsegs] != NULL ) {
23:       num_pts = text->fontp[chr][nsegs];
24:
25:       for( i = 0; i < num_pts; i++ ) {
26:         text->vpts[c][nsegs][i].x += (int)(dx *
27:             ((float)(text->vpts[c][nsegs][i].x -
28:                            minx)/(float)width)) + x;
29:
30:         text->vpts[c][nsegs][i].y += (int)(dy *
31:             ((float)(text->vpts[c][nsegs][i].y -
32:                           miny)/(float)height)) + y;
33:       }
34:       nsegs++;
35:     }
36:   }
37: }
```

The `apply_scale` function is critical for ensuring that the integrity of the `Text` object is maintained during the many calculations preformed continually on the points contained in the `vpts` array in the `GXText` structure.

The role it serves is to enable the user to scale the `Text` object and, unlike other objects in the editor, have the `Text` object revert periodically to the original points transformed from the character definitions in the font set (accomplished by the `reset_font_pts` function). After being reset to the original font points (transformed), the retention of the `dx` and `dy` values (scale deltas for the x and y-axis) is reapplied without the user even knowing.

The integrity (quality) of the points defining the `Text` object displayed on the canvas is critical because many of the line segments within a character definition for the more complex or fancier vector font sets are quite small. An error as slight as one pixel will have a detrimental effect on the appearance of the vector text.

The `apply_scale` function traverses the characters to access the segment and point arrays defined by the font set, much the same way as other functions introduced already. Important to the purpose of the `apply_scale` function, however, is the application of the `dx` and `dy` values *proportionally* to the `x` and `y` values stored in the `vpts` array:

```
26:         text->vpts[c][nsegs][i].x += (int)(dx *
27:             ((float)(text->vpts[c][nsegs][i].x -
28:                            minx)/(float)width)) + x;
```

```

        text->vpts[c][nsegs][i].y += (int)(dy *
            ((float)(text->vpts[c][nsegs][i].y -
                          miny)/(float)height)) + y;
```

The term *proportionally* refers to the fact that the application of the scale factor is weighted based on the proximity of the point to the extent of the object.

For this reason, the extents of the object are determined early in the function

```
10:   get_extents( text, &minx, &miny, &maxx, &maxy );
```

and from the extents the `width` and `height` values can be determined.

The `get_extents` function is found in Listing 24.15.

Listing 24.15 The `get_extents` Function

```
static void get_extents( GXTextPtr text,
                         int *minx, int *miny,
                         int *maxx, int *maxy )
{
  char *txt = text->text;
  int  c, chr, nsegs, x1, y1, x2, y2;

  *minx = *miny = 9999;
  *maxx = *maxy = -9999;

  for( c = 0; c < text->len; c ++, txt ++ ) {
    chr = *txt - ' ';

    nsegs = 0;
    while( text->font[chr][nsegs] != NULL ) {

      get_bounds( text->vpts[c][nsegs],
                  text->fontp[chr][nsegs],
                  &x1, &y1, &x2, &y2 );

      *minx = min( x1, *minx );
      *miny = min( y1, *miny );
      *maxx = max( x2, *maxx );
      *maxy = max( y2, *maxy );

      nsegs++;
    }
  }
}
```

The `get_extents` function is, perhaps, the most straightforward of any function introduced so far.

It, too, traverses the segments defined by the font set to represent each character of the `GXText` text field to obtain the bounds of the points contained in the segment:

```
get_bounds( text->vpts[c][nsegs],
            text->fontp[chr][nsegs],
            &x1, &y1, &x2, &y2 );
```

The bounds of each segment are used to keep track of the minimum and maximum points used to represent the object:

```
*minx = min( x1, *minx );
*miny = min( y1, *miny );
*maxx = max( x2, *maxx );
*maxy = max( y2, *maxy );
```

These minimums and maximums reflect the extents of the object.

The `get_bounds` function used by the `get_extents` function is located in Listing 24.16.

Listing 24.16 The `get_bounds` Function

```
void get_bounds( XPoint *pts, int num_pts,
                 int *x1, int *y1, int *x2, int *y2 )
{
  int i;

  *x1 = *y1 = SHRT_MAX;
  *x2 = *y2 = 0;

  for( i = 0; i < num_pts; i++ ) {
    *x1 = min( pts[i].x, *x1 );
    *y1 = min( pts[i].y, *y1 );

    *x2 = max( pts[i].x, *x2 );
    *y2 = max( pts[i].y, *y2 );
  }
}
```

The `get_bounds` function acts on a single array of `XPoints` and thus is suitable for use by the `Line` object as well.

For an array of points, the minimum and maximum points contained in the array are returned to the calling function:

```
 for( i = 0; i < num_pts; i++ ) {
    *x1 = min( pts[i].x, *x1 );
    *y1 = min( pts[i].y, *y1 );
    *x2 = max( pts[i].x, *x2 );
    *y2 = max( pts[i].y, *y2 );
  }
```

We started our winding way to this function by waiting for the user to press the mouse button and position the `creation_text`, shown in Listing 24.2. When she finally does, the temporary `GXText` structure referenced by `rubber_text` is used to create a `Text` object:

```
        create_text( NULL, rubber_text );
```

The `create_text` function is defined in Listing 27.17.

Listing 24.17 The `create_text` Function

```
static void create_text( GXObjPtr _obj, GXTextPtr text )
{
  /*
   * create the template object
   */
  GXObjPtr obj = _obj;

  if( obj == NULL ) {
      obj = gx_create_obj();
  }

  obj->data = copy_gxtext( text, 0, 0 );

  obj->draw     = text_draw;
  obj->erase    = text_erase;
  obj->find     = text_find;
  obj->move     = text_move;
  obj->scale    = text_scale;


  obj->copy     = text_copy;
  obj->select   = text_select;
  obj->deselect = text_deselect;

  obj->save = text_save;

  gx_add_obj( obj );
}
```

The parameter list of the `create_text` function

```
static void create_text( GXObjPtr _obj, GXTextPtr text )
```

accounts for an invocation of the procedure with the caller having already created the common object to contain the text data. This supports the restoring of the object from a saved file, as is seen later in the chapter.

During interactive creation, however, the function `create_text` is called with a `NULL` value as the first parameter forcing the function to create the common object portion through a call to `gx_create_obj`.

➔This is introduced in Chapter 17, in the section “Common Object Creation,” page 343:

```
9:       obj = gx_create_obj();
```

The creation routine then copies the `GXText` structure defining the object

```
12:   obj->data = copy_gxtext( text, 0, 0 );
```

following which the object methods and manipulation functions for controlling the object-specific data structure in lines 14–25 are assigned

```
14:   obj->draw     = text_draw;
15:   obj->erase    = text_erase;
16:   obj->find     = text_find;
17:   obj->move     = text_move;
18:   obj->scale    = text_scale;
19:
20:
21:   obj->copy     = text_copy;
22:   obj->select   = text_select;
23:   obj->deselect = text_deselect;
24:
25:   obj->save = text_save;
```

after which the object is added to the list of objects managed by the editor application:

```
27:   gx_add_obj( obj );
```

Listing 24.18 shows the contents of the `copy_gxtext` function used by the `create_text` function.

Listing 24.18 The `copy_gxtext` Function

```
static GXTextPtr copy_gxtext(GXTextPtr text, int off_x, int off_y)
{
  GXTextPtr text_data;
  int       i, c, chr, num_pts, nsegs;
  char      *txt;

  text_data =
    create_gxtext( text->text, text->x + off_x, text->y + off_y,
                   text->font, text->fontp );

  text_data->dx = text->dx;
  text_data->dy = text->dy;

  txt = text_data->text;
  for( c = 0; c < text_data->len; c++, txt++ ) {
    chr = *txt - ' ';

    nsegs = 0;
    while( text_data->font[chr][nsegs] != NULL ) {
```

```
20:
21:       num_pts = text_data->fontp[chr][nsegs];
22:
23:       for( i = 0; i < num_pts; i++ ) {
24:
25:         text_data->vpts[c][nsegs][i].x =
26:           text->vpts[c][nsegs][i].x + off_x;
27:         text_data->vpts[c][nsegs][i].y =
28:           text->vpts[c][nsegs][i].y + off_y;
29:       }
30:       nsegs++;
31:     }
32:   }
33:
34:   return text_data;
35: }
```

The `copy_gxtext` function begins by creating a new `GXText` structure with the origin of the Text structure incremented by the offset specified for the `x` and `y` components:

```
7:    text_data =
8:      create_gxtext( text->text, text->x + off_x, text->y + off_y,
9:                     text->font, text->fontp );
```

The function then traverses all points contained in the `vpts` array and increments them accordingly as well

```
25:         text_data->vpts[c][nsegs][i].x =
26:           text->vpts[c][nsegs][i].x + off_x;
27:         text_data->vpts[c][nsegs][i].y =
28:           text->vpts[c][nsegs][i].y + off_y;
```

finally returning the copy of the `GXText` structure with the offset values:

```
34:   return text_data;
```

The `create_text` function specifies `0` for the offset value of the copy created:

```
12:   obj->data = copy_gxtext( text, 0, 0 );
```

However, this same function will be used in the definition of the `text_copy` method.

A created object is no longer drawn using the interactive rubber-banding `GC`, but by invoking the `draw` method of the object.

The following section introduces the methods that draw and erase the `Text` object from the drawing area canvas.

Drawing and Erasing a `Text` Object

The act of drawing or erasing an object in the Graphics Editor differs only in the treatment of the `tile` field of the `GC` created for the request.

Notice in Listing 24.19, lines 17 and 25, that the `draw` and `erase` methods differ only in the value passed for the `tile` flag required as the second parameter to the `draw_erase` function. (Note as well that the `erase` method is responsible for removing the object's handles if present on the screen).

Listing 24.19 Drawing and Erasing a `Text` Object

```
1:  static void draw_erase( GXObjPtr text, Boolean tile )
2:  {
3:    GC gc;
4:
5:    gc = gx_allocate_gc( text, tile );
6:    GXDrawText( (GXTextPtr)text->data, gc );
7:
8:    XtReleaseGC( GxDrawArea, gc );
9:  }
10:
12: /*
13:  * text_draw
14:  */
15: static void text_draw( GXObjPtr obj )
16: {
17:   draw_erase( obj, False );
18: }
19:
20: /*
21:  * text_erase
22:  */
23: static void text_erase( GXObjPtr obj )
24: {
25:   draw_erase( obj, True );
26: }
```

The `draw_erase` function begins by creating a Graphics Context for use in the `GXDrawText` function:

```
5:   gc = gx_allocate_gc( text, tile );
```

➔ Important to the creation of the Graphics Context is the specification of the tile flag to the `gx_allocate_gc` function introduced in Chapter 17, in the section "Creating a Graphics Context," page 347.

Note

If you recall, this flag indicated whether the background `Pixmap` of the `GxDrawArea` was assigned as the value to the `tile` field of the `GC` created.

A `tile` value assigned to a Graphic Context causes an effective erase action to occur because the pixels from the background `Pixmap` are placed where otherwise the foreground value of the `GC` would be placed, making the underlying background the result of the `XDrawLines` (or any X Graphic Primitive) request.

An appropriately created GC created for the current draw or erase action, the draw_erase function can request that the object be updated in the canvas window:

```
6:   GXDrawText( (GXTextPtr)text->data, gc );
```

Because the X Server will attempt to cache the GC for future requests, it is important to specify our complete use of it with a call to XtReleaseGC:

```
8:    XtReleaseGC( GxDrawArea, gc );
```

With the object visible on the screen, it is now eligible for manipulation by the user. However, before it can be moved, scaled, or deleted, it must be selected by the user.

The next section introduces the Text object find method used to determine whether an event has successfully located the object on the drawing area.

Finding a Text Object

The text_find method shown in Listing 24.20 borrows greatly from the support functions created for the line_find method in Chapter 20, in the section "Finding a Line Object," page 390.

Listing 24.20 The text_find Function

```
static Boolean text_find( GXObjPtr obj, XEvent *event )
{
  GXTextPtr text = obj->data;

  char *txt = text->text;

  int  i, c, chr, nsegs, num_pts, found = False;

  for( c = 0; (c < text->len) && !found; c++, txt++ ) {
    chr = *txt - ' ';

    nsegs = 0;
    while((text->font[chr][nsegs] != NULL) && !found) {
      num_pts = text->fontp[chr][nsegs];

      for(i = 0; (i < (num_pts - 1)) && !found; i++) {
        found =
            near_segment( text->vpts[c][nsegs][i  ].x,
                          text->vpts[c][nsegs][i  ].y,
                          text->vpts[c][nsegs][i+1].x,
                          text->vpts[c][nsegs][i+1].y,
                          event->xbutton.x,
                          event->xbutton.y );
      }
      nsegs++;
```

continues

Listing 24.20 Continued

```
      }
    }

    return found;
  }
```

The `text_find` method parses the `GXText` structure to reduce it to arrays of points compatible with the `near_segment` function found in Chapter 10, in the section "Calculating Point and Line Intersections," page 211.

If the event point specified to the `text_find` method intersects any of the many segments composing the `Text` object, the loops are interrupted and the `True` value is returned to the calling function.

The `text_find` method serves two purposes in the management of the `Text` object. One purpose is to enable the user to select the object for manipulation. A second purpose is to determine whether the manipulation requested by the user is the `move` action.

The following section introduces the step stemming from the `text_find` method resulting in the selecting or deselecting of the object.

Selecting and Deselecting a `Text` Object

If the event passed to the `text_find` method discussed in the previous section results in the location of an object that is not currently selected, the object is made active by invoking its `select` method.

The `select` method for the `Text` object, as seen in Listing 24.21, ensures that handles are created and drawn for the object.

Listing 24.21 The `text_select` Function

```
static void text_select( GXObjPtr text )
{
  text_bounding_handles( text );
  gx_draw_handles( text );
}
```

➔ The function `gx_draw_handles` was introduced in Chapter 16, section "Managing Object Handles," page 327.

The process of creating the handles for the `Text` object begins with a call to `text_bounding_handles` found in Listing 24.22.

Listing 24.22 Creating Handles for a Text Object

```
static void text_bounding_handles( GXObjPtr gx_text )
{
  GXTextPtr text = gx_text->data;

  int i, minx, miny, maxx, maxy, width, height;

  gx_text->handles = (XRectangle *)
                XtMalloc( sizeof(XRectangle) * 8 );
  gx_text->num_handles = 8;

  if( gx_text->handles == NULL ) {
    perror( "Alloc failed for text handles" );
    gx_text->num_handles = 0;
    return;
  }

  for( i = 0; i < 8; i++ ) {
    gx_text->handles[i].width  = HNDL_SIZE;
    gx_text->handles[i].height = HNDL_SIZE;
  }

  get_extents(text, &minx, &miny, &maxx, &maxy);
  width = maxx - minx; height = maxy - miny;

  gx_text->handles[0].x =
        minx - HNDL_OFFSET - HNDL_SIZE;
  gx_text->handles[0].y =
        miny - HNDL_OFFSET - HNDL_SIZE;

  gx_text->handles[1].x =
        minx + (width/2) - HNDL_OFFSET;
  gx_text->handles[1].y =
        miny - HNDL_SIZE - HNDL_OFFSET;

  gx_text->handles[2].x =
        maxx + HNDL_OFFSET;

  gx_text->handles[2].y =
        miny - HNDL_SIZE - HNDL_OFFSET;

  gx_text->handles[3].x = maxx + HNDL_OFFSET;
  gx_text->handles[3].y =
        miny + (height/2) - HNDL_OFFSET;

  gx_text->handles[4].x = maxx + HNDL_OFFSET;
  gx_text->handles[4].y = maxy + HNDL_OFFSET;

  gx_text->handles[5].x =
        minx + (width/2) - HNDL_OFFSET;
```

continues

Listing 24.22 Continued

```
50:   gx_text->handles[5].y = maxy + HNDL_OFFSET;
51:
52:   gx_text->handles[6].x =
53:         minx - HNDL_OFFSET - HNDL_SIZE;
54:   gx_text->handles[6].y = maxy + HNDL_OFFSET;
55:
56:   gx_text->handles[7].x =
57:         minx - HNDL_OFFSET - HNDL_SIZE;
58:   gx_text->handles[7].y =
59:         miny + (height/2) - HNDL_OFFSET;
60: }
```

The creation of the `Text` object handles in `text_bounding_handles` begins by ensuring that the correct number of elements are created in the `handles` array and the correct handle count is assigned the `num_handles` field of the common object structure:

```
7:    gx_text->handles = (XRectangle *)
8:                   XtMalloc( sizeof(XRectangle) * 8 );
9:    gx_text->num_handles = 8;
```

After testing for a failure of the allocation routine, the widths of the handles for the `Text` object are assigned

```
17:   for( i = 0; i < 8; i++ ) {
18:     gx_text->handles[i].width  = HNDL_SIZE;
19:     gx_text->handles[i].height = HNDL_SIZE;
20:   }
```

and then the extents of the `Text` object are obtained in order to determine the placement of the handles:

```
22:   get_extents(text, &minx, &miny, &maxx, &maxy);
23:   width = maxx - minx; height = maxy - miny;
```

Last comes the tedious task of placing each of the eight handles at the appropriate location around the object in lines 25–59.

Note

The position of the handles runs clockwise, starting with `handles[0]` located in the upper-right corner of the object's bounds.

The `Text` object's `deselect` method introduced in Listing 24.23 shows the steps necessary to remove the handles, indicating the active state of the `Text` object.

Listing 24.23 Deselecting a `Text` Object

```
1:  static void text_deselect( GXObjPtr text )
2:  {
3:    if( text->handles && text->num_handles > 0 ) {
4:      gx_erase_handles( text );
5:
6:      XtFree((char *)text->handles );
7:
8:      text->handles = NULL;
9:      text->num_handles = 0;
10:   }
11: }
```

The function is simply responsible for erasing the handles

```
4:     gx_erase_handles( text );
```

freeing the memory associated with them

```
6:     XtFree((char *)text->handles );
```

and resetting the values of the variables:

```
8:      text->handles = NULL;
9:      text->num_handles = 0;
```

The next section provides the functionality of interactively moving a `Text` object.

Moving a `Text` Object

The steps required to relocate a `Text` object on the canvas are provided by the `Text_move` method seen in Listing 24.24.

Listing 24.24 The `text_move` Function

```
1:  static void text_move( GXObjPtr text, XEvent *event )
2:  {
3:    static int x = 0, y = 0;
4:
5:    if( x && y ) {
6:      GXDrawText( (GXTextPtr)text->data, rubberGC );
7:    } else {
8:      (*text->erase)( text );
9:
10:     x = event ? event->xbutton.x : 0;
11:     y = event ? event->xbutton.y : 0;
12:   }
13:
14:   if( event ) {
15:     apply_delta( text->data,
16:                  event->xbutton.x - x,
17:                  event->xbutton.y - y );
```

continues

Listing 24.24 Continued

```
18:
19:     GXDrawText( text->data, rubberGC );
20:     x = event->xbutton.x;
21:     y = event->xbutton.y;
22:  } else {
23:     x = 0;
24:     y = 0;
25:   }
26: }
```

The `text_move` function uses the `static` points `x` and `y` to track consecutive calls to the function for a single move operation:

```
5:   if( x && y ) {
```

If `x` and `y` are non-zero, the `text_move` function knows to erase a previously draw rubber-banding copy of the `Text` object:

```
6:     GXDrawText( (GXTextPtr)text->data, rubberGC );
```

Otherwise, the actual `Text` object is erased from the canvas

```
8:     (*text->erase)( text );
```

and the values of `x` and `y` are assigned:

```
10:     x = event ? event->xbutton.x : 0;
11:     y = event ? event->xbutton.y : 0;
```

In the presence of a valid event

```
14:   if( event ) {
```

the `apply_delta` function is used to reposition the interactive `Text` object

```
15:     apply_delta( text->data,
16:                  event->xbutton.x - x,
17:                  event->xbutton.y - y );
```

the `GXText` structure with the updated points is drawn using the rubber-banding `GC`

```
19:     GXDrawText( text->data, rubberGC );
```

and the current event point is retained in the `static` variables:

```
20:     x = event->xbutton.x;
21:     y = event->xbutton.y;
```

If a `NULL` event reference is passed to the function, the `x` and `y` values are reset to `0` and the current `move` action is cancelled:

```
23:     x = 0;
24:     y = 0;
```

Listing 24.25 shows the definition of the apply_delta function used by the text_move method.

Listing 24.25 The apply_delta Function

```
static void apply_delta( GXTextPtr text,
                         int dx, int dy )
{
  char *txt = text->text;
  int  i, c, chr, nsegs, num_pts;

  text->x += dx;
  text->y += dy;

  for( c = 0; c < text->len; c++, txt++ ) {
    chr = *txt - ' ';

    nsegs = 0;
    while( text->font[chr][nsegs] != NULL ) {
      num_pts = text->fontp[chr][nsegs];

      for( i = 0; i < num_pts; i++ ) {

        text->vpts[c][nsegs][i].x += dx;
        text->vpts[c][nsegs][i].y += dy;
      }
      nsegs++;
    }
  }
}
```

The apply_delta function is a pleasant reprieve from the complex support function introduced in this chapter. By now, you should be comfortable with traversing the GXText structure using the characters of the text field to index into the font set to find the segments and access the points.

The apply_delta simply makes this traversal to access the points composing each character and increments them by the delta values specified for the x and y components accomplishing a move for the Text object.

The following section addresses the steps required to scale the Text object.

Scaling a Text Object

The scale method of an object, as introduced in Chapter 16, in the section "Processing User Navigation of Objects," page 334, is assigned as the active action when the user selects one of the active object's handles.

The `text_scale` method introduced in Listing 24.26 manages the scale action for the Graphics Editor `Text` object.

Listing 24.26 The `text_scale` Function

```
1:  static void text_scale( GXObjPtr text, XEvent *event )
2:  {
3:    static int x = 0, y = 0;
4:
5:    if( x && y ) {
6:      GXDrawText( text->data, rubberGC );
7:
8:    } else {
9:      (*text->erase)( text );
10:
11:     x = event ? event->xbutton.x : 0;
12:     y = event ? event->xbutton.y : 0;
13:   }
14:
15:   if( event ) {
16:     calc_apply_scale( text->data,
17:          event->xbutton.x - x, event->xbutton.y - y );
18:     x = event->xbutton.x;
19:     y = event->xbutton.y;
20:
21:     GXDrawText( text->data, rubberGC );
22:   } else {
23:     x = 0;
24:     y = 0;
25:   }
26: }
```

The `text_scale` method of the `Text` object is responsible for managing the event references passed to the function. Identical in structure to scale methods for previously introduced objects, focus on the portion of this function that is unique to the `Text` object:

```
16:     calc_apply_scale( text->data,
17:          event->xbutton.x - x, event->xbutton.y - y );
18:     x = event->xbutton.x;
19:     y = event->xbutton.y;
```

Listing 24.27 shows the `calc_apply_scale` function used by the `text_scale` method to accomplish the scale request.

Listing 24.27 The `calc_apply_scale` Function

```
1:  static void calc_apply_scale( GXTextPtr text,
2:                                int dx, int dy )
3:  {
4:    switch( GxActiveHandle ) {
```

```
5:    case 0:
6:      apply_scale_top ( text, dx, dy );
7:      apply_scale_left( text, dx, dy );
8:      break;
9:
10:   case 1:
11:     apply_scale_top( text, dx, dy );
12:     break;
13:
14:   case 2:
15:     apply_scale_top  ( text, dx, dy );
16:     apply_scale_right( text, dx, dy );
17:     break;
18:
19:   case 3:
20:     apply_scale_right( text, dx, dy );
21:     break;
22:
23:   case 4:
24:     apply_scale_right ( text, dx, dy );
25:     apply_scale_bottom( text, dx, dy );
26:     break;
27:
28:   case 5:
29:     apply_scale_bottom( text, dx, dy );
30:     break;
31:
32:   case 6:
33:     apply_scale_bottom( text, dx, dy );
34:     apply_scale_left  ( text, dx, dy );
35:     break;
36:
37:   case 7:
38:     apply_scale_left( text, dx, dy );
39:     break;
40:
41:   default:
42:     setStatus( "TEXT: The end is nigh!" );
43:   }
44:
45:   /* resetting to the original points ensures we don't */
46:   /* compound rounding errors for points that have     */
47:   /* already been scaled                               */
48:   reset_pts( text, text->x, text->y, text->dx, text->dy );
49: }
```

The function switches on the `GxActiveHandle` value set at the start of the scale action

```
4:   switch( GxActiveHandle ) {
```

to determine which direction the scale action should be applied. Then one of the several `apply_scale_<direction>` functions found in Listing 24.28 is invoked to adjust the `x`, `y`, `dx`, or `dy` fields of the `GXText` structure appropriate for the direction the object is being scaled.

Listing 24.28 The Support Functions Scaling a `Text` Object

```
static void apply_scale_top( GXTextPtr text,
                             int dx, int dy )
{
  text->y  += dy;
  text->dy += (dy * -1);
}
static void apply_scale_left( GXTextPtr text,
                              int dx, int dy )
{
  text->x  += dx;
  text->dx += (dx * -1);
}
static void apply_scale_right( GXTextPtr text,
                               int dx, int dy )
{
  text->dx += dx;
}
static void apply_scale_bottom( GXTextPtr text,
                                int dx, int dy )
{
  text->dy += dy;
}
```

Following the adjustment of the necessary fields within the `GXText` structure, the `text_scale` function invokes the `reset_pts` function to do the work:

```
  reset_pts( text, text->x, text->y, text->dx, text->dy );
```

The call to `reset_pts` ensures that no rounding errors from previous actions have adversely affected the quality of the `Text` object being displayed.

The critical support functions required by the `text_copy` feature introduced in the next section have already been seen, but let's go there anyway.

Copying a `Text` Object

To accomplish the `text_copy` method seen in Listing 24.29 for duplicating the `Text` object, you need only apply functions already introduced in this chapter.

Listing 24.29 The `text_copy` Function

```
static void text_copy( GXObjPtr obj )
{
  GXTextPtr temp_data;

  (*obj->deselect)( obj );

7:    temp_data = copy_gxtext( (GXTextPtr)obj->data,
8:                             OFFSET, OFFSET );
9:    create_text( NULL, temp_data );
10:
11:   freeGXText(temp_data);
12:   XtFree((char *)temp_data);
13: }
```

By first deselecting the active `Text` object, the handles associated with it are removed from the screen

```
5:   (*obj->deselect)( obj );
```

following which the `copy_gxtext` function can be employed, specifying the value of `OFFSET` to prevent the copy from exactly overlaying the original object:

```
7:    temp_data = copy_gxtext( (GXTextPtr)obj->data,
8:                             OFFSET, OFFSET );
```

The temporary `GXText` structure is then used in a call to `create_text`

```
9:   create_text( NULL, temp_data );
```

and the memory associated with the temporary structure is freed:

```
11:   freeGXText(temp_data);
12:   XtFree((char *)temp_data);
```

The following section presents the last area of functionality required by the `Text` object—saving and restoring the object-specific data.

Saving and Restoring a `Text` Object

The ability to save and restore objects contained in the Graphics Editor, as introduced in Chapter 19, provides a mature level of capability to our application.

The `text_save` method found in Listing 24.30 adheres to the save strategy introduced earlier.

Listing 24.30 The `text_save` Function

```
1:  static void text_save( FILE *fp, GXObjPtr obj )
2:  {
3:      GXTextPtr text = (GXTextPtr)obj->data;
4:
5:      fprintf( fp, "TEXT [ str x y ]\n" );
6:      fprintf( fp, "%s\n %d %d\n",
7:               text->text, text->x, text->y );
8:  }
```

The `text_save` function extracts the `GXText` structure from the common object structure

```
3:     GXTextPtr text = (GXTextPtr)obj->data;
```

writes a simple tag line showing the format of the data to follow

```
5:     fprintf( fp, "TEXT [ str x y ]\n" );
```

and then writes the data

```
6:      fprintf( fp, "%s\n %d %d\n",
7:               text->text, text->x, text->y );
```

which consists of simply the location of the object and text string associated with it.

Notice the `newline` character inserted after the `%s` format token in the data line. Because it is possible for the user to embed spaces in the text value, we must ensure that the restore function can distinguish completely the text string from the location.

Listing 24.31 shows how the data written by the `text_save` method is retrieved to the editor.

Listing 24.31 The `GXTextLoad` Function

```
1:  void gxTextLoad( FILE *fp, GXObjPtr obj )
2:  {
3:      char text[256], *ptr;
4:
5:      GXTextPtr data;
6:      int x, y;
7:
8:      fgets( text, 256, fp );
9:      if( (ptr = strchr( text, '\n' )) != NULL ) {
10:         *ptr = '\0';
11:     }
12:     fscanf( fp, "%d %d\n", &x, &y );
13:
14:     data = create_gxtext( text, x, y,
15:                           plain_simplex, plain_simplex_p );
16:
17:     create_text( obj, data );
18: }
```

The `GxLoadText` function reads first the string using the C `fgets` function reads until it finds the `newline` character written by the `text_save` method

```
8:     fgets( text, 256, fp );
```

and then strips the newline character from the string because it cannot be represented by a vector font set:

```
    if( (ptr = strchr( text, '\n' )) != NULL ) {
         *ptr = '\0';
    }
```

After retrieving the object's location

```
    fscanf( fp, "%d %d\n", &x, &y );
```

the restore function has the necessary components to create a GXText structure

```
    data = create_gxtext( text, x, y,
                          plain_simplex, plain_simplex_p );
```

which it passes to the create_text function to create the Text object and add it to the list managed by the editor application:

```
    create_text( obj, data );
```

With the completion of the discussion concerning saving and restoring the Text object, you have completed the addition of all objects supported by the editor application. Congratulations!

Next Steps

Chapter 25, "Introduction to PostScript," will introduce the PostScript programming language as we work our way closer to the next goal of adding print capability to the Graphics Editor application.

Part VI

Adding a Print Driver

Chapter 25

In this chapter

Introduction to PostScript

The usefulness of the Graphics Editor is significantly advanced by the addition of print capability. There are several ways to approach the task of printing; therefore, I have selected a simple approach that easily allows for future enhancements.

To accomplish the Graphics Editor print function, the PostScript page description language will be used. PostScript enables us to communicate with a PostScript-compatible printer to effectively describe the page to be printed.

The following sections introduce the PostScript language that is used in future chapters to print the graphics objects drawn by the Graphics Editor.

Note

This chapter covers only a portion of PostScript Level 1 and Level 2, which are the earlier versions of PostScript created by Adobe. The current version is Level 3.

PostScript

PostScript is a programming language that describes the appearance of a printed page. It was developed by Adobe in 1985 and has become an industry standard for printing and imaging.

All major printer manufacturers make printers that contain or can be loaded with PostScript interpreter software. A PostScript file can generally be identified by a `ps` suffix. Like C language source code files, PostScript files are plain text and can be viewed using your favorite editor.

PostScript is an interpreted, stack-oriented language for describing—in a device-independent fashion—the way in which pages can be composed of characters, shapes, and digitized images in black and white, grayscale, or color.

Concerning the fields of text and graphics, PostScript is the most widely used printer controller in the industry. It gives computer users total control over text, graphics, color-separations, and halftones.

The page description language recognizes a page the user creates as a unit and converts the elements of the page into control data for the output device. The printing engine receives the control data in its own format and resolution. For the printed page the resolution can be up to 300 dots per inch (dpi).

PostScript language programs are used for communication between a software product such as the Graphics Editor and a printing system, and they enable the user to mix text, graphics, and images from various sources.

The following section introduces the PostScript language by drawing on what you already know about the C programming language.

Learning PostScript

The PostScript page description language serves as an interface between an application program and a printing system (usually a printer). A C program can generate the PostScript language code needed by a printer for rendering the pages described.

Using the C programming language classic “Hello World” example,

```
main()
{
   printf("hello world\n");
}
```

to print the character string `hello world` to the screen, we will begin our instruction in PostScript.

The translation of this C program into PostScript would look like the following:

```
%!
/Courier findfont 10 scalefont setfont
   0 100 moveto
   (hello world) show
   0 88 moveto
showpage
```

At first glance, you don’t see similarities between the C program sample and its PostScript translation. Because PostScript is a graphical programming language, you must specify how the character string will be printed. This requires specifying the typeface and point size:

```
/Courier findfont 10 scalefont setfont
```

`Courier` is the font that is specified with the `findfont` command, and `10` is the point size PostScript is told to scale the font to with the `scalefont` command. The last keyword, `setfont`, instructs the PostScript interpreter to make the font active.

You next must specify the origin for placing the string:

```
0 100 moveto
```

This instruction moves `0` points to the right and `100` points up from the lower-left corner of the paper sheet.

Next, you designate the line feed by a vertical motion downward from the previous location:

```
0 88 moveto
```

Finally, you instruct the interpreter to display the page that's been described:

```
 showpage
```

Notice the order of the sample PostScript commands relative to the parameters required by the command. As with the `moveto` command

```
0 88 moveto
```

the values `0` and `88` are the x and y coordinates needed by the command, but they precede `moveto`. This syntax is necessary because the PostScript language is stack based.

Stacks

A *stack* is a data structure implemented as a last in, first out (LIFO) list. On a stack, the last item added to the structure is the first item removed.

Everything read by PostScript is placed (pushed) on a stack internal to the interpreter. When the interpreter reads a command or instruction requiring arguments, the appropriate number of parameters are popped (removed) from the stack to satisfy the command.

Several stacks are in a PostScript system, but only two are important for this discussion: the *operand stack* and the *dictionary stack*.

The operand stack is a stack with arguments to procedures (or operators, using PostScript vernacular) that are pushed prior to use. The dictionary stack is, as the name implies, for dictionaries, and it provides storage for variables.

Looking again at the `moveto` command, envision a list where everything read by the interpreter is placed. In the end, the list would appear as follows:

```
2. moveto
1. 88
0. 0
```

When the `moveto` command is reached, the elements placed on the list are removed in the reverse order they were inserted.

The following section introduces the commands needed by the Graphics Editor for printing the objects drawn on the canvas.

PostScript Commands

PostScript is a highly capable language. The following sections discuss many PostScript operators used by the editor application and provide a description for each.

Comments

Inserting comments into a PostScript file is done using the percent (`%`) character. As with other languages, anything following on the same line as a comment token is ignored by the interpreter.

The special comment `%!` token used as the first two characters of a PostScript program is seen as a tag marking the file as PostScript code by many systems, including the UNIX `lpr` command.

It is a good idea to start every PostScript document with the `%!` token, because it ensures that every spooler and printer the document can encounter recognizes it as PostScript code.

In addition to the information found in the comment specifiers, PostScript understands numerous commands. The following section introduces PostScript conventions for forming the commands used by the Graphics Editor for printing the canvas window.

PostScript Programming

Programming in PostScript is quite easy. The fundamentals are that you *push* operands onto the operand stack by naming them, and then you invoke the operands to employ them.

The challenge in programming with PostScript is in knowing which operand to use, and when.

Operators to draw and put text on the screen make up the bulk of PostScript operands. A couple of operands, however, are used for maintaining the program itself.

The first of these operators is the `def` operand, which is responsible for entering a definition into the top-most dictionary on the dictionary stack.

Note

A *dictionary* is a collection of name-value pairs. All named variables are stored in dictionaries. In addition, all available operators are stored in dictionaries along with their code. The dictionary stack is a stack of all currently open dictionaries. When a program refers to some key, the interpreter wanders down the stack looking for the first instance of that key in a dictionary. In this manner, names can be associated with variables and a simple form of managing scope is implemented. Conveniently, dictionaries can be given names and can be stored in other dictionaries.

The top operand on the operand stack is the value, and the operand below the value is the key and is typically a name.

For instance, defining the name `x` and assigning it a value of `5` in PostScript would be done as follows:

```
/x 5 def
```

Notice the use of the forward slash before the `x`. This ensures that the *name* `x`, and not any value it might represent, will be pushed onto the operand stack in any defined dictionary stack.

The `def` operand is also used to define new operators. The value in this case is just a procedure. The following code defines an operator `foo`, which adds its top-most two operands and multiplies the result with the next operand on the stack:

```
/foo {add mul} def
```

Remember that operands, which return results, *push* them onto the stack so they can be used later.

An important point to know when defining procedures is that the elements in a procedure are not evaluated until the procedure is invoked.

That means that in the procedure

```
{1 2 add 3 mul}
```

the actual names `add` and `mul` are stored in the procedure array. This is different from an actual array in which the components are evaluated when the array is created.

This delayed evaluation of procedure components has two important effects. First, the definition of an operator used in a procedure is the one that is in effect when the procedure is run, not when it is defined. Second, because each operator must be looked up each time the procedure is invoked, things can be a little slow.

Fortunately, PostScript provides the `bind` operator to replace each name in a procedure object with its current definition:

```
/foo {add mul} bind def
```

If `add` or `mul` is redefined after defining `foo`, `foo` will have the same behavior as before. Without the use of `bind`, the behavior of `foo` would change. Comfortable with the basic syntax forming a PostScript operator, we will look at a summary of some of the commands understood by PostScript.

Table 25.1 shows many of the commands provided with Level 1 PostScript.

Table 25.1 PostScript Level 1 Commands

Operator	*Example*	*Description*
`add`	`num1 num2 add num3`	Returns the addition of the two arguments.
`arc`	`x-coord y-coord r ang1 ang2 arc`	Adds an arc to the current path.The arc is generated by sweeping a line segment of length `r`, and tied at the point (`x-coord y- coord`), in a counter-clockwise direction from an angle `ang1` to an angle `ang2`. Note: A straight- line segment will connect the current point to the first point of the arc if they are not the same.
`begin`	`dict begin`	Pushes the dictionary `dict` onto the dictionary stack, where it can be used for `def` and name lookup. This operator enables an operator to set up a dictionary for its own use (for example, for local variables).
`bind`	`procedure1 bind procedure2`	Goes through `procedure1` and replaces any operator names with their associate operators. Names that do not refer to operators are left alone. Operators within `procedure1` that have unrestricted access will have `bind` called on themselves before they are inserted into the procedure. The new procedure, with operators instead of operator names, is returned on the stack as `procedure2`. The main effect and use of this operator is to reduce the amount of name lookup done by the interpreter. This speeds up execution and ties down the behavior of operators.
`clip`	`clip`	Intersects the current clipping path with the current path and sets the current clipping path to the results. Any part of a path drawn after calling this operator that extends outside this new clipping area will simply not be drawn. If the given path is open, `clip` will treat it as if it were closed. Also, `clip` does not destroy the current path when it is finished; it can be used for other activities. It is important to note that there is no easy way to restore the clip path to a larger size after it has been set. The best way to set the clip path is to wrap it in a `gsave` and `grestore` pair.

Operator	Example	Description
`closepath`	`closepath`	Adds a line segment to the current path from the current point to the first point in the path. This closes the path so it can be filled.
`charpath`	`string bool charpath`	Takes the given string and appends the path, which the characters define to the current path. The result can be used as any other path for stroking, filling, or clipping. The Boolean argument informs `charpath` what to do if the font is not designed to be stroked. If the Boolean is `true`, the path will be modified to be filled and clipped (but not stroked). If the Boolean is `false`, the path will be suitable to be stroked (but not filled or clipped).
`curveto`	`x1 y1 x2 y2 x3 y3 curveto`	Draws a curve from the current point to the point (x3, y3) using points(x1, y1) and (x2, y2) as control points. The curve is a Bézier cubic curve. In such a curve, the tangent of the curve at the current point will be a line segment running from the current point to (x1, y1), and the tangent at (x3, y3) is the line running from (x3, y3) to (x2, y2).
`def`	`name value def`	Associates the name with a value in the dictionary at the top of the dictionary stack. This operator essentially defines names to have values in the dictionary and is used to define variables and operators.
`div`	`num1 num2 div num3`	Returns the result of dividing num1 by num2. The result is always a real.
`dup`	`object dup object object`	Pushes a second copy of the topmost object on the operand stack. If the object is a reference to an array, string, or similar composite object, only the reference is duplicated; both references will still refer to the same object.
`end`	`end`	Pops the topmost dictionary from the dictionary stack. The dictionary below it becomes the new current dictionary.
`exch`	`value1 value2 exch value2 value1`	Simply exchanges the top two items on the operand stack. It does not matter what the operands are.
`fill`	`fill`	Closes and fills the current path with the current color. Any ink within the path is obliterated. Note that `fill` blanks out the current path as if it had called `newpath`. If you want the current path preserved, you should use `gsave` and `grestore` to preserve the path.
`findfont`	`name findfont font`	Looks for the named font in the font dictionary. If it finds the font, it pushes the font on the stack for later processing. It signals an error if the font cannot be found.

continues

Table 25.1 Continued

Operator	*Example*	*Description*
`for`	`initial increment limit proc for`	Executes `proc` repeatedly. The first time `proc` is executed, it will be given `initial` as the top operand. Each time it is executed after that, the top operand will be incremented by `increment`. This process will continue until the argument exceeds the `limit`.
`grestore`	`grestore`	Sets the current graphics state to the topmost graphics state on the graphics state stack and pops that state off the stack. This operator is usually used in conjunction with `gsave`.
`gsave`	`gsave`	Pushes a copy of the current graphics state onto the graphics state stack. The graphics state consists of (among other things): the current font and the current color.
`if`	`bool proc if`	Executes `proc` if `bool` is `true`.
`ifelse`	`bool proc1 proc2 ifelse`	Executes `proc1` if `bool` is `true`, and `proc2` otherwise.
`index`	`value_n ... value_0 n index value_n ... value_0 value_n`	Grabs the nth item off the operand stack (item is the one just under the index you push on the stack for the operator) and pushes it on top of the stack.
`lineto`	`x-coord y-coord lineto`	Adds a line into the path. The line is from the current point to the point (`x-coord y-coord`). After the line is added to the path, the current point is set to (`x-coord y-coord`). It is an error to call `lineto` without having a current point.
`moveto`	`x-coord y-coord moveto`	Moves the current point of the current path to the given point in user space. If a `moveto` operator immediately follows another `moveto` operator, the previous one is erased.
`mul`	`value1 value2 mul product`	Multiplies the first two operands on the stack and pushes the result back onto the stack. The result is an integer if both operands are integers and the product is not out of range. If the product is too big or one of the operands is a real, the result will be a real.
`newpath`	`newpath`	Clears the current path and prepares the system to start a new current path. This operator should be called before starting any new path, although some operators call it implicitly.
`pop`	`value pop`	Removes the top-most item from the operand stack.
`restore`	`state restore`	Restores the total state of the PostScript system to the state saved in `state`.

Operator	Example	Description
`rlineto`	`dx dy rlineto`	Adds a line into the path. The line is from the current point to a point found by adding `dx` to the current x and `dy` to the current y. After the line is added to the path, the current point is set to the new point. It is an error to call `lineto` without having a current point.
`rmoveto`	`dx dy rmoveto`	Moves the current point of the current path by adding `dx` to the current x and `dy` to the current y.
`rotate`	`angle rotate`	Rotates the user space counter-clockwise by angle degrees (negative angles rotate clockwise). The rotation occurs around the current origin.
`save`	`save state`	Gathers the complete state of the PostScript system and saves it in `state`. Possible errors resulting from this call include `limitcheck` and `stackoverflow`.
`scale`	`sx sy scale`	Scales the user coordinates. `Sx` in the horizontal direction and `sy` in the vertical direction will multiply all coordinates. The origin will not be affected by this operation.
`scalefont`	`font size scalefont font`	Takes the given font and scales it by the given scale factor. The resulting scaled font is pushed onto the stack. A size of 1 produces the same sized characters as the original font, 0.5 produces half-size characters, and so on.
`setfont`	`font setfont`	Sets the current font to be `font`. This font can be the result of any font creation or modification operator. This font is used in all subsequent character operations like show.
`setgray`	`gray-value setgray`	Sets the current intensity of the ink to `gray-value`, which must be a number from 0 (black) to 1 (white). This will affect all markings stroked or filled onto the page. This applies even to path components created before the call to `setgray`, as long as they have not yet been stroked.
`setlinewidth`	`width setlinewidth`	Sets the width of all lines to be stroked to `width`, which must be specified in points. A line width of 0 is possible and is interpreted to be a hairline, as thin as can be rendered on the given device.
`show`	`string show`	Draws the given string onto the page. The current graphics state applies, so the current font, font size, gray value, and current transformation matrix all apply. The current point determines the location for the text. The current point will specify the leftmost point of the baseline for the text.

continues

Table 25.1 Continued

Operator	*Example*	*Description*
`showpage`	`showpage`	Commits the current page to print and ejects the page from the printing device. `showpage` also prepares a new blank page.
`stroke`	`stroke`	Draws a line along the current path using the current settings. This includes the current line thickness, current pen color, current dash pattern, current settings for how lines should be joined, and what kind of caps they should have. These settings are the settings at the time the stroke operator is invoked. A closed path consisting of two or more points at the same location is a degenerate path. A degenerate path will be drawn only if you have set the line caps to round caps. If your line caps are not round caps, or if the path is not closed, the path will not be drawn. If the path is drawn, it will appear as a filled circle center at the point.
`sub`	`num1 num2 sub num3`	Returns the result of subtracting `num2` from `num1`.
`translate`	`x-coord y-coord translate`	Moves the origin to the point (`x-coord`, `y-coord`) in the current user space.

The file description page created using the PostScript commands listed in Table 25.1 can be viewed on your computer screen using a PostScript viewer utility.

Viewing PostScript Files

Although generally used to communicate with a print device, PostScript can be viewed on the computer monitor. This is especially useful when troubleshooting a print driver or learning PostScript language.

Ghostscript, a utility provided with Linux for viewing PostScript files, is invoked with the command `gv`.

Ghostscript is the name of a set of software that provides an interpreter for the PostScript language. It has the capability to convert PostScript language files to many raster formats, view them on displays, and print them to printers that don’t have PostScript language capability built in.

The Ghostscript software is also able to interpret Portable Document Format (PDF) files and has the capability to convert PostScript language files to PDF (with some limitations) and vice versa.

Ghostscript provides a set of C procedures through the Ghostscript library that implement the graphics capabilities that appear as primitive operations in the PostScript language.

If you are not using Linux, or if the Ghostscript software is not installed on your system, you can obtain it from `ftp://ftp.cs.wisc.edu/ghost`.

Comments Understood by Ghostscript

The comments found in Listing 25.1 are recognized by the Ghostscript interpreter and are useful when forming the prolog of a page description file. There are no required elements when forming a page prolog; however, the more directives you employ the closer your description will be honored.

Note

As described in the text, the prolog is the initial area of the PostScript file. The prolog, when it is used, contains any procedures that are used in the body of the document. These are surrounded by

```
%%BeginProlog
definitions go here
%%EndProlog
```

In keeping with the `BeginProlog` and `EndProlog` directives, the word *prolog* is often used in lieu of what might be grammatically more correct to refer to as *prologue*.

The terms, however, in the context of a PostScript document, are synonymous.

Listing 25.1 Comments Understood by PostScript

```
%!PS-Adobe-<real> [EPSF-<real>]
%%BoundingBox: <int> <int> <int> <int>|(atend)
%%CreationDate: <textline>
%%Orientation: Portrait|Landscape|(atend)
%%Pages: <uint>|(atend)
%%PageOrder: Ascend|Descend|Special|(atend)
%%Title: <textline>
%%DocumentMedia: <text> <real> <real> <real> <text> <text>
%%DocumentPageSizes: <text>
%%EndComments

%%BeginPreview
%%EndPreview

%%BeginDefaults
%%PageBoundingBox: <int> <int> <int> <int>|(atend)
```

continues

Listing 25.1 Continued

```
%%PageOrientation: Portrait|Landscape
%%PageMedia: <text>
%%EndDefaults

%%BeginProlog
%%EndProlog

%%BeginSetup
%%PageBoundingBox: <int> <int> <int> <int>|(atend)
%%PageOrientation: Portrait|Landscape
%%PaperSize: <text>
%%EndSetup

%%Page: <text> <uint>
%%PageBoundingBox: <int> <int> <int> <int>|(atend)
%%PageOrientation: Portrait|Landscape
%%PageMedia: <text>
%%PaperSize: <text>

%%Trailer
%%EOF

%%BeginDocument: <text> [<real>[<text>]]
%%EndDocument

%%BeginBinary: <uint>
%%EndBinary

%%BeginData: <uint> [Hex|Binary|ASCII[Bytes|Lines]]
%%EndData
```

Many of these comments are used in the page description file created by the Graphics Editor in the following chapters.

Next Steps

The use of color in a PostScript file poses a certain challenge that is complicated by trying to form a single page description file to work when interpreted by both color and black-and-white printers.

The next chapter introduces a PostScript color conversion function that will enable color images to be converted to grayscale during interpretation by black-and-white printers.

Chapter 26

In this chapter

Color Versus Black and White

Formatting the image data contained in the canvas area of the Graphics Editor as a color image creates an incompatibility with black and white printers. However, the capability to support printing in color or black and white from a single PostScript page description file is possible and is the focus of this chapter.

Determining a Printer's Capability

You must consider whether the destination printer specified for receiving output from the Graphics Editor requires the color values contained in the page description file to be gray-scaled to correctly interpret them.

If you simply try testing for the presence of a color operator such as `colorimage`, your print system will fail on PostScript Level 2 printers. Many applications make the mistake of testing for the presence of an operator (`colorimage`, `setcmykcolor`, and so forth) to determine the printer's capability; however, this is not sufficient. Some Level 1 PostScript black and white devices and all Level 2 devices include the color operators.

Two reasons exist for an application such as the Graphics Editor to care about the color capabilities of a printer. First is to determine whether the printer will accept PostScript language programs using the color extensions (`colorimage`, `setcmykcolor`, and so forth). The second reason is to determine whether the printer, as currently configured, will actually produce color output.

If the printer will accept the color extensions, the application should send it color-formatted output, whether or not the printer will actually produce color. In this way, the PostScript language file captures the application's intent, and color will be produced if the file is later diverted to a color printer.

The following section describes how to determine whether a printer is capable of PostScript color extensions and how to accommodate those printers that are not.

Defining Color Images for Black and White Printers

The `write_ps_color_conv` function in Listing 26.1 writes to the page description file referenced by the parameter the PostScript code needed to convert a color image to gray-scale for printers not able to process color extension operands.

Listing 26.1 Gray-Scaling Color Raster Data

```
static void write_ps_color_conv( FILE *fp )
{
  int i = 0;

  static char *ColorConvStr[] =  {
    "% define colorimage if it is not defined            \n",
    "/colorimage where   % do we know about colorimage?  \n",
    "  { pop }           % yes: pop off the dict returned\n",
    "  {                 % no:  define one\n             \n",
    "    /colortogray {  % define an RGB->I function     \n",
    "      /rgbdata exch store    % call input rgbdata   \n",
    "      rgbdata length 3 idiv \n",
    "      /npixls exch store    \n",
    "      /rgbindx 0 store      \n",
    "      0 1 npixls 1 sub {    \n",
    "        grays exch          \n",
    "        rgbdata rgbindx       get 20 mul    % Red   \n",
    "        rgbdata rgbindx 1 add get 32 mul    % Green \n",
    "        rgbdata rgbindx 2 add get 12 mul    % Blue  \n",
    "        add add 64 idiv       \n",
    "        put \n",
    "        /rgbindx rgbindx 3 add store  \n",
    "     } for \n",
    "      grays 0 npixls getinterval \n",
    "    } bind def \n",
    "                                                 \n",
    "    % Utility procedure for colorimage operator. \n",
    "    % This procedure takes two procedures off the\n",
    "    % stack and merges them into a single one.   \n",
    "                        \n",
    "    /mergeprocs { % def \n",
    "      dup length        \n",
    "      3 -1 roll         \n",
    "      dup               \n",
    "      length      \n",
    "      dup         \n",
    "      5 1 roll    \n",
    "      3 -1 roll   \n",
    "      add         \n",
40:     "      array cvx   \n",
41:     "      dup         \n",
42:     "      3 -1 roll   \n",
43:     "      0 exch      \n",
44:     "      putinterval \n",
45:     "      dup         \n",
46:     "      4 2 roll    \n",
47:     "      putinterval \n",
48:     "    } bind def    \n",
49:     "                                         \n",
50:     "    /colorimage { % def                  \n",
51:     "      pop pop     % remove false 3 operands \n",
52:     "      {colortogray} mergeprocs           \n",
53:     "      image    \n",
54:     "    } bind def \n",
55:     "  } ifelse        % end of false case\n \n",
56:     NULL };
57:
58:   while( ColorConvStr[i] ) {
59:     fprintf( fp, "%s", ColorConvStr[i++] );
60:   }
61: }
```

The color conversion function in Listing 26.1 can seem overwhelming if your introduction to PostScript syntax began with Chapter 25, "Introduction to PostScript." However, this function is really not difficult to understand and will serve as an intermediate introduction to the language before proceeding to implement print capability in the Graphics Editor in Chapter 27, "Working with XImages and Colormaps."

The function begins by determining whether the operand `colorimage` is defined in the existing dictionaries:

```
7:     "/colorimage where
```

If it is defined, simply `pop` the returned value from the stack

```
 8:     "  { pop }
```

and let normal interpretation of the file continue.

Otherwise, it is necessary to define it. The function defined in absence of the `colorimage` operand is a procedure to convert the color values of a raster image into weighted gray-scale values:

```
10:     "    /colortogray {
```

Lines 11–15 invoke the `rgbdata`, define the number of color components, and prepare to substitute the color values with proportionally weighted gray values based on the intensity of the corresponding color component:

```
17:     "        rgbdata rgbindx       get 20 mul    % Red   \n",
18:     "        rgbdata rgbindx 1 add get 32 mul    % Green \n",
19:     "        rgbdata rgbindx 2 add get 12 mul    % Blue  \n",
```

The sum of the values is found and returned to the stack

```
20:     "        add add 64 idiv       \n",
21:     "        put \n",
```

before continuing to the next line of data:

```
23:     "      } for \n",
24:     "       grays 0 npixls getinterval \n",
```

Next, lines 31–48 define `mergeprocs`, a process which performs some extensive stack manipulation to merge the `colorimage` and `colortogray` functions. This is defined for execution in line 52:

```
52:     "      {colortogray} mergeprocs              \n",
```

When the `mergeprocs` is placed on the stack, it is given the image for processing

```
53:     "      image    \n",
```

and the work is done.

The effect of this PostScript segment is to create a single page description file that will print successfully on both color and black and white printers.

Next Steps

In the next chapter we will look closely at the implementation of the print capability for the Graphics Editor, and see how the color conversion functions introduced here fit into the overall print system.

In this chapter

Chapter 27

Working with XImages and Colormaps

The introduction to the PostScript page description language provided by the preceding chapters has prepared you for implementing print capability in the Graphics Editor project.

To accomplish this, it is necessary to create an `XImage` of the drawing area window and parse the image data to form a digitized raster image understandable by PostScript.

The next section introduces the `gx_print` function defined as the control action for the print icon of the application's menu panel.

Note

The code introduced in this chapter is targeted for the `gxGx.c` source file. Important too is that non-static function definitions have prototypes for them placed in the `gxProtos.h` header file. Finally, it will be necessary to ensure that the headers `time.h`, `unistd.h`, and `sys/stat.h` have preprocessor `include` directives for each of them placed at the beginning of the `gxGx.c` file.

Printing the Canvas

The print capability of the Graphics Editor prompts the user for the destination of the print action.

As you'll see shortly, the print function tries to determine whether the destination specified is a printer device contained in the `/dev` directory. If a match to it is not found, the destination is treated like a file with the PostScript page description for the canvas being written to it.

The `gx_print` function was previously defined as a stump function to satisfy the link phase of building the application. The actual function definition is presented in Listing 27.1.

Listing 27.1 The `gx_print` Function

```
1:  void gx_print( void )
2:  {
3:    char *_filename = "/tmp/gx-print-data.ps";
4:
5:    XImage *xi = NULL;
6:    FILE   *fp = NULL;
7:
8:    Dimension    width, height;
9:    int          numcols = 0;
10:
11:   byte   *data;
12:   byte   Red[MAXCOLORMAPSIZE],
13:          Green[MAXCOLORMAPSIZE],
14:          Blue[MAXCOLORMAPSIZE];
15:   XColor cm[MAXCOLORMAPSIZE];
16:
17:   fp = fopen( _filename, "w+" );
18:
19:   if( fp ) {
20:     char *printTo = gxGetFileName();
21:
22:     get_image( GxDrawArea, &xi, &width, &height );
23:     write_ps_prolog( fp, width, height );
24:
25:     if( xi ) {
26:       data =
27:         set_color_data(GxDrawArea, xi, width, height,
28:                        cm, Red, Green, Blue, &numcols);
29:       write_ps( fp, xi, cm, width, height );
30:
31:       XtFree( (char *)data );
32:       XDestroyImage( xi );
33:
34:       fclose( fp );
35:     }
36:     doPrintTo( _filename, printTo );
37:   }
38: }
```

The `gx_print` function defines a temporary storage file for constructing the PostScript page definition of the canvas window:

```
3:   char *_filename = "/tmp/gx-print-data.ps";
```

The function begins by opening the file for writing

```
fp = fopen( _filename, "w+" );
```

and prompting the user for the destination of the print action:

```
char *filename = gxGetFileName();
```

The `gx_print` function then creates an `XImage` from the canvas window and writes the dimensions of the image in a prolog for the print file being constructed:

```
get_image( GxDrawArea, &xi, &width, &height );
write_ps_prolog( fp, width, height );
```

These functions are defined in listings discussed later in this chapter.

If an `XImage` was successfully created, the image data is parsed to determine the pixel values of the cells forming the image:

```
  data =
    set_color_data(GxDrawArea, xi, width, height,
                   cm, Red, Green, Blue, &numcols);
```

Finally, the raster image is written to the file

```
  write_ps( fp, xi, cm, width, height );
```

and the destination specified by the user is processed with a call to `doPrintTo`:

```
doPrintTo( _filename, filename );
```

It is now necessary to look at the function in more detail, starting with the creation of an `XImage` from the drawing area.

Creating an `XImage`

Listing 27.2 defines the `get_image` function for creating an `XImage` from the canvas window.

Listing 27.2 Creating an Image from an X Window

```
static void get_image( Widget w, XImage **xi,
                 Dimension *width, Dimension *height )
{
  Window     window;
  int        max_w, max_h, x = 0, y = 0;

  XtVaGetValues( w,
                 XtNwidth,      width,
                 XtNheight,     height,
                 NULL );
```

continues

Listing 27.2 Continued

```
11:
12:   *width  -= 2; *height -= 2;
13:
14:
15:   window = XtWindow( w );
16:
17:   /* ensure even num of rows, postscript demands it */
18:   *height -= ((*height)%2);
19:
20:   if( *width > 1 && *height > 1 ) {
21:
22:       *xi = XGetImage( XtDisplay(w), window,
23:                        x, y, *width, *height,
24:                        AllPlanes, ZPixmap );
25:   }
26: }
```

The width and height of the image extracted from the GxDrawArea are important to the generation of the PostScript description for it. Therefore, the width and height values are returned separately to the calling function for use elsewhere:

```
7:    XtVaGetValues( w,
8:                   XtNwidth,       width,
9:                   XtNheight,      height,
10:                  NULL );
```

The width and height of the widget are reduced slightly

```
12:   *width  -= 2; *height -= 2;
```

to ensure that the requested image is completely contained in the window. An even number of columns must be specified in the raster image passed to the PostScript interpreter as the language requires it:

```
18:   *height -= ((*height)%2);
```

Finally, the function is able to request:

```
22:       *xi = XGetImage( XtDisplay(w), window,
23:                        x, y, *width, *height,
24:                        AllPlanes, ZPixmap );
```

The `XGetImage` function creates an `XImage` structure definition from the window specified as the second parameter.

The image created does not have to encompass the entire window. Through use of the `x`, `y` and `width`, `height` parameters, an image smaller than the entire window can be created.

The constants `AllPlanes` and `ZPixmap` are used to specify which planes should be employed when creating the image and what the format of the image data should be.

If the format argument is `XYPixmap`, the image contains only the bit planes you passed to the plane mask if something other than `AllPlanes` is used as the argument. If the plane mask argument only requests a subset of the planes of the display, the depth of the returned image will be the number of planes requested. If the format argument is `ZPixmap`, `XGetImage` returns as `0` the bits in all planes not specified in the plane mask argument.

Having determined the width and height of the image, you can write the prolog for the page description file.

Creating a PostScript Prolog

The creation of the PostScript Prolog is the responsibility of the `write_ps_prolog` function shown in Listing 27.3.

Listing 27.3 Defining the PostScript Prolog

```
static void write_ps_prolog( FILE *fp, int w, int h )
{
  time_t btime;
  time( &btime );

  fprintf(fp, "%%!PS-Adobe-2.0 EPSF-2.0\n"          );
  fprintf(fp, "%%%%Creator: J. R. Brown\n"          );
  fprintf(fp, "%%%%Title: 2D Graphic Editor\n"      );
  fprintf(fp, "%%%%CreationDate: %s", ctime(&btime));
  fprintf(fp, "%%%%Pages: 1\n");
  fprintf(fp, "%%%%BoundingBox: 0 0 %d %d\n", w, h );
  fprintf(fp, "%%%%EndComments\n");
  fprintf(fp, "%%%%EndProlog\n\n");
  fprintf(fp, "%%%%Page: 1 1\n\n");
}
```

Although some of the data written in the prolog is for informational purposes only, as discovered in Chapter 25, in the section “Comments Understood by Ghostscript,” page 489, the prolog is an important element of the output file.

Following the creation of the PostScript prolog, you are ready to parse the image created by `XGetImage` to determine the colors used in the many cells that compose the image, as seen in Listing 27.4.

Parsing the X Colormap

An X `Colormap` is an array of pixel values managing all colors currently displayed on the screen.

The `set_color_data` function found in Listing 27.4 parses the default `Colormap` to determine the colors used by the `XImage` and create the raster image definition that will be passed to the PostScript page description file.

Listing 27.4 Determining Image Color Use

```
#define MAXCOLORMAPSIZE 65536
typedef unsigned char byte;
static byte *set_color_data( Widget w, XImage *image,
                             int width, int height,
                             XColor *colors, int *n )
{
  Boolean mapcols[MAXCOLORMAPSIZE],
          colused[MAXCOLORMAPSIZE];

  Display *dsp = XtDisplay(w);
  int     x = 0, i;

  byte *dptr, *data, *iptr = (byte *) image->data;

  dptr =data=(unsigned char *)XtMalloc(height * width);
  memset( colused, False,
          sizeof( Boolean ) *  MAXCOLORMAPSIZE );

  setStatus( "Gathering color data... please wait" );
  for ( i = 0; i < MAXCOLORMAPSIZE; i++) {
    colors[i].pixel = i;
    colors[i].flags = DoRed | DoGreen | DoBlue;

    XQueryColor(dsp,
                DefaultColormap(dsp, 0), &colors[i]);
  }

  setStatus( "Determing color usage ..." );
  for (i = 0;
       i < image->bytes_per_line*height; i++, iptr++) {

    if (x >= image->bytes_per_line)
      x = 0;

    if (x < width) {
      colused[*iptr] = True; /* mark this color as used */
      *dptr++ = *iptr;
    }
    x++;
42:   }
43:
44:   setStatus( "Processing color data ..." );
45:   for (i = 0; i < MAXCOLORMAPSIZE; i++) {
46:     if( colused[i] ) {
47:
48:       mapcols[i] = *n;
49:
54:       (*n)++;
55:     }
56:   }
57:
58:   setStatus( "Transferring image colors..." );
59:   dptr = data;
60:   for (i = 0; i < width*height; i++) {
61:     *dptr = mapcols[*dptr];
62:     dptr++;
63:   }
64:
65:   return data;
66: }
```

The actual image data comprising the `XImage` data field is an array of pixel values where each pixel corresponds to an entry in the `Colormap` used to draw the image to the screen.

The `set_color_data` function in lines 22–28 creates an array of all color data for the pixels contained in the `Colormap`.

It then traverses the image data to determine which color is used:

```
38:       colused[*iptr] = True; /* mark this color as used */
```

The color data extracted from the `Colormap` is then traversed to see whether the color was used:

```
46:     if( colused[i] ) {
```

If the color was used, the number of the color (assigned according to order of occurrence) is used as an index into an array, tracking the colors used from the `Colormap`:

```
48:       mapcols[i] = *n;
```

The pixel values stored to reflect the colors used from the `Colormap` are transferred to the raster data array that is used as the image definition passed to the PostScript interpreter:

```
60:   for (i = 0; i < width*height; i++) {
61:     *dptr = mapcols[*dptr];
62:     dptr++;
63:   }
```

Finally, the data is returned to the calling function by passing back the pointer of the beginning of the data block:

```
return data;
```

With the raster image data formed for the page description file, the data can now be written to the temporary file using the `write_ps` function defined in Listing 27.5.

Writing the PostScript Page Definition File

The function `write_ps` provided in Listing 27.5 is responsible for forming the contents of the page description file that is eventually passed to the PostScript interpreter.

Listing 27.5 The `write_ps` Function

```
static void write_ps(FILE *fp, XImage *xi,
                     XColor *cm, Dimension h, Dimension w )
{
  write_ps_prolog( fp, w, h );

  fprintf(fp, "%% remember original state\n/saveorig save def\n\n");
  fprintf(fp, "%% define string to hold scanline's worth of data\n");
  fprintf(fp, "/imgstr %d string def\n\n", w );
  fprintf(fp, "%% build a temporary dictionary\n20 dict begin\n\n" );
  fprintf(fp, "%% define space for color conversion\n");
  fprintf(fp, "/grays %d string def %% space for gray scale line\n", w*3);
  fprintf(fp, "/npixls 0 def\n/rgbindx 0 def\n\n" );

  fprintf(fp, "%% corner of image\n%d %d translate\n\n", 25, 25);

  fprintf(fp, "%% size of image on paper\n%d %d scale\n\n",
  w > 550 ? 550 : w,
  h > 740 ? 740 : h ); /* could be changed for scaling */

  write_ps_color_conv( fp );

  fprintf( fp, "%d %d 8\n", w, h );
  fprintf( fp, "[%d 0 0 %d 0 0]\n", w, h );

  fprintf( fp, "{ currentfile\n\timgstr readhexstring pop }\n");
  fprintf( fp, "false 3 colorimage\n\n");

  write_ps_data( fp, xi, cm, w, h );

  fprintf( fp, "showpage\n\nend\n\nsaveorig restore\n\n");
  fprintf( fp, "%%%%Trailer\n\n%% End-of-file\n");
}
```

Following the writing of the PostScript prolog, the current state of the interpreter is stored

```
fprintf(fp, "%% remember original state\n/saveorig save def\n\n");
```

a variable is defined to store a single line of data from the raster image

```
fprintf(fp, "/imgstr %d string def\n\n", w );
```

and a temporary dictionary is defined:

```
fprintf(fp, "%% build a temporary dictionary\n20 dict begin\n\n" );
```

Then, in preparation for color translation, space is reserved for performing gray scaling of the color data

```
fprintf(fp, "/grays %d string def %% space for gray scale line\n", w*3);
```

additional variables are defined and initialized

```
fprintf(fp, "/npixls 0 def\n/rgbindx 0 def\n\n" );
```

and it's time to start working by assigning the placement of the image corner:

```
fprintf(fp, "%% corner of image\n%d %d translate\n\n", 25, 25);
```

The image being described in the file is then scaled to fit the paper

```
fprintf(fp, "%% size of image on paper\n%d %d scale\n\n",
w > 550 ? 550 : w,
h > 740 ? 740 : h ); /* could be changed for scaling */
```

and a call to the `write_ps_color_conv` function, introduced in Chapter 26, "Color Versus Black and White," is called to support color printers.

More housekeeping chores are addressed with the definition of the image data size and number of data bits:

```
fprintf( fp, "%d %d 8\n", w, h );
fprintf( fp, "[%d 0 0 %d 0 0]\n", w, h );
fprintf( fp, "{ currentfile\n\timgstr readhexstring pop }\n");
fprintf( fp, "false 3 colorimage\n\n");
```

Finally, the image data is written to the file using the `write_ps_data` function found in Listing 27.6.

The command to show the page is ordered

```
fprintf( fp, "showpage\n\nend\n\nsaveorig restore\n\n");
fprintf( fp, "%%%%Trailer\n\n%% End-of-file\n");
```

and the original state is restored before ending the page.

Listing 27.6 The `write_ps_data` Function

```
1:  static void write_ps_data( FILE *fp, XImage *xi,
2:                   XColor *cm, Dimension w, Dimension h )
3:  {
4:    int    rows, cols, cell;
5:
6:    for ( rows = 0; rows < h; rows++ ) {
7:      for( cols = 0; cols < w; cols++ ) {
8:
9:        cell = XGetPixel( xi, rows, cols );
10:
11:       fprintf( fp, "%2.2x", cm[cell].red   / 256 );
12:       fprintf( fp, "%2.2x", cm[cell].green / 256 );
13:       fprintf( fp, "%2.2x", cm[cell].blue  / 256 );
14:
15:     }
16:     fprintf( fp, "\n" );
17:   }
18: }
```

Using the `XImage` captured from the `GXDrawArea` window, the `write_ps_data` loops over the `width` and `height` of the image to generate a row and column coordinate. This `row` and `column` is then passed to the `XGetPixel` to get the pixel value defining the position:

```
9:       cell = XGetPixel( xi, rows, cols );
```

The pixel value returned by `XGetPixel` is used as an index into the `cm` array generated to reflect the colors in use by the image. Each color component is stored independently to the PostScript file in the range of 0–256:

```
11:       fprintf( fp, "%2.2x", cm[cell].red   / 256 );
12:       fprintf( fp, "%2.2x", cm[cell].green / 256 );
13:       fprintf( fp, "%2.2x", cm[cell].blue  / 256 );
```

As the entirety of the image cells are considered, the data gets written to the page description file.

With the contents of the PostScript description file complete, it is possible to parse the destination entered by the user at the beginning of the `gx_print` function to determine where the PostScript file should placed.

Directing the Output to a Printer or File

The value returned by the `gxGetFileName` function in Listing 27.1

```
20:     char *printTo = gxGetFileName();
```

could be the name of a printer configured for the system or, optionally, the name of a file.

The `doPrintTo` function in Listing 27.7 attempts to determine whether the destination is a printer by looking for a corresponding device in the `/dev/` directory; otherwise, it treats the user input as a file.

Listing 27.7 The `doPrintTo` Function for Determining the Print Data Destination

```
1:  static void doPrintTo( char *datafile, char *printTo )
2:  {
3:    struct stat sbuf;
4:    char cmd[128], lpDev[128];
5:
6:    FILE *fp = NULL;
7:
8:    if( printTo ) {
9:      sprintf( lpDev, "/dev/%s", printTo );
10:
11:     if( stat( lpDev, &sbuf ) < 0 ) {
12:       sprintf( cmd, "mv %s %s", datafile, printTo );
13:     } else {
14:        sprintf( cmd,
15:                 "unalias lp; lp -c -d %s %s;rm %s",
16:                 printTo, datafile, datafile );
17:     }
18:   } else {
19:       sprintf( cmd,
20:                "unalias lp;lp -c %s;rm %s",
21:                datafile, datafile );
22:   }
23:
24:   if((fp = popen(cmd, "r")) != NULL) {
25:     char buf[128];
26:     fgets( buf, sizeof( buf ), fp );
27:     pclose( fp );
28:   }
29:
30:   setStatus( "Printing complete!" );
31: }
```

If a printer of the name entered by the user does not match the name of a device in the `/dev` directory, a command is assembled to move the temporary PostScript page definition file to a file of the name entered:

```
12:       sprintf( cmd, "mv %s %s", datafile, printTo );
```

Otherwise, if the `stat` command reflects that a printer of the same name as entered by the user exists, the contents of the temporary PostScript file are directed to the `lp` command, specifying the printer name entered as the destination device:

```
14:        sprintf( cmd,
15:                 "unalias lp; lp -c -d %s %s;rm %s",
16:                 printTo, datafile, datafile );
```

If nothing was entered by the user, it is assumed that the system default printer is the desired destination

```
     sprintf( cmd,
              "unalias lp;lp -c %s;rm %s",
              datafile, datafile );
```

and a command is formed omitting the destination (`-d`) parameter.

With a command properly assembled, it is possible to open a stream with the operating system specifying the command to execute:

```
   if((fp = popen(cmd, "r")) != NULL) {
```

As the stream is opened for reading, it is possible to get the status of the command returned to the application:

```
     fgets( buf, sizeof( buf ), fp );
```

The stream must be closed when you are finished with it because a finite number of streams can be opened concurrently by an application:

```
     pclose( fp );
```

This completes the addition of print capability to the Graphics Editor.

Clearly, this approach to printing is direct and easy to implement. However, it could be improved upon; namely, the use of the raster image in the PostScript file does not ensure that the page described by the PostScript file looks exactly like the image on the canvas.

Because PostScript provides graphic primitives similar to the X Window System, a superior approach for printing the canvas would be to enable each of the objects in the editor to have a second draw method defined to use the appropriate PostScript function to draw the graphic objects. Although a more complex solution, it is one that would offer a higher quality print product and *WYSIWYG* (*what you see is what you get*) output.

Next Steps

The final chapters of this text propose ideas for furthering the Graphics Editor project and integrating it into other applications that could benefit from an annotation subsystem.

Part VII

What's Next?

Chapter 28

In this chapter

- *Attributes*
- *Rotating Objects*
- *Next Steps*

Extending the Graphics Editor

The Graphics Editor project, although able to draw, move, scale, and delete objects, requires many more capabilities before it is considered complete.

This chapter introduces features that the project structure lends itself to but that are left for you to implement.

Attributes

The definition of the common object data structure in Chapter 15, "Common Object Definition," page 305, provided fields within the structure to store the foreground and background colors as well as the line width and line style attributes of the editor objects.

These fields were included in the Save and Restore function of the editor application introduced in Chapter 19, in the section "Common Object Save and Restore," page 367.

However, the construction of the application did not provide for the selection of colors or line attributes by the user.

The following sections pose ideas to lead you in the extension of the Graphics Editor application to support the assignment of object attributes.

Color

The first consideration when extending the Graphics Editor project is how to provide a method for the user to assign, alter, and manipulate colors. Specific to this is the interface mechanism for making colors available to the user for selection and assignment.

Either by adding an attributes button panel adjacent to the control panel or a drop-down menu accessed from the canvas window, a palette should be defined for presenting allowable colors.

Note

The phrase *allowable colors* is important.

One shortcoming of the Graphics Editor application in its current state is the storage of `Pixel` values in the file created for the Save and Restore feature.

This is counted as a shortcoming because the order of the colors installed in the `Colormap`, reflected by the `Pixel` value index into the map, is arbitrary and affected entirely by the order in which the colors are requested.

As applications are started on the desktop, the colors they require are installed into the default `Colormap`. Changing the order in which the applications are started changes the order in which the colors are loaded.

For this reason, storing the `Pixel` value does not ensure that, in consecutive sessions of the application, objects restored from the data file will be drawn in the correct or original colors.

Defining *allowable colors* enables the Save and Restore feature to use an index that is meaningful to represent the color values.

After the interface mechanism is decided, the active objects selected by the user could have their color values altered.

In addition to affecting color settings, the Graphics Editor should account for the changing of line attributes, as described in the next section.

Line Attributes

Similar to assigning color values to the editor objects, the extension of the Graphics Editor project requires providing the capability to alter the line width and line style of the point-array–based editor objects.

Line Width

Valid values assigned as the line width attribute of point-array objects are any integer describing the thickness of the line.

Note

A Graphics Context created with the `line_width` field of the `XGCValues` structure set to 0 instructs the X Server to draw the line using the fastest algorithm possible.

The interface mechanism to support this could simply be an entry field where the user could provide a number. This clearly would require validation to ensure that characters are not erroneously entered. Optionally, an up and down arrow could be created to increment or decrement the object's line width respectively.

Line Style

Valid values for the line style attribute assignment of editor objects include `LineSolid`, `LineDashed`, and `LineDoubleDashed`.

These draw either a solid, dashed, or double-dashed line, as indicated by their name.

Note

A double-dashed line is drawn with one segment of the dash drawn in the foreground color assigned to the Graphics Context created for the invocation of the X Window Graphic Primitive and the next segment drawn in the background color of the `GC`.

A normal dashed line uses only the foreground color and skips alternate segments by drawing nothing.

Arc Angles

Complementing the line attributes of the previous section, attributes of the `Arc` object are provided by the `XArc` data structure but not addressed by the interface of the Graphics Editor application.

These are, specifically, the `angle1` and `angle2` fields of the structure. Review Chapter 8, "Vector Versus Raster Graphics," page 197, for a discussion of these fields and valid assignments.

Rotating Objects

In Chapter 11, "Graphic Transformations," in the section "Rotating," page 247, the mathematics required to rotate the point-array editor objects was introduced.

Considered an advanced feature of the editor, structuring the capability of assigning and altering object attributes should include a degree factor of rotation as an editable field.

The interface mechanism most suited for altering the degree field you'll add to the common object definition is the scale bar.

The scale should be assigned a minimum scale factor of `0` and a maximum of `360`, enabling the user to apply a meaningful value for degrees of rotation.

Next Steps

After the management of attributes outlined in this chapter is implemented, the Graphics Editor application will be complete. The steps remaining are limited only by your imagination.

The following chapter structures the addition of a context-sensitive help system to the editor application.

Chapter 29

In this chapter

Adding Context-Sensitive Help

I offer this chapter to the reader desiring to continue the development of the Graphics Editor Project beyond the scope of the text.

A suggested modification to the editor application's GUI is the support of a context-sensitive help facility.

As implied by the name, *context-sensitive help* enables a user to query the application's interface effectively for assistance in determining how to use the application. This capability would expand the hints currently provided in the status window at the bottom center of the interface.

Clearly, the graphical user interface for the Graphics Editor is not complex. However, the user has no intuitive way of knowing the specific mouse actions expected for creating each of the objects supported by the editor.

A context-sensitive help system would enable the user to press the F1 key to enter the help mode. The cursor would change to resemble a question mark, and the user would select an item from the application's interface for which help is desired.

Note

The purpose of this chapter is to pose an idea for your continued learning using the Graphics Editor project.

The following sections of this chapter do not fully implement a context-sensitive help system. Instead, a structure for the system is discussed, and key areas are expanded and explained.

Processing Help-Related Events

The key to a context-sensitive help system is trapping the user's selection of the F1 key to enter the help mode. Communication of this event to the Graphics Editor application can be gained in several ways.

For instance, an event handler, as introduced in Chapter 14, "Program Flow," can be assigned the various widgets of the graphical user interface. However, a more graceful approach is to define your own event loop to replace the `XtAppMainLoop` call used by the application.

The event loop is easy to construct and normally would only require an endless loop to extract events from the application's event queue, using `XtAppNextEvent` and subsequently sending the event to the application's widgets for processing using `XtDispatchEvent`.

The requirement of the context-sensitive help, however, is to determine when the user presses the F1 key. Therefore, it is necessary to sample the events removed from the queue.

Listing 29.1 demonstrates how a unique event loop can be defined to support the proposed help system.

Listing 29.1 The `HelpAppMainLoop` Function

```
void HelpAppMainLoop( XtAppContext app )
{
  XEvent event;

  for( ;; ) {
    XtAppNextEvent( app, &event );
    XtDispatchEvent( &event );

    if( helpEnabled == G_FALSE ) continue;

    switch( event.type ) {
    case KeyPress:
      processKeyPress( &event );
      break;
    default:
      break;
    }
  }
}
```

As with the `XtAppMainLoop`, our event loop will need access to the `XtAppContext` structure created from the call to `XtVaAppInitialize`.

The only requirement of the event loop is that the events be removed and dispatched continuously:

```
6:      XtAppNextEvent( app, &event );
7:      XtDispatchEvent( &event );
```

The sample loop from Listing 29.1 enables the help system to be optional by testing an implied global variable `helpEnabled`:

```
9:     if( helpEnabled == G_FALSE ) continue;
```

In the case of the help system being enabled, the `HelpAppMainLoop` event type is tested in search of the `KeyPress` event

```
12:     case KeyPress:
```

and when it is found, the value of the key pressed by the user is determined in a call to the `processKeyEvent` function found in Listing 29.2.

Listing 29.2 The `processKeyEvent` Function

```
1:  static void processKeyPress( XEvent *xe )
2:  {
3:    XEvent e;
4:    KeySym keysym;
5:    Widget help_widget;
6:
7:    Cursor cursor =
8:      XCreateFontCursor(XtDisplay(GxDrawArea), XC_question_arrow);
9:
10:  if( xe->type == KeyPress ) {
11:     keysym = XKeycodeToKeysym(XtDisplay(GxDrawArea),
12:                                 xe->xkey.keycode, 0);
13:
14:     if(keysym == XK_F1 || keysym == XK_KP_F1) {
15:
16:       /* we got a 'help' KeyPress, activate the selection */
17:       help_widget = trackingEvent( toplevel, cursor, True, &e );
18:
19:       XUndefineCursor( ui_display, XtWindow(toplevel));
20:
21:       /* a specific widget was selected, offer help */
22:       activate_help( help_widget );
23:     }
24:   }
25: }
```

Two critical areas are addressed by the `processKeyEvent` function. The first entails deciphering the key pressed

```
11:     keysym = XKeycodeToKeysym(XtDisplay(GxDrawArea),
12:                                 xe->xkey.keycode, 0);
```

and the second is waiting for the next ButtonPress event to determine the widget for which context-sensitive help is being requested:

```
      help_widget = trackingEvent( toplevel, cursor, True, &e );
```

The KeyPress event structure contains a field xe->xkey.keycode, which holds, for the purpose of supporting multiple language definitions, the encoded key value for the button pressed. The value of the encoded key is converted to a KeySym for comparison by the XKeycodeToKeysum Xlib function.

The trackingEvent function left for your definition must return the widget receiving the very next button press event that signals the user's desire for help on the entity.

Using the widget's explicit path within the application's instance hierarchy, as discussed in the next section, a correlation is made to the help text specific for this widget.

Widget Paths

Every widget has an explicit place in the hierarchy of all widgets contained in the application. This placement in the hierarchy is known as the widget's *path*. If the programmer chooses unique widget names (as assigned by the first parameter of the XtVaCreateManagedWidget function), the path will uniquely identify the widget.

The widget's instance name preceded by the instance name of its parent forms a portion of the widget's path. If you continually prepend the name of the parent widget's parent (and so forth), an explicit widget path for the initial widget is determined.

The getWidgetPath and getWidgetPathComponents functions defined in Listing 29.3 demonstrate how to find a widget path for the widget returned by trackingEvent seen earlier.

Listing 29.3 The getWidgetPathComponents Function

```
static void getWidgetPathComponents( Widget parent,
                     Widget w, char **path, int *len )
{
  char *wname;

  if( parent ) {
    /* throw away the toplevel */
    if( !XtParent(parent) ) return;

    getWidgetPathComponents( XtParent(parent), w, path, len );

    /* extract the name of the widget */
    wname = XtName( parent );

    /*
     * grow if necessary
     */
    if( strlen(wname) + 1 >= *len ) {
      /*
       * find the current end position
       */
      int endPos = strlen( *path ) - 1;

      (*len) += 512;
      *path = (char *)realloc( *path, *len);

      /*
       * zero out what has just been alloc'd
       */
      memset( *path + endPos, 0, (*len) - endPos );
    }

    /*
     * store the name of this widget and a '.'
     * if there are more components to follow
     */
    strcat( *path, wname );
    if( parent != w )
      strcat( *path, ".\0" );
  }
}

/*
** getWidgetPath
**
** follow the widget up its tree (hierarchy) assembling names
*/
static char *getWidgetPath( Widget w )
{
  static char *wpath = NULL;
  static int  wpathLen = 0;

  /* get a starting point for our buffer */
  if( wpath == NULL || wpathLen == 0 ) {
    wpathLen = 2048; /* hopefully it won't grow */
    wpath = (char *)malloc( wpathLen );
  }

  /* clear it out */
  memset( wpath, 0, wpathLen );
  getWidgetPathComponents( w, w, &wpath, &wpathLen );

  return wpath;
}
```

Invoking `getWidgetPath`

```
48: static char *getWidgetPath( Widget w )
```

for the widget returned from `trackingEvent` constructs a widget path to uniquely identify it within the application.

The `getWidgetPath` begins by creating sufficient memory space to store the path

```
56:     wpath = (char *)malloc( wpathLen );
```

which it initializes

```
60:   memset( wpath, 0, wpathLen );
```

and then begins the recursive process of extracting instance names for the widget's ancestors:

```
61:   getWidgetPathComponents( w, w, &wpath, &wpathLen );
```

The `getWidgetPathComponents` function continually invokes itself, passing the parent of the current parent:

```
10:     getWidgetPathComponents( XtParent(parent), w, path, len );
```

When the top of the hierarchy is found the recursion ceases

```
8:     if( !XtParent(parent) ) return;
```

and the names are assembled in the memory set aside for its use:

```
37:     strcat( *path, wname );
```

This widget path is related to either a help text string or the name of an HTML file that is displayed to satisfy the user's query for help, as described in the next section.

Relating Widgets to Text

With a complete path formed for the selected widget, it is possible to search an external configuration file that you will define to relate the widgets of your application to the help file describing each entity of the application.

A sample structure for the entries in the configuration file is provided in Listing 29.4.

Listing 29.4 Sample Help Configuration File Entry

```
static char *help_entry = "<\n\
<"filename.html#OPT_TAG\">\n\
<widgetpath>\n\
<Title>\n\
>\n";
```

The purpose of the configuration file entries is to associate each widget path of the application with help information.

This information can either be a string displayed in a help dialog created by the application, or as illustrated in Listing 29.4, an HTML file that the application uses to invoke an external browser to display help for the item selected by the user.

Next Steps

This chapter did not fully implement the context-sensitive help system but formed a to-do list outlining the key elements required to accomplish the task.

Sufficient information and structure is provided for the reader to work independently solving the proposed context-sensitive help system..

Part VIII

Appendixes

Appendix A

Command Shells and Scripting

The UNIX command shell is adequately named because what is visible is only the outside. Internally, there is much more to a command shell than is immediately understood.

UNIX is unique in letting users choose their command shell. Most operating systems (DOS, VMS, and so forth) have the command interpreter built in, giving users no alternative.

Under UNIX, the command interpreter is not a part of the operating system but rather a wrapper around it.

This appendix provides an overview of the command shells commonly available under many UNIX systems. Following an overview of shells, the focus is on writing shell scripts using the Bourne shell.

UNIX Command Shells

The Bourne shell (`sh`) is the standard among the UNIX shells because it was the original command shell provided by the Computer Research Group of Bell Labs.

The Bourne shell, however, lacks a significant feature known as job control, making it less than ideal for interactive use. Typically, the Bourne shell is used for scripting, independent of the shell assigned for interactive use.

Note

Every user is assigned a default shell in the last field of the `/etc/passwd` file for the entry defining their account. This is the shell that executes automatically when the user's login process completes.

Knowing the different command shells enables users to execute and employ other shells that are not their default.

Optionally, some versions of UNIX enable users to change their default shell through use of the `passwd` command by specifying the `-s` flag. Read the man page for the `passwd` command available under the version of UNIX you are running to determine whether this option is available; otherwise, a request to your System Administrator might be necessary to change your default shell.

EXCURSION
Controlling Jobs from Within a Command Shell

The job control feature enables processes to be moved from foreground to background and vice versa without closing and restarting the program.

A foreground process running in your command shell does not enable you to continue to issue commands until the process has finished. A background process (executed with a trailing ampersand) enables use of the command shell by the user for issuing commands and navigating the system to continue.

A shell that supports job control will enable a foreground process, meaning a process executed without a trailing ampersand (&), to be suspended by holding down the Control key and pressing the letter Z in the window where the program is running. Further, the process can be continued by use of the `fg` command to return it to the foreground or the `bg` command to return it to the background.

The command `jobs` is used to list the processes that a shell controls.

Each process displayed in the list is preceded with a number assigned sequentially as consecutive commands are executed. This number is referred to as a *job number,* and when prefaced with a percent sign (%) can be used in place of the process's ID obtained with the `ps` (process status) command.

Command shells commonly assigned as the interactive shell employed by users include the Korn shell (`ksh`), C shell (`csh`) and TC shell (`tcsh`).

how tōō prō nouns′ it

The C Shell is pronounced as if it were written *sea shell* and the TC shell as if it were written *tea sea shell*.

The *C shell* first introduced job control. It is a commonly assigned interactive shell but a poor choice for scripting because the method of implementing functions is non-standard between implementations of the shell and because nested `if` statements often do not work as expected.

The Korn shell is an extension of the Bourne shell and made significant improvements to the Bourne shell's interactive friendliness. As a derivative of the Bourne shell, the Korn shell is well suited for shell programming.

The Korn shell is proprietary to AT&T but is available as the Bourne Again shell (`bash`) on non-AT&T versions of UNIX.

Command Shell Environment

The command shell you employ dictates certain differences for configuring and working in the UNIX system. For instance, the syntax for setting variables in the shell environment and making them available between sessions varies depending on the shell used.

Shell Variables

The method of defining variables within a command shell varies based on the syntax understood by the shell. The following sections show the differing ways of defining variables for common command shells.

Bourne Shell Variables

The syntax to set a variable in the environment of the Bourne shell or one of its variants adheres to the following form:

```
VAR=value
```

By convention, an environment variable is fully capitalized. This aids in distinguishing it from a shell or system command.

Specifying an environment variable on the shell command line defines the variable only for the current session and ensures visibility only for the current shell.

To make the variable visible to sub-shells, the internal shell command `export` is used.

```
export VAR
```

A *sub-shell* is a shell executed from within another shell.

Note

An *internal command* refers to the fact that only the shell will interpret the command. Other shells might not understand it, and it will not physically reside on the UNIX system.

To read about an internal shell command, you will have to display the man page for the shell to which it is internal. An example of this is the `export` command, introduced for making variables available to subshells.

The `export` command can be combined with the variable declaration and value assignment.

```
export VAR=value
```

C Shell Variables

Setting variables in the C shell or one of its derivatives requires the internal shell command `setenv`.

```
setenv VAR value
```

Note

Syntactically, it is incorrect to use an equal sign (=) with the C shell variable declaration because the equal sign is unique to the Bourne shell. In other words, attempting the Korn shell syntax

```
    VAR=value
```

generates a `command unknown` error in the C Shell.

Unlike the Bourne shell, the C shell enables a variable to unset by use of the `unsetenv` internal shell command.

```
unsetenv VAR
```

To display all the variables set within an environment, use the `env` command. A specific variable's value can be displayed by either echoing its value using the syntax

```
echo $VAR
```

or using the following `printenv` internal shell command:

```
printenv VAR
```

Aliases

Most shells enable UNIX commands to have an alias defined in the shell environment to simplify the command's use.

For instance, using C shell syntax, the following alias can be defined for performing long directory listings:

```
alias ll ls -l
```

In this example, the `alias` internal shell command defines the alias `ll` to be the `ls` command with the `-l` flag. After executing the `alias` command, the `ll` alias is available as if it were a valid UNIX or shell command.

Variables and aliases defined on the command line are destroyed when the shell exits unless they are placed in an *environment file* read by the shell when it executes.

Environment Files

Command shells available under UNIX search for and interpret the contents of an environment file when it executes. This environment file enables a user to make variables and aliases available between user sessions.

Which file to create or modify within your user environment depends upon the shell being used.

Bourne Shell Environment Files

The Bourne shell (and derivatives) expects the environment configuration file to be placed in the user's home directory and given the name

```
$HOME/.profile
```

Note

Notice the dot (.) preceding the filename. A file preceded by a dot is called a hidden file because this makes the file invisible to the directory listing obtained by the `ls` command unless the `-a` (all) flag is used.

The contents of the `.profile` file can be any valid UNIX or internal shell command, one per line or separated by a semicolon (;), and entered exactly as if typed on the command line.

Listing A.1 shows a sample `.profile` file for initializing an environment when using the Bourne command shell or derivative.

Listing A.1 Sample `.profile` File

```
# Sample .profile file
PATH="$PATH:/usr/X11R6/bin"
PS1="[\u@\h ]\\$ "

USER=`id -un`
```

continues

Listing A.1 Continued

```
6:      LOGNAME=$USER
7:      MAIL="/var/spool/mail/$USER"
8:      HOSTNAME=`/bin/hostname`
9:      HISTSIZE=1000
10:     HISTFILESIZE=1000
11:
12:     export PATH PS1 USER LOGNAME MAIL \
13:             HOSTNAME HISTSIZE HISTFILESIZE INPUTRC
```

Understanding Listing A.1 only requires that you apply many of the things you learned in previous chapters of this book.

Starting after the comment of line 1, consider

```
2:     PATH="$PATH:/usr/X11R6/bin"
```

which sets an environment variable called `PATH`. All command shells employ a `PATH` variable for identifying where the shell should look for UNIX commands. A UNIX command not contained in one of the elements of the `PATH` variable will be reported as `command not found` unless an explicit path is specified.

As this setting is in the `.profile` file, the value is set every time the Bourne shell executes.

Following the assignment of the `PATH` variable, the variable called `PS1` is set:

```
3:     PS1="[\u@\h ]\\$ "
```

The PS1 variable is a secondary or command continuation prompt. When the command shell is told that a command will extend to a second line through use of the line continuation character backslash (\) followed by return, the value of `PS1` is used as the secondary prompt.

Every command shell will employ the PS1 variable for this purpose; however, the syntax understood by the shell when interpreting this variable depends upon the shell being used.

The syntax demonstrated in Listing A.1 for the Bourne Again Shell (`bash`) will yield the prompt

```
[username@hostname] $
```

because the token `\u` specifies the username, `\h` the hostname and `\\$` simply '$'.

The next few lines of Listing A.1 set several useful environment variables using standard UNIX commands and variables after they are defined.

```
USER=`id -un`
LOGNAME=$USER
MAIL="/var/spool/mail/$USER"
HOSTNAME=`/bin/hostname`
```

Pay close attention to the syntax for calling a UNIX command from within a shell script and employing the result. The shell must be told to evaluate the command before performing the assignment. This is accomplished by encasing the command between tick marks (`).

The `HISTSIZE` and `HISTFILESIZE` variables determine the number of commands remembered by the shell during a session and how many are retained in the `.history` file.

```
HISTSIZE=1000
HISTFILESIZE=1000
```

Shell command history enables the user to repeat commands by either using the up arrow key to have previously executed commands repeated at the command prompt or by using the `history` command to see and select commands from a list.

EXCURSION

Repeating Commands from the Shell History

To repeat a command from the `history` list, preface the sequential number assigned every command with an exclamation point on the command line.

```
bash[100]: !23
```

would repeat the 23rd command as it appears in the `history` list.

Optionally, you can use the exclamation point followed by the first few letters of the command.

```
bash[101]: !m
```

would repeat the last command executed that started with the letter `m`.

```
bash[102]: !ps
```

would repeat the last `ps` command.

The last two lines of Listing A.1

```
export PATH PS1 USER LOGNAME MAIL \
       HOSTNAME HISTSIZE HISTFILESIZE INPUTRC
```

export the variables set to ensure that they are visible to any sub-shells the current shell might parent.

C Shell Environment Files

The file sought and loaded by C shell and its derivatives is the `.cshrc` file. A sample is shown in Listing A.2.

Listing A.2 Sample `.cshrc` File

```
1:     # Sample cshrc
2:
3:     if ($?PATH) then
4:         setenv PATH "${PATH}:/usr/X11R6/bin"
5:     else
6:         setenv PATH "/bin:/usr/bin:/usr/local/bin:/usr/X11R6/bin"
7:     endif
8:
9:     if ($?prompt) then
10:      if ($?tcsh) then
11:        set prompt='[%n@%m %c]$ '
12:      else
13:        set prompt=\[`id -nu`@`hostname -s`\]\$\
14:      endif
15:    endif
16:
17:    setenv HOSTNAME `/bin/hostname`
18:    set history=1000
```

In the sample `.cshrc` shown in Listing A.2, a test is conducted following the comment to see whether the `PATH` variable is already set in the environment.

```
3:    if ($?PATH) then
4:        setenv PATH "${PATH}:/usr/X11R6/bin"
5:    else
6:        setenv PATH "/bin:/usr/bin:/usr/local/bin:/usr/X11R6/bin"
7:    endif
```

By now, the syntax `$`*`VAR`* should be understood as the method for accessing the value of a variable.

The syntax shown in the sample `.cshrc` introduces the question mark in conjunction with the variable access as the means for testing whether the variable value is of non-zero length to indicate whether a value is set.

The sample either appends a value to an existing value to ensure the X client commands are seen by the shell

```
4:        setenv PATH "${PATH}:/usr/X11R6/bin"
```

or performs an initial assignment

```
6:        setenv PATH "/bin:/usr/bin:/usr/local/bin:/usr/X11R6/bin"
```

The next section of Listing A.2 determines whether the `prompt` value has been set. If not, based on which shell is being used, this section sets it accordingly:

```
if ($?prompt) then
  if ($?tcsh) then
    set prompt='[%n@%m %c]$ '
  else
    set prompt=\[`id -nu`@`hostname -s`\]\$\
  endif
endif
```

The test for which shell is currently being executed is necessary because the syntax understood by the TC Shell differs from the syntax understood by the C shell for specifying the `prompt` value.

The final step is the setting of the variable `HOSTNAME`

```
setenv HOSTNAME `/bin/hostname`
```

and the setting of the size of the shell history buffer

```
set history=1000
```

The following section describes how to write shell scripts using the Bourne shell.

Scripting with the Bourne Shell

Note

Although targeted for the Bourne shell, this section applies to the Bourne shell enhancement *Korn shell* and the Korn shell clone *Borne Again shell*.

Shell scripting gives a UNIX software developer the ability to automate simple tasks in a portable and extensible manner.

Scripts are easy to create and modify and are immediately available.

To begin a shell script, the first line *must* specify the UNIX system that is the intended shell for interpreting the script. This designator looks very much like a comment, but is read and used by the system.

The shell designator uses the tokens `#!` followed by the command for the shell, including an explicit path.

```
#!/bin/sh
```

After specifying the interpreter, there are no other requirements beyond obeying the syntax for the commands and conventions used in the body of the shell.

Shell Variables

Declaring variables in shell script follows syntax that you have already seen in the `.profile` read by the Bourne shell when it executes.

```
VAR=value
```

A syntactical requirement is that no spaces exist between the components of the variable definition *VAR*, =, and *value*. If a space is present, the Bourne shell will perceive that an attempt is being made to execute a command and attempt to invoke VAR leading to a statement of the sort

```
VAR: command not found
```

A default value can be specified when using a variable by encasing the variable in braces and using a dash (-) separator between the variable name and the default value

```
${VAR-default}
```

Quoting Variables

When learning shell programming, you often won't know when to quote, what to quote, and which quotes to use.

Single forward quotes protect against all processing when spaces are used during variable assignments, as in the following example:

```
VAR='value1 value2'
```

Without the use of single quotes when a variable contains nested spaces, the shell will attempt to invoke *VAR* as a command.

Double quotes enable variable substitutions, with the result being a single argument, shown in the following example:

```
VAR='value1 value2'
VAR2="$VAR"
```

By convention, and to afford maximum portability, double quotes should be used consistently around variables, especially when the script itself does not control the variable's value.

Performing Tests

Performing tests is a crucial part of creating useful programs, as was discussed when reviewing programming conventions in Chapter 2, "Programming Constructs."

Tests in the Bourne shell are performed with either the `if` command or by simply nesting the test expression in square braces ([ ]).

The `if` statement in Bourne shell implies its own body; therefore, start and end body markers are not required. Instead, the `if` must be closed with the `fi` keyword.

Test expressions understood by the Bourne shell depend on the variable types being tested. Unlike C, Bourne test expressions yield zero (0) if the condition is true.

Table A.1 shows the test expressions understood by Bourne shell and examples of their use.

Table A.1 Bourne Shell Tests

Data Type	*Test*	*Description*	*Example*
String	`=`	Determines whether two strings are equal	`if [ "$var1" = "$var2" ]; then echo "Match found" fi`
	`!=`	Determines whether two strings are not equal	`if [ "$var1" != "$var2" ]; then echo "No Match found" fi`
Numeric	`-eq`	Determines whether two numbers are equal	`if [ $var1 -eq $var2 ]; then echo "Numbers are equal" fi`
	`-nq`	Sees whether two numbers are not equal	`if [ $var1 -nq $var2 ]; then echo "Nums not equal" fi`
	`-lt`	Sees whether the first number is less than the second	`[ $var1 -lt $var2 ]; then echo "$var1 less than" fi`
	`-gt`	Determines whether the first number is greater than a second	`if [ $var1 -gt $var2 ]; then echo "$var2 greater than" fi`
	`-ge`	Determines whether the first number is greater than or equal to a second	`fi [ $var1 -ge $var2 ]; then echo "$var1 greater or equal" fi`
Files	`-f`	Tests for existence of a file	`if [ -f "$filename" ]; then echo "File exists" fi`
	`-d`	Tests for existence of a directory	`if [ -d "$dirName" ]; then echo "Dir exists" fi`
	`-r`	Tests whether a file is readable	`if [ -r "$filename" ]; then echo "Can read" fi`
	`-w`	Tests whether a file is writeable	`if [ -w "$filename" ]; then echo "Can write" fi`
	`-x`	Tests whether a file is executable	`if [ -x "$filename" ]; then echo "Can exec" fi`
Boolean	`!`	Negates a test	`if [ ! -f "$filename" ]; then echo "$filename does not exist" fi`

continues

Table A.1 continued

Data Type	Test	Description	Example
	`-a`	Combines test with the Boolean AND function	`if[ -f "$filename" -a -w "$filename" ]; then # do something else # do something fi`
	`-o`	Combines test with the Boolean OR function	`if[ ! -f "$filename" -o -w "$filename" ]; then exit 1 else # do something fi`

Note

Notice the syntax for specifying an `else` condition to a test as demonstrated in the last line of Table A.1.

Similar to the C `switch` statement, the Bourne shell provides for establishing actions for multiple conditions using the `case` statement.

Case Tables

The syntax for performing a case is

```
case "$file" in
*.c) echo "$file is a C source file"
     ;;
*.h) echo "$file is a C header file"
     ;;
.*)  echo "$file is a hidden file"
     ;;
*)   echo "$file is an unknown file type"
     ;;
```

A feature of the Bourne shell `case` statement is the capability to use wildcards in the conditions being tested. These fields, as shown in the previous example, are terminated with a double semicolon (`;;`) following the condition body.

The syntax for conditions in a `case` statement can optionally be ORed as well

```
case "$input" in
Q*|q*) echo "Exiting..."
       exit 1
     ;;
*) echo "Input not recognized ($input)"
     ;;
```

The capability to perform looping is also necessary, as it is with any programming language.

Looping

Two loop constructs, the `while` loop and the `for` loop, exist in the Bourne shell.

The `while` Loop

The following is an example of the `while` loop:

```
cnt=0
while [ $cnt -lt 10 ]
do
  echo $cnt
  cnt=`expr "$cnt" + 1`
done
```

The sample `while` loop simply counts to 10, printing the value of the variable `cnt` for each iteration. The use of `do` and `done` statements for marking the code body of the loop is important to all loops understood by the Bourne shell.

EXCURSION

Use the `expr` Command to Perform Mathematical Equations in a Shell Script

The command `expr` is a standard UNIX command used to manipulate variables in a variety of ways. It is a useful command. Review its man page for a full description.

Because the Bourne shell has no mathematical capability, you must use the `expr` command for mathematical operations. Give each component of the expression to `expr` as a separate argument.

In the `while` loop example, `expr` is used to add 1 to the variable `cnt`. Notice the tick marks instructing the shell to interpret the UNIX call before making the assignment. In this way, the return value of the evaluated `expr` call is used as the new value of `cnt`.

The next loop to consider is the Bourne shell `for` loop.

The `for` Loop

Consider the following example of the `for` loop:

```
count=0
for file in `ls /dev/tty*`
do
  count=`expr "$count" + 1'
done
echo "$count TTYs found"
```

The `for` loop expands the `ls` command and makes the result a token, assigning one token per loop iteration to the variable `file`.

One shortcoming of the Bourne shell `for` loop is that it will always perform one iteration of the loop, even when the evaluated statement yields no tokens.

Many other internal commands and features exist in the Bourne shell script. The preceding sections provide an excellent starting point for writing shell scripts.

Writing a Script with Function Calls

Using your favorite editor, open a file and add the necessary first line to instruct the shell which shell interpreter to use.

Then begin entering shell or UNIX commands to accomplish the purpose of the script.

Note

After saving the file, be sure to add execute permission using the command

```
chmod +x filename
```

Listing A.3 shows a sample shell script for printing a menu and responding to user input.

Listing A.3 Sample Script `menu.sh`

```
#!/bin/sh

print_menu() {
  echo
  echo "    Sample Menu"
  echo "    -----------"
  echo "1)  Print a file"
  echo "2)  List a file"
  echo "3)  Remove a file"
  echo "4)  Exit"
  echo
  echo -n "Selection: "

};

#
# Shell execution starts here
#
forever=0
while [ $forever -lt 1 ]
do
  print_menu

  #
  # get user input
  read ans

  case "$ans" in
```

```
29:      1) echo -n "Enter filename: "
30:        read filename
31:        if [ "$?filename" ]; then
32:          echo "Printing file ($filename) to default printer..."
33:           lp "$filename"
34:        else
35:          echo "No file specified"
36:        fi
37:        sleep 2 # pause so message is read
38:        ;; # end condition
39:      2) echo -n "Enter file: "
40:        read filename
41:        if [ -f "$filename" ]; then
42:          ls -l "$filename"
43:        else
44:          echo "No file to list..."
45:        fi
46:        sleep 2
47:        ;;
48:      3) echo -n "Enter file to remove"
49:        read filename
50:        if [ -w "$filename" ]; then
51:          echo -n "Are you sure [yes/no]: "
52:          read ans
53:          if [ "$ans" = "yes" ]; then
54:           rm -f "$filename"
55:          fi
56:        else
57:          echo "You do not have permission to remove $filename"
58:        fi
59:        sleep 2
60:        ;;
61:      *) echo "Bye!"
62:        exit 0
63:        ;;
64:      esac
65:    done
66:    #
67:    # end of menu.sh
68:    #
```

Two new conventions appear in Listing A.3, specifically, the `print_menu` function definition and the commands `sleep`, `read`, and `exit`.

A function is defined in a Bourne shell by providing a name followed by open and close parentheses. Open and close curly braces are used to mark the beginning and ending of the function body.

The function is invoked by simply entering its name:

```
22:      print_menu
```

The commands `sleep`, `read`, and `exit` perform the function implied by their name.

The `sleep` Command

The `sleep` command causes the shell script to pause for the number of seconds specified as the parameter when the command is executed.

```
        sleep 2 # pause so message is read
```

The `read` Command

The `read` command fills a variable passed as a parameter with user input. Everything typed by the user until the Return key is pressed is assigned as the variable's value.

```
     read ans
```

The `exit` Command

The `exit` command ceases the shell script's execution and returns as the exit status the value specified as the command's parameter.

```
        exit 0
```

Unlike other languages, Bourne shell uses the exit value of zero (0) to indicate success.

This is useful because many errors can cause a shell script to exit. By returning a value greater than zero when there is an error, the value can be used to determine the failure's cause.

When an error occurs because of unexpected behavior of the script, it can be debugged.

Debugging Shell Scripts

The `-x` flag for placing the shell in debug mode is common to all UNIX shells.

Two ways to employ the flag are by either editing the first line of the shell script to include the `-x`

```
#!/bin/sh -x
```

or executing the script within another shell as in

```
sh -x menu.sh
```

Note

Running a shell script within a shell, as demonstrated in the previous example for enabling debug, works for executing a script for which you don't have execute permission.

When a shell script is in debug mode, it prints every line prior to execution. This enables you to see the value of variables as they are evaluated by the script.

Appendix B

Application Layout Code Listing

This appendix provides the complete source code listing from Chapter 6, "Components of an X Window Application."

The code is sufficient to create the initial layout of the Graphics Editor project. The intent is to

- Illustrate proper coding structure and C syntax
- Introduce the Graphics Editor application layout
- Show concise modular programming techniques
- Provide examples of X Toolkit Intrinsics programming

Figure B.1 shows the code structure as captured from my development environment.

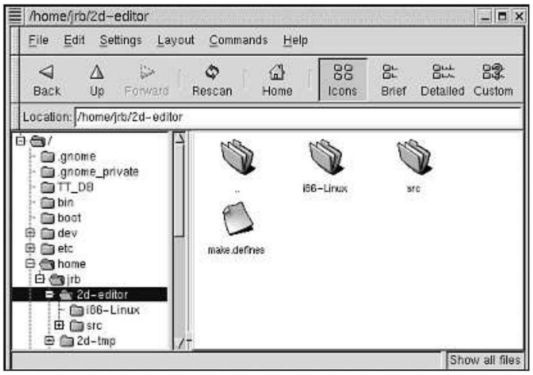


Figure B.1

Project directory structure.

The directory `2d-editor` is the root directory for the project; it contains a source directory called `src` and an object directory called `i86-Linux`.

These directories can be created using the commands discussed in Chapter 1, "UNIX for Developers."

`make.defines` File Contents

Listing B.1 shows the contents of the `make.defines` file, which should be placed in the `2d-editor` project root directory.

Listing B.1 `make.defines` File Contents

```
###
# 2d-gx make.defines
#
# This file should be included in each Make file used with 2d-gx
#
# Dependencies on the following environment variables:
#  TARGET - machine-os
#
# The syntax of this file is for use # with  'gmake' (GNU version of make)
#
# See the text for a discussion of this file and its syntax.
###
TARGET_SPARC_SUNOS    = sun4u-SunOS
TARGET_i86_SOLARIS    = i86pc-SunOS
TARGET_i86_LINUX      = i86-Linux

ifdef GxHOME
    GxSRCDIR = ${GxHOME}/src
else
    GxSRCDIR = ../src
endif

vpath %.h ${GxSRCDIR}/include
vpath %.c ${GxSRCDIR}

#
# Configure for Linux running on a PC
#
ifeq ($(TARGET),$(TARGET_i86_LINUX))
  X11INC    = -I/usr/include/X11
  X11LIB    = -L/usr/X11R6/lib -L/usr/lib
  INCS      = -I${GxSRCDIR}/include ${X11INC}

  CC        = gcc
  OPTS      = -ansi -Wall -g
endif

```

```
#
# Configure for Solaris running on a Sparc
#
ifeq ($(TARGET),$(TARGET_SPARC_SUNOS))
  X11INC     = -I/usr/openwin/include
  X11LIB     = -L/usr/openwin/lib
  INCS       = -I${GxSRCDIR}/include

  CC         = gcc
  OPTS       = -g -Wall -ansi
endif

#
# Configure for Solaris running on a PC
#
ifeq ($(TARGET),$(TARGET_i86_SOLARIS))
  X11INC     = -I/usr/openwin/include
  X11LIB     = -L/usr/openwin/lib
  INCS       = -I${GxSRCDIR}/include

  CC         = gcc
  OPTS       = -ansi -Wall -g
endif

#
# Force all Makefiles using this file to check the configuration
# of the environment before building the target
#
all: make-env-check make-target

#
# Check to environment variables need to build are set
#
make-env-check:
    ifndef TARGET
      @echo
      @echo "TARGET not defined!"
      @echo "Set environment variable TARGET to:"
      @echo "    sun4u-SunOS     or"
      @echo "    i86pc-SunOS     or"
      @echo "    i86-Linux"
      @echo
      @exit 1
    endif

clean:
    @rm -f *~ *.o $(PROGRAM)
#
# end of make.defines
#
```

As discussed in Chapter 1 where this file was introduced, the variable `TARGET` must be set in your environment for this portion of the `GNUmakefile` to work.

`GNUmakefile` File Contents

Listing B.2 shows the contents of the `GNUmakefile`, which should be placed in the `src` directory below the project root.

Listing B.2 `GNUmakefile` File Contents

```
#
# Makefile for 2d-gx
#
# See the text for a discussion of this file and its syntax
###

# Read in the system specific environment configuration
include ../make.defines

PROGRAM = 2d-gx

LIBS    = -lXaw -lXt -lX11

CFLAGS  = $(OPTS) $(INCS)

OBJS =    gxMain.o \
          gxGraphics.o \
          gxLine.o \
          gxText.o \
          gxArc.o \
          gxGx.o


make-target: $(PROGRAM)

$(PROGRAM): $(OBJS)
    @echo "Building $(PROGRAM) for $(TARGET)..."
    $(CC) -o $(PROGRAM) $(OBJS) $(X11LIB) $(LIBS)
    @echo "done"
#
# end of Makefile for 2d-gx
#
```

Note

It is necessary to create a *symbolic link* to this file in the object directory where the project will be built. See Chapter 1 for a discussion of symbolic links and how to create them.

`gxMain.c` File Contents

Listing B.3 shows the primary source file `gxMain.c`, which should be placed in the `src` directory.

Listing B.3 `gxMain.c` File Contents

```
/**
 ** 2D Graphical Editor (2d-gx)
 **
 ** gxMain.c
 */
#include <stdlib.h>
#include <stdio.h>

#include <X11/Intrinsic.h>   /* for creation routines */
#include <X11/StringDefs.h>  /* resource names        */
#include <X11/Xaw/Form.h>

#include "gxGraphics.h"
#include "gxProtos.h"

/*
 * Entry point for the application
 */
int main( int argc, char **argv )
{
  XtAppContext appContext;
  Widget       toplevel;
  Widget       form;
  Widget       canvas;

  toplevel = XtVaAppInitialize( &appContext, "2D Graphical Editor",
                                NULL, 0, &argc, argv, NULL,
                                NULL );

  form = XtVaCreateWidget( "topForm",
                            formWidgetClass, toplevel,
                            NULL );

  canvas = create_canvas( form );
  create_status( form, canvas );

  create_buttons( form );

  XtManageChild( canvas );
  XtManageChild( form );

  XtRealizeWidget( toplevel );
  XtAppMainLoop( appContext );

  exit(0); /* never reached as XtAppMainLoop is infinite */
}

/**
 ** end of gxMain.c
 */
```

gxGraphics.c File Contents

Listing B.4 shows the contents of the gxGraphics.c file, which should be placed in the src directory.

Listing B.4 gxGraphics.c File Contents

```
/**
 ** 2D Graphical Editor (2d-gx)
 **
 ** gxGraphics.c
 */
 #include <stdio.h>

 #include <X11/Intrinsic.h>
 #include <X11/StringDefs.h>
 #include <X11/Xaw/Form.h>
 #include <X11/Xaw/Command.h>
 #include <X11/Xaw/Box.h>
 #include <X11/cursorfont.h>

 #include "gxGraphics.h"
 #include "gxIcons.h"
 #include "gxBitmaps.h"

 static GxIconData gxDrawIcons[] = {
 { &line_icon, (void (*)(void))gx_line,
➥"Draw an elastic line..."       },
 { &pen_icon,  (void (*)(void))gx_pencil,
➥"Draw a freestyle line..."      },
 { &arc_icon,  (void (*)(void))gx_arc,
➥"Draw a circle..."              },
 { &box_icon,  (void (*)(void))gx_box,
➥"Draw a square or rectangle..." },
 { &arr_icon,  (void (*)(void))gx_arrow,
➥"Draw an arrow..."              },
 { &text_icon, (void (*)(void))gx_text,
➥"Draw dynamic text..."          },
 /*---------------------------------*/
 /* this list MUST be NULL terminated */
 /*---------------------------------*/
 { NULL },
 };

 /*
  * Create the region of the application where we will draw
  */
 Widget create_canvas( Widget parent )
 {
   GxDrawArea = XtVaCreateWidget( "drawingArea",
                                  formWidgetClass, parent,
                                  XtNbackground,
                                  WhitePixelOfScreen(XtScreen(parent)),
                                  XtNtop,           XawChainTop,
                                  XtNleft,          XawChainLeft,
                                  XtNbottom,        XawChainBottom,
                                  XtNright,         XawChainRight,
                                  XtNheight,        220,
                                  XtNwidth,         250,
                                  NULL );

   XtAddEventHandler( GxDrawArea, PointerMotionMask, False,
                     (XtEventHandler)drawAreaEventProc,
➥                  (XtPointer)NULL);
   XtAddEventHandler( GxDrawArea, ButtonPressMask, False,
                     (XtEventHandler)drawAreaEventProc,
➥                  (XtPointer)NULL);
   XtAddEventHandler( GxDrawArea, ButtonReleaseMask, False,
                     (XtEventHandler)drawAreaEventProc,
➥                  (XtPointer)NULL);

   return GxDrawArea;
 }

/*
 * create_status
 */
 void create_status( Widget parent, Widget fvert )
 {
   GxStatusBar =
➥  XtVaCreateManagedWidget( "statusBar",
                              labelWidgetClass, parent,
                              XtNtop,         XawChainBottom,
                              XtNleft,        XawChainLeft,
                              XtNbottom,      XawChainBottom,
                              XtNright,       XawChainRight,
                              XtNfromVert,    fvert,
                              XtNborderWidth, 0,
                              NULL );
   setStatus( "2D-GX (c)Starry Knight Software - Ready..." );
 }

 /*
  * statusProc
  */
 void statusProc( Widget w,
➥               XtPointer msg, XEvent *xe, Boolean flag )
 {
   if( msg == NULL )
     setStatus( "\0" );
   else
     setStatus( msg );
 }

```

continues

Listing B.4 continued

```
/*
 *  create_icons
 */
void create_icons( Widget parent, GxIconData *iconData,
                   void (*callback)( Widget,
                                     XtPointer, XtPointer ))
{
  Widget btn;
  Pixmap pix;

  while( iconData->info != NULL ) {
    if( iconData->info->bits != NULL ) {
      pix = create_pixmap( parent, iconData->info );

      btn =
➥    XtVaCreateManagedWidget( "",
                                 commandWidgetClass, parent,
                                 XtNwidth,  iconData->info->w + 1,
                                 XtNheight, iconData->info->h + 1,
                                 XtNbackgroundPixmap,     pix,
                                 XtNhighlightThickness,   1,
                                 NULL );

      XtAddEventHandler( btn, EnterWindowMask, False,
                        (XtEventHandler)statusProc,
➥                   (XtPointer)iconData->mesg);
      XtAddEventHandler( btn, LeaveWindowMask, False,
                        (XtEventHandler)statusProc,
➥                   (XtPointer)NULL );

      XtAddCallback( btn, XtNcallback, callback,
➥               (XtPointer)iconData->func );
    }
    /*
     * go to the next element
     */
    iconData++;
  }
}

/*
 * Create a panel of buttons that will
➥ * allow control of the application
 */
void create_buttons( Widget parent )
{
  Widget butnPanel, exitB;

  /*
   * create a panel for the drawing icons
   */
  butnPanel = XtVaCreateWidget( "drawButnPanel",
                                 boxWidgetClass, parent,
                                 XtNtop,           XawChainTop,
                                 XtNright,         XawChainRight,
                                 XtNbottom,        XawChainTop,
                                 XtNleft,          XawChainRight,
                                 XtNhorizDistance, 10,
                                 XtNfromHoriz,     GxDrawArea,
                                 XtNhSpace,        1,
                                 XtNvSpace,        1,
                                 NULL );

  create_icons( butnPanel, gxDrawIcons, draw_manager );
  XtManageChild( butnPanel );

  exitB = XtVaCreateManagedWidget( " Exit ",
                                   commandWidgetClass, parent,
                                   XtNtop,       XawChainBottom,
                                   XtNbottom,    XawChainBottom,
                                   XtNleft,      XawChainRight,
                                   XtNright,     XawChainRight,
                                   XtNfromVert,  butnPanel,
                                   XtNfromHoriz, GxStatusBar,
                                   NULL );

  XtAddCallback( exitB, XtNcallback, gx_exit, NULL );
}

/*
 * create_pixmap
 */
Pixmap create_pixmap( Widget w, XbmDataPtr data )
{
  return( XCreatePixmapFromBitmapData( XtDisplay(w),
                                  DefaultRootWindow(XtDisplay(w)),
                                  (char *)data->bits,
                                  data->w, data->h,
                                  BlackPixelOfScreen(XtScreen(w)),
                                  WhitePixelOfScreen(XtScreen(w)),
                                  DefaultDepthOfScreen(XtScreen(w))));
}

/**
 ** end of gxGraphics.c
 */
```

`gxGx.c` File Contents

Listing B.5 shows the contents of the file `gxGx.c`, which should be placed in the `src` directory.

Listing B.5 `gxGx.c` File Contents

```
/**
 ** 2D Graphical Editor (2d-gx)
 **
 ** gxGx.c
 */
#include <stdio.h>

#include "gxGraphics.h"
#include "gxProtos.h"

#include <X11/Xaw/Label.h>

static void (*draw_mgr_func)( XEvent * ) = NULL;

/*
 * gx_exit
 */
void gx_exit( Widget w, XtPointer cd, XtPointer cbs )
{
  exit(0);
}

/*
 * setStatus
 */
void setStatus( char *message )
{
  XtVaSetValues( GxStatusBar, XtNlabel, message, NULL );
}

/*
 * draw_manager
 */
void draw_manager( Widget w, XtPointer cd, XtPointer cbs )
{
  void (*draw_func)( XEvent * ) = (void (*)(XEvent *))cd;

  if( draw_func != NULL ) (*draw_func)( NULL );
  draw_mgr_func = draw_func;
}

/*
 * drawAreaEventProc
 */
void drawAreaEventProc( Widget w, XtPointer cd,
                      XEvent *event, Boolean flag )
{
  if( draw_mgr_func != NULL ) (*draw_mgr_func)( event );
}

```

```
/**
 ** end of gxGx.c
 */
```

gxArc.c File Contents

Listing B.6 shows the contents of the file gxArc.c, which should be placed in the src directory.

Listing B.6 gxArc.c File Contents

```
/**
 ** 2D Graphical Editor (2d-gx)
 **
 ** gxArc.c
 */
#include <stdio.h>

void gx_arc( void )
{
  printf( "draw an arc...\n" );
}

/**
 ** end of gxArc.c
 */
```

Clearly, the files representing the various objects in the Graphics Editor project are simply stubs. Beginning with Chapter 13, "Application Structure," the text will advance these files.

gxLine.c File Contents

Listing B.7 shows the contents of the gxLine.c file, which should be placed in the src directory.

Listing B.7 gxLine.c File Contents

```
/**
 ** 2D Graphical Editor (2d-gx)
 **
 ** gxLine.c
 */
#include <stdio.h>
#include <X11/Intrinsic.h>

void gx_line( XEvent *event )
{
  printf( "draw a line...\n" );
}
```

continues

Listing B.7 continued

```

void gx_pencil( XEvent *event )
{
  printf( "draw freestlye\n" );
}

void gx_arrow( XEvent *event )
{
  printf( "draw an arrow\n" );
}

void gx_box( XEvent *event )
{
  printf( "draw a box\n" );
}

/**
 ** end of gxLine.c
 */
```

`gxText.c` File Contents

Listing B.8 shows the file contents of `gxText.c`. It, too, should be placed in the `src` directory.

Listing B.8 `gxText.c` File Contents

```
/**
 ** 2D Graphical Editor (2d-gx)
 **
 ** gxText.c
 */
#include <stdio.h>

void gx_text( void )
{
  printf( "draw text...\n" );
}

/**
 ** end of gxText.c
 */
```

`gxGraphics.h` File Contents

The `gxGraphics.h` file is the first header file introduced in the application layout. Its placement will require creating a new directory under `src` called `include`.

The contents of the file `gxGraphics.h` are shown in Listing B.9.

Listing B.9 `gxGraphics.h` File Contents

```
/**
 ** 2D Graphical Editor (2d-gx)
 **
 ** gxGraphics.h
 */
#include <X11/Intrinsic.h>

#ifndef _GX_GRAPHICS_H_INC_
#define _GX_GRAPHICS_H_INC_

#endif /* _GX_GRAPHICS_H_INC_ */

#ifndef GLOBAL
#define GLOBAL
#else
#undef  GLOBAL
#define GLOBAL extern
#endif

GLOBAL Widget GxStatusBar;
GLOBAL Widget GxDrawArea;

/**
 ** end of gxGraphics.h
 */
```

`gxIcons.h` File Contents

Listing B.10 shows the contents of the header file `gxIcons.h`. The placement of this file is in the `include` directory in `src`.

Listing B.10 `gxIcons.h` File Contents

```
/**
 ** 2D Graphical Editor (2d-gx)
 **
 ** gxIcons.h
 */
#ifndef _GX_ICONS_H_INC_
q#define _GX_ICONS_H_INC_

#include "gxProtos.h"

/*
 * Storage for pertinent XBM data
 */
typedef  struct _xbm_data {
  unsigned char *bits;
  int           w, h;
} XbmData, *XbmDataPtr;
```

continues

Listing B.10 continued

```

/*
 * IconData necessary to create icon
 */
typedef struct _gx_icon_data {
  XbmDataPtr info;

  void (*func)(void);
  char *mesg;
} GxIconData, *GxIconDataPtr;

#define icon_static( name, bits, width, height ) \
 static XbmData name = { bits, width, height }



/* prototypes */
extern void   create_icons  ( Widget, GxIconDataPtr,
                              void (*)(Widget,
                                     XtPointer, XtPointer ));
extern Pixmap create_pixmap ( Widget, XbmDataPtr    );

#endif /* _GX_ICONS_H_INC_ */

/**
 ** end of gxIcons.h
 */
```

gxBitmaps.h File Contents

Listing B.11 shows the file contents of the header file `gxBitmaps.h`, which should be placed in the `include` directory under `src`.

Listing B.11 `gxBitmaps.h` File Contents

```
/**
 ** 2D Graphical Editor (2d-gx)
 **
 ** gxBitmaps.h
 */
#ifndef _GX_BITMAPS_H_INC_
#define _GX_BITMAPS_H_INC_

#include "gxIcons.h"

  /*
   * drawing icons
   */
  static unsigned char line_bits[] = {
     0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x20,0x00,0x04,0x00,0x00,0x30,0x00,
   0x0c,0x00,0x00,0x50,0x00,0x0c,0x00,0x00,0x48,0x00,0x14,
   0x00,0x00,0x88,0x00,0x14,0x00,0x00,0x84,0x00,0x14,0x00,
   0x00,0x04,0x01,0x22,0x00,0x00,0x02,0x01,0x22,0x00,0x00,
   0x02,0x02,0x22,0x00,0x00,0x01,0x02,0x42,0x00,0x00,0x01,
   0x04,0x42,0x00,0x80,0x00,0x04,0x42,0x00,0x80,0x00,0x02,
   0x01,0x00,0x40,0x00,0x01,0x01,0x00,0x40,0x80,0x00,0x01,
   0x00,0x20,0x40,0x00,0x01,0x00,0x20,0x20,0x00,0x01,0x00,
   0x00,0x10,0x00,0x01,0x00,0x00,0x08,0x80,0x00,0x00,0x00,
   0x30,0x40,0x00,0x00,0x00,0xc0,0x20,0x00,0x00,0x00,0x00,
   0x13,0x00,0x00,0x00,0x00,0x0c,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00};
icon_static( line_icon, line_bits, 36, 32 );

static unsigned char pencil_bits[] = {
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0xc0,0x00,0x00,0x00,0x00,0xe0,0x01,0x00,0x00,0x00,0xd0,
   0x03,0x00,0x00,0x00,0x88,0x03,0x00,0x00,0x00,0x14,0x01,0x00,
   0x00,0x00,0xa6,0x00,0x00,0x00,0x00,0x49,0x00,0x00,0x00,0x80,
   0x30,0x00,0x00,0x00,0x40,0x10,0x00,0x00,0x00,0x20,0x08,0x00,
   0x00,0x00,0x10,0x04,0x00,0x00,0x00,0x08,0x02,0x00,0x00,0x00,
   0x04,0x01,0x00,0x00,0x00,0x82,0x00,0x00,0x00,0x00,0x41,0x00,
   0x00,0x00,0x80,0x20,0x00,0x00,0x00,0x40,0x10,0x00,0x00,0x00,
   0xa0,0x08,0x00,0x00,0x00,0x10,0x05,0x00,0x00,0x00,0x10,0x02,
   0x00,0x00,0x00,0x30,0x01,0x00,0x00,0x28,0xf0,0x00,0x00,0x00,
   0x44,0x10,0x00,0x00,0x00,0x84,0x20,0x00,0x00,0x00,0x04,0x41,
   0x00,0x00,0x00,0x08,0x42,0x00,0x00,0x00,0x10,0x44,0x00,0x00,
   0x00,0x20,0x38,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00};
icon_static( pen_icon, pencil_bits, 36, 32 );

static unsigned char arc_bits[] = {
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0x3f,0x00,0x00,0x00,
   0x70,0xc0,0x01,0x00,0x00,0x0c,0x00,0x06,0x00,0x00,0x02,0x00,
   0x08,0x00,0x00,0x01,0x00,0x10,0x00,0x80,0x00,0x00,0x20,0x00,
   0x40,0x00,0x00,0x40,0x00,0x40,0x00,0x04,0x40,0x00,0x20,0x00,
   0x04,0x80,0x00,0x20,0x00,0x1f,0x80,0x00,0x20,0x00,0x04,0x80,
   0x00,0x40,0x00,0x04,0x40,0x00,0x40,0x00,0x00,0x40,0x00,0x80,
   0x00,0x00,0x20,0x00,0x00,0x01,0x00,0x10,0x00,0x00,0x02,0x00,
   0x08,0x00,0x00,0x0c,0x00,0x86,0x00,0x00,0x70,0xc0,0x81,0x00,
   0x00,0x80,0x3f,0xe0,0x03,0x00,0x00,0x00,0x80,0x00,0x00,0x00,
   0x00,0x80,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
   0x00,0x00,0x00,0x00};
icon_static( arc_icon, arc_bits, 36, 32 );

static unsigned char box_bits[] = {
   0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
```

continues

Listing B.11 continued

```
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,0x1f,
      0x00,0x80,0x00,0x00,0x10,0x00,0x80,0x00,0x00,0x10,0x00,
      0x80,0x00,0x00,0x10,0x00,0x80,0x00,0x00,0x10,0x00,0x80,
      0x00,0x00,0x10,0x00,0x80,0x00,0x00,0x10,0x00,0x80,0x00,
      0x00,0x10,0x00,0x80,0x00,0x00,0x10,0x00,0x80,0x00,0x00,
      0x10,0x00,0x80,0x00,0x00,0x10,0x00,0x80,0x00,0x00,0x10,
      0x00,0x80,0x00,0x00,0x10,0x00,0x80,0x00,0x00,0x10,0x00,
      0x80,0x00,0x00,0x10,0x00,0x80,0xff,0xff,0x1f,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00};
   icon_static( box_icon, box_bits, 36, 32 );

   static unsigned char arrow_bits[] = {
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01,0x00,
      0x00,0x00,0x80,0x02,0x00,0x00,0x00,0x40,0x04,0x00,0x00,0x00,
      0x20,0x08,0x00,0x00,0x00,0x10,0x10,0x00,0x00,0x00,0x08,0x20,
      0x00,0x00,0x00,0x04,0x40,0x00,0x00,0x00,0x02,0x80,0x00,0x00,
      0x00,0x01,0x00,0x01,0x00,0x80,0x00,0x00,0x02,0x00,0x40,0x00,
      0x00,0x04,0x00,0x20,0x00,0x00,0x08,0x00,0xe0,0x07,0xc0,0x0f,
      0x00,0x00,0x04,0x40,0x00,0x00,0x00,0x04,0x40,0x00,0x00,0x00,
      0x04,0x40,0x00,0x00,0x00,0x04,0x40,0x00,0x00,0x00,0x04,0x40,
      0x00,0x00,0x00,0x04,0x40,0x00,0x00,0x00,0x04,0x40,0x00,0x00,
      0x00,0x02,0x80,0x00,0x00,0xc0,0x01,0x00,0x07,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00};
   icon_static( arr_icon, arrow_bits, 36, 32 );

   static unsigned char text_bits[] = {
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x80,0xff,0xff,
      0x1f,0x00,0x80,0x83,0x1f,0x1c,0x00,0x80,0x01,0x0f,0x18,
      0x00,0x80,0x00,0x0f,0x10,0x00,0x00,0x00,0x0f,0x00,0x00,
      0x00,0x00,0x0f,0x00,0x00,0x00,0x00,0x0f,0x00,0x00,0x00,
      0x00,0x0f,0x00,0x00,0x00,0x00,0x0f,0x00,0x00,0x00,0x00,
      0x0f,0x00,0x00,0x00,0x00,0x0f,0x00,0x00,0x00,0x00,0x0f,
      0x00,0x00,0x00,0x00,0x0f,0x00,0x00,0x00,0x00,0x0f,0x00,
      0x00,0x00,0x00,0x0f,0x00,0x00,0x00,0x00,0x0f,0x00,0x00,
      0x00,0x00,0x0f,0x00,0x00,0x00,0x00,0x0f,0x00,0x00,0x00,
      0x80,0x1f,0x00,0x00,0x00,0xe0,0x7f,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
      0x00,0x00,0x00,0x00,0x00,0x00};
   icon_static( text_icon, text_bits, 36, 32 );
```

```

#endif /* _GX_BITMAPS_H_INC_ */

/**
 ** end of gxIcons.h
 */
```

gxProtos.h File Contents

Listing B.12 shows the file contents of `gxProtos.h`, which should be placed with the other header files in the `include` directory under `src`.

Listing B.12 gxProtos.h File Contents

```
/**
 ** 2D Graphical Editor (2d-gx)
 **
 ** gxProtos.h
 */
#include <X11/Intrinsic.h>
#include "gxIcons.h"

#ifndef EXTERN
#define EXTERN
#else
#undef  EXTERN
#define EXTERN extern
#endif

/*
 * Creations routines in gxGraphics.c
 */
EXTERN Widget create_canvas ( Widget );
EXTERN void   create_status ( Widget, Widget );
EXTERN void   create_buttons( Widget );
EXTERN void   drawAreaEventProc( Widget, XtPointer,
➥                                 XEvent *, Boolean );

/*
 * Drawing functions used in GxIcons.h
 */
EXTERN void gx_line( void );
EXTERN void gx_pencil( void );
EXTERN void gx_arc( void );
EXTERN void gx_box( void );
EXTERN void gx_arrow( void );
EXTERN void gx_text( void );

/*
 * Control functions used in GxIcons.h
 */
```

continues

Listing B.12 continued

```
EXTERN void gx_exit( Widget, XtPointer, XtPointer );

/*
 * Utilities in gxGx.c
 */
EXTERN void setStatus( char * );
EXTERN void draw_manager( Widget, XtPointer, XtPointer );
EXTERN void ctrl_manager( Widget, XtPointer, XtPointer );

/**
 ** end of gxProtos.h
 */
```

Appendix C

In this appendix

- *Script Simple Vector Font Set*
- *Triplex Bold Vector Font Set*
- *Triplex Bold Italic Vector Font Set*
- *The* `vector_chars.h` *Header File*

Additional Vector Font Sets and `vector_chars.h`

The following sections introduce several additional groupings of vector font sets that can be used in the Graphics Editor. This appendix also provides the definition of the contents of the `vector_chars.h` header file needed by the `Text` object introduced in Chapter 24, "Vector `Text` Object."

Script Simple Vector Font Set

Listing C.1 defines a simple italic-style vector font set. It can be used in lieu of the simplex font introduced in Chapter 24.

Listing C.1 The `script_simplex.h` Font Definition

```
#ifndef S_SIM_INC_H
#define S_SIM_INC_H

#include "vfonts/vector_chars.h"

XPoint **script_simplex[] = {
char699, char2764, char2778, char733, char2769, char2271,
char2768, char2767, char2771, char2772, char2773, char725,
char2761, char724, char710, char2770, char2750, char2751,
char2752, char2753, char2754, char2755, char2756, char2757,
char2758, char2759, char2762, char2763, char2241, char726,
char2242, char2765, char2273, char551, char552, char553,
char554, char555, char556, char557, char558, char559,
char560, char561, char562, char563, char564, char565,
char566, char567, char568, char569, char570, char571,
```

continues

Listing C.1 Continued

```
 char572, char573, char574, char575, char576, char2223,
 char804, char2224, char2262, char999, char2766, char651,
 char652, char653, char654, char655, char656, char657,
 char658, char659, char660, char661, char662, char663,
 char664, char665, char666, char667, char668, char669,
 char670, char671, char672, char673, char674, char675,
 char676, char2225, char723, char2226, char2246, char718,
};

int *script_simplex_p[] = {
 char_p699, char_p2764, char_p2778, char_p733, char_p2769,
 char_p2271, char_p2768, char_p2767, char_p2771, char_p2772,
 char_p2773, char_p725, char_p2761, char_p724, char_p710,
 char_p2770, char_p2750, char_p2751, char_p2752, char_p2753,
 char_p2754, char_p2755, char_p2756, char_p2757, char_p2758,
 char_p2759, char_p2762, char_p2763, char_p2241, char_p726,
 char_p2242, char_p2765, char_p2273, char_p551, char_p552,
 char_p553, char_p554, char_p555, char_p556, char_p557,
 char_p558, char_p559, char_p560, char_p561, char_p562,
 char_p563, char_p564, char_p565, char_p566, char_p567,
 char_p568, char_p569, char_p570, char_p571, char_p572,
 char_p573, char_p574, char_p575, char_p576, char_p2223,
 char_p804, char_p2224, char_p2262, char_p999, char_p2766,
 char_p651, char_p652, char_p653, char_p654, char_p655,
 char_p656, char_p657, char_p658, char_p659, char_p660,
 char_p661, char_p662, char_p663, char_p664, char_p665,
 char_p666, char_p667, char_p668, char_p669, char_p670,
 char_p671, char_p672, char_p673, char_p674, char_p675,
 char_p676, char_p2225, char_p723, char_p2226, char_p2246,
 char_p718,
};

#endif /* S_SIM_INC_H */
```

Triplex Bold Vector Font Set

Listing C.2 introduces a complex vector font that appears as elaborate boldface font.

Listing C.2 The `triplex_bold.h` Font Definition

```
#ifndef TRI_BLD_INC_H
#define TRI_BLD_INC_H

#include "vfonts/vector_chars.h"

XPoint **plain_triplex_bold[] = {
 char3199, char3214, char3228, char2275, char3219, char2271,
 char3218, char3217, char3221, char3222, char3223, char3225,
 char3211, char3224, char3210, char3220, char3200, char3201,
 char3202, char3203, char3204, char3205, char3206, char3207,
 char3208, char3209, char3212, char3213, char2241, char3226,
 char2242, char3215, char2273, char3001, char3002, char3003,
 char3004, char3005, char3006, char3007, char3008, char3009,
 char3010, char3011, char3012, char3013, char3014, char3015,
 char3016, char3017, char3018, char3019, char3020, char3021,
 char3022, char3023, char3024, char3025, char3026, char2223,
 char804, char2224, char2262, char999, char3216, char3101,
 char3102, char3103, char3104, char3105, char3106, char3107,
 char3108, char3109, char3110, char3111, char3112, char3113,
 char3114, char3115, char3116, char3117, char3118, char3119,
 char3120, char3121, char3122, char3123, char3124, char3125,
 char3126, char2225, char2229, char2226, char2246, char3229,
};

int *plain_triplex_bold_p[] = {
 char_p3199, char_p3214, char_p3228, char_p2275, char_p3219,
 char_p2271, char_p3218, char_p3217, char_p3221, char_p3222,
 char_p3223, char_p3225, char_p3211, char_p3224, char_p3210,
 char_p3220, char_p3200, char_p3201, char_p3202, char_p3203,
 char_p3204, char_p3205, char_p3206, char_p3207, char_p3208,
 char_p3209, char_p3212, char_p3213, char_p2241, char_p3226,
 char_p2242, char_p3215, char_p2273, char_p3001, char_p3002,
 char_p3003, char_p3004, char_p3005, char_p3006, char_p3007,
 char_p3008, char_p3009, char_p3010, char_p3011, char_p3012,
 char_p3013, char_p3014, char_p3015, char_p3016, char_p3017,
 char_p3018, char_p3019, char_p3020, char_p3021, char_p3022,
 char_p3023, char_p3024, char_p3025, char_p3026, char_p2223,
 char_p804, char_p2224, char_p2262, char_p999, char_p3216,
 char_p3101, char_p3102, char_p3103, char_p3104, char_p3105,
 char_p3106, char_p3107, char_p3108, char_p3109, char_p3110,
 char_p3111, char_p3112, char_p3113, char_p3114, char_p3115,
 char_p3116, char_p3117, char_p3118, char_p3119, char_p3120,
 char_p3121, char_p3122, char_p3123, char_p3124, char_p3125,
 char_p3126, char_p2225, char_p2229, char_p2226, char_p2246,
 char_p3229,
};

#endif /* TRI_BLD_INC_H */
```

Triplex Bold Italic Vector Font Set

Similar in appearance to the vector font introduced in Listing C.2, the triplex bold italic font found in Listing C.3 can be used for the `Text` object introduced in Chapter 24.

Listing C.3 The `triplex_bold_italic.h` Font Definition

```
#ifndef TRI_BLD_ITL_INC_H
#define TRI_BLD_ITL_INC_H

#include "vfonts/vector_chars.h"
```

continues

Listing C.3 Continued

```

XPoint **triplex_bold_italic[] = {
 char3249, char3264, char3278, char2275, char3269, char2271,
 char3268, char3267, char3271, char3272, char3273, char3275,
 char3261, char3274, char3260, char3270, char3250, char3251,
 char3252, char3253, char3254, char3255, char3256, char3257,
 char3258, char3259, char3262, char3263, char2241, char3276,
 char2242, char3265, char2273, char3051, char3052, char3053,
 char3054, char3055, char3056, char3057, char3058, char3059,
 char3060, char3061, char3062, char3063, char3064, char3065,
 char3066, char3067, char3068, char3069, char3070, char3071,
 char3072, char3073, char3074, char3075, char3076, char2223,
 char804, char2224, char2262, char999, char3266, char3151,
 char3152, char3153, char3154, char3155, char3156, char3157,
 char3158, char3159, char3160, char3161, char3162, char3163,
 char3164, char3165, char3166, char3167, char3168, char3169,
 char3170, char3171, char3172, char3173, char3174, char3175,
 char3176, char2225, char2229, char2226, char2246, char3279,
};

int *triplex_bold_italic_p[] = {
 char_p3249, char_p3264, char_p3278, char_p2275, char_p3269,
 char_p2271, char_p3268, char_p3267, char_p3271, char_p3272,
 char_p3273, char_p3275, char_p3261, char_p3274, char_p3260,
 char_p3270, char_p3250, char_p3251, char_p3252, char_p3253,
 char_p3254, char_p3255, char_p3256, char_p3257, char_p3258,
 char_p3259, char_p3262, char_p3263, char_p2241, char_p3276,
 char_p2242, char_p3265, char_p2273, char_p3051, char_p3052,
 char_p3053, char_p3054, char_p3055, char_p3056, char_p3057,
 char_p3058, char_p3059, char_p3060, char_p3061, char_p3062,
 char_p3063, char_p3064, char_p3065, char_p3066, char_p3067,
 char_p3068, char_p3069, char_p3070, char_p3071, char_p3072,
 char_p3073, char_p3074, char_p3075, char_p3076, char_p2223,
 char_p804, char_p2224, char_p2262, char_p999, char_p3266,
 char_p3151, char_p3152, char_p3153, char_p3154, char_p3155,
 char_p3156, char_p3157, char_p3158, char_p3159, char_p3160,
 char_p3161, char_p3162, char_p3163, char_p3164, char_p3165,
 char_p3166, char_p3167, char_p3168, char_p3169, char_p3170,
 char_p3171, char_p3172, char_p3173, char_p3174, char_p3175,
 char_p3176, char_p2225, char_p2229, char_p2226, char_p2246,
 char_p3279,
};

#endif /* TRI_BLD_ITL_INC_H */
```

The `vector_chars.h` Header File

The content of the `vector_chars.h` header file is found in Listing C.4. It provides the character, segment, and point definitions for all the vector font sets introduced in this appendix as well as in Chapter 24.

Listing C.4 The `vector_chars.h` File

```
#ifndef VEC_FONTS_INC_H
#define VEC_FONTS_INC_H

#include <X11/Xlib.h>

static XPoint seg0_501[] = {
    {0,-12},{-8,9},
};
static XPoint seg1_501[] = {
    {0,-12},{8,9},
};
static XPoint seg2_501[] = {
    {-5,2},{5,2},
};
static XPoint *char501[] = {
    seg0_501,seg1_501,seg2_501,
    NULL,
};
static int char_p501[] = {
    XtNumber(seg0_501),XtNumber(seg1_501),XtNumber(seg2_501),
};
static XPoint seg0_502[] = {
    {-7,-12},{-7,9},
};
static XPoint seg1_502[] = {
    {-7,-12},{2,-12},{5,-11},{6,-10},{7,-8},{7,-6},{6,-4},{5,-3},
    {2,-2},
};
static XPoint seg2_502[] = {
    {-7,-2},{2,-2},{5,-1},{6,0},{7,2},{7,5},{6,7},{5,8},
    {2,9},{-7,9},
};
static XPoint *char502[] = {
    seg0_502,seg1_502,seg2_502,
    NULL,
};
static int char_p502[] = {
    XtNumber(seg0_502),XtNumber(seg1_502),XtNumber(seg2_502),
};
static XPoint seg0_503[] = {
    {8,-7},{7,-9},{5,-11},{3,-12},{-1,-12},{-3,-11},{-5,-9},
    {-6,-7},{-7,-4},{-7,1},{-6,4},{-5,6},{-3,8},{-1,9},{3,9},
    {5,8},{7,6},{8,4},
};
static XPoint *char503[] = {
    seg0_503,
    NULL,
};
static int char_p503[] = {
    XtNumber(seg0_503),
};
```

continues

Listing C.4 Continued

```
static XPoint seg0_504[] = {
    {-7,-12},{-7,9},
};
static XPoint seg1_504[] = {
    {-7,-12},{0,-12},{3,-11},{5,-9},{6,-7},{7,-4},{7,1},{6,4},
    {5,6},{3,8},{0,9},{-7,9},
};
static XPoint *char504[] = {
    seg0_504,seg1_504,
    NULL,
};
static int char_p504[] = {
    XtNumber(seg0_504),XtNumber(seg1_504),
};
static XPoint seg0_505[] = {
    {-6,-12},{-6,9},
};
static XPoint seg1_505[] = {
    {-6,-12},{7,-12},
};
static XPoint seg2_505[] = {
    {-6,-2},{2,-2},
};
static XPoint seg3_505[] = {
    {-6,9},{7,9},
};
static XPoint *char505[] = {
    seg0_505,seg1_505,seg2_505,seg3_505,
    NULL,
};
static int char_p505[] = {
    XtNumber(seg0_505),XtNumber(seg1_505),XtNumber(seg2_505),
    XtNumber(seg3_505),
};
static XPoint seg0_506[] = {
    {-6,-12},{-6,9},
};
static XPoint seg1_506[] = {
    {-6,-12},{7,-12},
};
static XPoint seg2_506[] = {
    {-6,-2},{2,-2},
};
static XPoint *char506[] = {
    seg0_506,seg1_506,seg2_506,
    NULL,
};
static int char_p506[] = {
   XtNumber(seg0_506),XtNumber(seg1_506),XtNumber(seg2_506),
};
```

```
static XPoint seg0_507[] = {
    {8,-7},{7,-9},{5,-11},{3,-12},{-1,-12},{-3,-11},
    {-5,-9},{-6,-7},{-7,-4},{-7,1},{-6,4},{-5,6},{-3,8},
    {-1,9},{3,9},{5,8},{7,6},{8,4},{8,1},
};
static XPoint seg1_507[] = {
    {3,1},{8,1},
};
static XPoint *char507[] = {
    seg0_507,seg1_507,
    NULL,
};
static int char_p507[] = {
    XtNumber(seg0_507),XtNumber(seg1_507),
};
static XPoint seg0_508[] = {
    {-7,-12},{-7,9},
};
static XPoint seg1_508[] = {
    {7,-12},{7,9},
};
static XPoint seg2_508[] = {
    {-7,-2},{7,-2},
};
static XPoint *char508[] = {
    seg0_508,seg1_508,seg2_508,
    NULL,
};
static int char_p508[] = {
    XtNumber(seg0_508),XtNumber(seg1_508),XtNumber(seg2_508),
};
static XPoint seg0_509[] = {
    {0,-12},{0,9},
};
static XPoint *char509[] = {
    seg0_509,
    NULL,
};
static int char_p509[] = {
    XtNumber(seg0_509),
};
static XPoint seg0_510[] = {
    {4,-12},{4,4},{3,7},{2,8},
    {0,9},{-2,9},{-4,8},{-5,7},{-6,4},{-6,2},
};
static XPoint *char510[] = {
    seg0_510,
    NULL,
};
static int char_p510[] = {
    XtNumber(seg0_510),
};
```

continues

Listing C.4 Continued

```
static XPoint seg0_511[] = {
    {-7,-12},{-7,9},
};
static XPoint seg1_511[] = {
    {7,-12},{-7,2},
};
static XPoint seg2_511[] = {
    {-2,-3},{7,9},
};
static XPoint *char511[] = {
    seg0_511,seg1_511,seg2_511,
    NULL,
};
static int char_p511[] = {
    XtNumber(seg0_511),XtNumber(seg1_511),XtNumber(seg2_511),
};
static XPoint seg0_512[] = {
    {-6,-12},{-6,9},
};
static XPoint seg1_512[] = {
    {-6,9},{6,9},
};
static XPoint *char512[] = {
    seg0_512,seg1_512,
    NULL,
};
static int char_p512[] = {
    XtNumber(seg0_512),XtNumber(seg1_512),
};
static XPoint seg0_513[] = {
    {-8,-12},{-8,9},
};
static XPoint seg1_513[] = {
    {-8,-12},{0,9},
};
static XPoint seg2_513[] = {
    {8,-12},{0,9},
};
static XPoint seg3_513[] = {
    {8,-12},{8,9},
};
static XPoint *char513[] = {
    seg0_513,seg1_513,seg2_513,seg3_513,
    NULL,
};
static int char_p513[] = {
    XtNumber(seg0_513),XtNumber(seg1_513),XtNumber(seg2_513),
    XtNumber(seg3_513),
};
static XPoint seg0_514[] = {
    {-7,-12},{-7,9},
};
```

```
static XPoint seg1_514[] = {
    {-7,-12},{7,9},
};
static XPoint seg2_514[] = {
    {7,-12},{7,9},
};
static XPoint *char514[] = {
    seg0_514,seg1_514,seg2_514,
    NULL,
};
static int char_p514[] = {
    XtNumber(seg0_514),XtNumber(seg1_514),XtNumber(seg2_514),
};
static XPoint seg0_515[] = {
    {-2,-12},{-4,-11},{-6,-9},{-7,-7},{-8,-4},{-8,1},
    {-7,4},{-6,6},{-4,8},{-2,9},{2,9},{4,8},{6,6},{7,4},{8,1},
    {8,-4},{7,-7},{6,-9},{4,-11},{2,-12},{-2,-12},
};
static XPoint *char515[] = {
    seg0_515,
    NULL,
};
static int char_p515[] = {
    XtNumber(seg0_515),
};
static XPoint seg0_516[] = {
    {-7,-12},{-7,9},
};
static XPoint seg1_516[] = {
    {-7,-12},{2,-12},{5,-11},{6,-10},{7,-8},{7,-5},{6,-3},{5,-2},
    {2,-1},{-7,-1},
};
static XPoint *char516[] = {
    seg0_516,seg1_516,
    NULL,
};
static int char_p516[] = {
    XtNumber(seg0_516),XtNumber(seg1_516),
};
static XPoint seg0_517[] = {
    {-2,-12},{-4,-11},{-6,-9},{-7,-7},{-8,-4},{-8,1},{-7,4},
    {-6,6},{-4,8},{-2,9},{2,9},{4,8},{6,6},{7,4},{8,1},{8,-4},
    {7,-7},{6,-9},{4,-11},{2,-12},{-2,-12},
};
static XPoint seg1_517[] = {
    {1,5},{7,11},
};
static XPoint *char517[] = {
    seg0_517,seg1_517,
    NULL,
};
```

continues

Listing C.4 Continued

```
static int char_p517[] = {
    XtNumber(seg0_517),XtNumber(seg1_517),
};
static XPoint seg0_518[] = {
    {-7,-12},{-7,9},
};
static XPoint seg1_518[] = {
    {-7,-12},{2,-12},{5,-11},{6,-10},{7,-8},{7,-6},{6,-4},{5,-3},
    {2,-2},{-7,-2},
};
static XPoint seg2_518[] = {
    {0,-2},{7,9},
};
static XPoint *char518[] = {
    seg0_518,seg1_518,seg2_518,
    NULL,
};
static int char_p518[] = {
    XtNumber(seg0_518),XtNumber(seg1_518),XtNumber(seg2_518),
};
static XPoint seg0_519[] = {
    {7,-9},{5,-11},{2,-12},{-2,-12},{-5,-11},{-7,-9},
    {-7,-7},{-6,-5},{-5,-4},{-3,-3},{3,-1},{5,0},{6,1},
    {7,3},{7,6},{5,8},{2,9},{-2,9},{-5,8},{-7,6},
};
static XPoint *char519[] = {
    seg0_519,
    NULL,
};
static int char_p519[] = {
    XtNumber(seg0_519),
};
static XPoint seg0_520[] = {
    {0,-12},{0,9},
};
static XPoint seg1_520[] = {
    {-7,-12},{7,-12},
};
static XPoint *char520[] = {
    seg0_520,seg1_520,
    NULL,
};
static int char_p520[] = {
    XtNumber(seg0_520),XtNumber(seg1_520),
};
static XPoint seg0_521[] = {
    {-7,-12},{-7,3},{-6,6},{-4,8},{-1,9},{1,9},
    {4,8},{6,6},{7,3},{7,-12},
};
static XPoint *char521[] = {
    seg0_521,
    NULL,
};
```

```
static int char_p521[] = {
    XtNumber(seg0_521),
};
static XPoint seg0_522[] = {
    {-8,-12},{0,9},
};
static XPoint seg1_522[] = {
    {8,-12},{0,9},
};
static XPoint *char522[] = {
    seg0_522,seg1_522,
    NULL,
};
static int char_p522[] = {
    XtNumber(seg0_522),XtNumber(seg1_522),
};
static XPoint seg0_523[] = {
    {-10,-12},{-5,9},
};
static XPoint seg1_523[] = {
    {0,-12},{-5,9},
};
static XPoint seg2_523[] = {
    {0,-12},{5,9},
};
static XPoint seg3_523[] = {
    {10,-12},{5,9},
};
static XPoint *char523[] = {
    seg0_523,seg1_523,seg2_523,seg3_523,
    NULL,
};
static int char_p523[] = {
    XtNumber(seg0_523),XtNumber(seg1_523),XtNumber(seg2_523),
    XtNumber(seg3_523),
};
static XPoint seg0_524[] = {
    {-7,-12},{7,9},
};
static XPoint seg1_524[] = {
    {7,-12},{-7,9},
};
static XPoint *char524[] = {
    seg0_524,seg1_524,
    NULL,
};
static int char_p524[] = {
    XtNumber(seg0_524),XtNumber(seg1_524),
};
static XPoint seg0_525[] = {
    {-8,-12},{0,-2},{0,9},
};
```

continues

Listing C.4 Continued

```
static XPoint seg1_525[] = {
    {8,-12},{0,-2},
};
static XPoint *char525[] = {
    seg0_525,seg1_525,
    NULL,
};
static int char_p525[] = {
    XtNumber(seg0_525),XtNumber(seg1_525),
};
static XPoint seg0_526[] = {
    {7,-12},{-7,9},
};
static XPoint seg1_526[] = {
    {-7,-12},{7,-12},
};
static XPoint seg2_526[] = {
    {-7,9},{7,9},
};
static XPoint *char526[] = {
    seg0_526,seg1_526,seg2_526,
    NULL,
};
static int char_p526[] = {
    XtNumber(seg0_526),XtNumber(seg1_526),XtNumber(seg2_526),
};
static XPoint seg0_551[] = {
    {-11,9},{-9,8},{-6,5},{-3,1},{1,-6},
    {4,-12},{4,9},{3,6},{1,3},{-1,1},{-4,-1},
    {-6,-1},{-7,0},{-7,2},{-6,4},{-4,6},{-1,8},{2,9},{7,9},
};
static XPoint *char551[] = {
    seg0_551,
    NULL,
};
static int char_p551[] = {
    XtNumber(seg0_551),
};
static XPoint seg0_552[] = {
    {1,-10},{2,-9},{2,-6},{1,-2},
    {0,1},{-1,3},{-3,6},{-5,8},{-7,9},{-8,9},{-9,8},{-9,5},{-8,0},
    {-7,-3},{-6,-5},{-4,-8},{-2,-10},{0,-11},{3,-12},{6,-12},
    {8,-11},{9,-9},{9,-7},{8,-5},{7,-4},{5,-3},{2,-2},
};
static XPoint seg1_552[] = {
    {1,-2},{2,-2},{5,-1},{6,0},{7,2},{7,5},{6,7},{5,8},
    {3,9},{0,9},{-2,8},{-3,6},
};
static XPoint *char552[] = {
    seg0_552,seg1_552,
    NULL,
};
static int char_p552[] = {
    XtNumber(seg0_552),XtNumber(seg1_552),
};
static XPoint seg0_553[] = {
    {2,-6},{2,-5},{3,-4},{5,-4},{7,-5},
    {8,-7},{8,-9},{7,-11},{5,-12},{2,-12},{-1,-11},{-3,-9},
    {-5,-6},{-6,-4},{-7,0},{-7,4},{-6,7},{-5,8},{-3,9},
    {-1,9},{2,8},{4,6},{5,4},
};
static XPoint *char553[] = {
    seg0_553,
    NULL,
};
static int char_p553[] = {
    XtNumber(seg0_553),
};
static XPoint seg0_554[] = {
    {2,-12},{0,-11},{-1,-9},{-2,-5},{-3,1},{-4,4},{-5,6},{-7,8},
    {-9,9},{-11,9},{-12,8},{-12,6},{-11,5},{-9,5},{-7,6},{-5,8},
    {-2,9},{1,9},{4,8},{6,6},{8,2},{9,-3},{9,-7},{8,-10},{7,-11},
    {5,-12},{2,-12},{0,-10},{0,-8},{1,-5},{3,-2},{5,0},{8,2},
    {10,3},
};
static XPoint *char554[] = {
    seg0_554,
    NULL,
};
static int char_p554[] = {
    XtNumber(seg0_554),
};
static XPoint seg0_555[] = {
    {4,-8},{4,-7},{5,-6},{7,-6},{8,-7},{8,-9},{7,-11},{4,-12},
    {0,-12},{-3,-11},{-4,-9},{-4,-6},{-3,-4},{-2,-3},{1,-2},
    {-2,-2},{-5,-1},{-6,0},{-7,2},{-7,5},{-6,7},{-5,8},{-2,9},
    {1,9},{4,8},{6,6},{7,4},
};
static XPoint *char555[] = {
    seg0_555,
    NULL,
};
static int char_p555[] = {
    XtNumber(seg0_555),
};
static XPoint seg0_556[] = {
    {0,-6},{-2,-6},{-4,-7},{-5,-9},{-4,-11},{-1,-12},{2,-12},
    {6,-11},{9,-11},{11,-12},
};
static XPoint seg1_556[] = {
    {6,-11},{4,-4},{2,2},{0,6},{-2,8},{-4,9},{-6,9},{-8,8},
    {-9,6},{-9,4},{-8,3},{-6,3},{-4,4},
};
```

continues

Listing C.4 Continued

```
static XPoint seg2_556[] = {
    {-1,-2},{8,-2},
};
static XPoint *char556[] = {
    seg0_556,seg1_556,seg2_556,
    NULL,
};
static int char_p556[] = {
    XtNumber(seg0_556),XtNumber(seg1_556),XtNumber(seg2_556),
};
static XPoint seg0_557[] = {
    {-11,9},{-9,8},{-5,4},{-2,-1},{-1,-4},{0,-8},
    {0,-11},{-1,-12},{-2,-12},{-3,-11},{-4,-9},{-4,-6},{-3,-4},
    {-1,-3},{3,-3},{6,-4},{7,-5},{8,-7},{8,-1},{7,4},{6,6},
    {4,8},{1,9},{-3,9},{-6,8},{-8,6},{-9,4},{-9,2},
};
static XPoint *char557[] = {
    seg0_557,
    NULL,
};
static int char_p557[] = {
    XtNumber(seg0_557),
};
static XPoint seg0_558[] = {
    {-5,-5},{-7,-6},{-8,-8},{-8,-9},{-7,-11},
    {-5,-12},{-4,-12},{-2,-11},{-1,-9},{-1,-7},{-2,-3},
    {-4,3},{-6,7},{-8,9},{-10,9},{-11,8},{-11,6},
};
static XPoint seg1_558[] = {
    {-5,0},{4,-3},{6,-4},{9,-6},{11,-8},{12,-10},{12,-11},{11,-12},
    {10,-12},{8,-10},{6,-6},{4,0},{3,5},{3,8},{4,9},{5,9},{7,8},
    {8,7},{10,4},
};
static XPoint *char558[] = {
    seg0_558,seg1_558,
    NULL,
};
static int char_p558[] = {
    XtNumber(seg0_558),XtNumber(seg1_558),
};
static XPoint seg0_559[] = {
    {5,4},{3,2},{1,-1},{0,-3},{-1,-6},{-1,-9},{0,-11},
    {1,-12},{3,-12},{4,-11},{5,-9},{5,-6},{4,-1},{2,4},{1,6},
    {-1,8},{-3,9},{-5,9},{-7,8},{-8,6},{-8,4},{-7,3},{-5,3},
    {-3,4},
};
static XPoint *char559[] = {
    seg0_559,
    NULL,
};
```

```
static int char_p559[] = {
    XtNumber(seg0_559),
};
static XPoint seg0_560[] = {
    {2,12},{0,9},{-2,4},{-3,-2},{-3,-8},{-2,-11},{0,-12},{2,-12},
    {3,-11},{4,-8},{4,-5},{3,0},{0,9},{-2,15},{-3,18},{-4,20},
    {-6,21},{-7,20},{-7,18},{-6,15},{-4,12},{-2,10},{1,8},{5,6},
};
static XPoint *char560[] = {
    seg0_560,
    NULL,
};
static int char_p560[] = {
    XtNumber(seg0_560),
};
static XPoint seg0_561[] = {
    {-5,-5},{-7,-6},{-8,-8},{-8,-9},{-7,-11},{-5,-12},{-4,-12},
    {-2,-11},{-1,-9},{-1,-7},{-2,-3},{-4,3},{-6,7},
    {-8,9},{-10,9},{-11,8},{-11,6},
};
static XPoint seg1_561[] = {
    {12,-9},{12,-11},{11,-12},{10,-12},{8,-11},{6,-9},{4,-6},
    {2,-4},{0,-3},{-2,-3},
};
static XPoint seg2_561[] = {
    {0,-3},{1,-1},{1,6},{2,8},{3,9},{4,9},{6,8},{7,7},
    {9,4},
};
static XPoint *char561[] = {
    seg0_561,seg1_561,seg2_561,
    NULL,
};
static int char_p561[] = {
    XtNumber(seg0_561),XtNumber(seg1_561),XtNumber(seg2_561),
};
static XPoint seg0_562[] = {
    {-5,0},{-3,0},{1,-1},{4,-3},{6,-5},{7,-7},{7,-10},{6,-12},
    {4,-12},{3,-11},{2,-9},{1,-4},{0,1},{-1,4},{-2,6},{-4,8},
    {-6,9},{-8,9},{-9,8},{-9,6},{-8,5},{-6,5},{-4,6},{-1,8},
    {2,9},{4,9},{7,8},{9,6},
};
static XPoint *char562[] = {
    seg0_562,
    NULL,
};
static int char_p562[] = {
    XtNumber(seg0_562),
};
static XPoint seg0_563[] = {
    {-13,-5},{-15,-6},{-16,-8},{-16,-9},{-15,-11},{-13,-12},
    {-12,-12},{-10,-11},{-9,-9},{-9,-7},{-10,-2},{-11,2},{-13,9},
};
```

continues

Listing C.4 Continued

```
static XPoint seg1_563[] = {
    {-11,2},{-8,-6},{-6,-10},{-5,-11},{-3,-12},{-2,-12},{0,-11},
    {1,-9},{1,-7},{0,-2},{-1,2},{-3,9},
};
static XPoint seg2_563[] = {
    {-1,2},{2,-6},{4,-10},{5,-11},{7,-12},{8,-12},{10,-11},{11,-9},
    {11,-7},{10,-2},{8,5},{8,8},{9,9},{10,9},{12,8},{13,7},{15,4},
};
static XPoint *char563[] = {
    seg0_563,seg1_563,seg2_563,
    NULL,
};
static int char_p563[] = {
    XtNumber(seg0_563),XtNumber(seg1_563),XtNumber(seg2_563),
};
static XPoint seg0_564[] = {
    {-8,-5},{-10,-6},{-11,-8},{-11,-9},{-10,-11},{-8,-12},{-7,-12},
    {-5,-11},{-4,-9},{-4,-7},{-5,-2},{-6,2},{-8,9},
};
static XPoint seg1_564[] = {
    {-6,2},{-3,-6},{-1,-10},{0,-11},{2,-12},{4,-12},{6,-11},{7,-9},
    {7,-7},{6,-2},{4,5},{4,8},{5,9},{6,9},{8,8},{9,7},{11,4},
};
static XPoint *char564[] = {
    seg0_564,seg1_564,
    NULL,
};
static int char_p564[] = {
    XtNumber(seg0_564),XtNumber(seg1_564),
};
static XPoint seg0_565[] = {
    {2,-12},{-1,-11},{-3,-9},{-5,-6},{-6,-4},{-7,0},{-7,4},{-6,7},
    {-5,8},{-3,9},{-1,9},{2,8},{4,6},{6,3},{7,1},{8,-3},{8,-7},
    {7,-10},{6,-11},{4,-12},{2,-12},{0,-10},{0,-7},{1,-4},{3,-1},
    {5,1},{8,3},{10,4},
};
static XPoint *char565[] = {
    seg0_565,
    NULL,
};
static int char_p565[] = {
    XtNumber(seg0_565),
};
static XPoint seg0_566[] = {
    {1,-10},{2,-9},{2,-6},{1,-2},{0,1},{-1,3},{-3,6},{-5,8},
    {-7,9},{-8,9},{-9,8},{-9,5},{-8,0},{-7,-3},{-6,-5},{-4,-8},
    {-2,-10},{0,-11},{3,-12},{8,-12},{10,-11},{11,-10},{12,-8},
    {12,-5},{11,-3},{10,-2},{8,-1},{5,-1},{3,-2},{2,-3},
};
static XPoint *char566[] = {
    seg0_566,
    NULL,
};
static int char_p566[] = {
    XtNumber(seg0_566),
};
static XPoint seg0_567[] = {
    {3,-6},{2,-4},{1,-3},{-1,-2},{-3,-2},{-4,-4},{-4,-6},{-3,-9},
    {-1,-11},{2,-12},{5,-12},{7,-11},{8,-9},{8,-5},
    {7,-2},{5,1},{1,5},{-2,7},{-4,8},{-7,9},{-9,9},{-10,8},{-10,6},
    {-9,5},{-7,5},{-5,6},{-2,8},{1,9},{4,9},{7,8},{9,6},
};
static XPoint *char567[] = {
    seg0_567,
    NULL,
};
static int char_p567[] = {
    XtNumber(seg0_567),
};
static XPoint seg0_568[] = {
    {1,-10},{2,-9},{2,-6},{1,-2},{0,1},{-1,3},{-3,6},{-5,8},
    {-7,9},{-8,9},{-9,8},{-9,5},{-8,0},{-7,-3},{-6,-5},{-4,-8},
    {-2,-10},{0,-11},{3,-12},{7,-12},{9,-11},{10,-10},{11,-8},
    {11,-5},{10,-3},{9,-2},{7,-1},{4,-1},{1,-2},{2,-1},{3,1},
    {3,6},{4,8},{6,9},{8,8},{9,7},{11,4},
};
static XPoint *char568[] = {
    seg0_568,
    NULL,
};
static int char_p568[] = {
    XtNumber(seg0_568),
};
static XPoint seg0_569[] = {
    {-10,9},{-8,8},{-6,6},{-3,2},{-1,-1},{1,-5},{2,-8},{2,-11},
    {1,-12},{0,-12},{-1,-11},{-2,-9},{-2,-7},{-1,-5},{1,-3},{4,-1},
    {6,1},{7,3},{7,5},{6,7},{5,8},{2,9},{-2,9},{-5,8},{-7,6},
    {-8,4},{-8,2},
};
static XPoint *char569[] = {
    seg0_569,
    NULL,
};
static int char_p569[] = {
    XtNumber(seg0_569),
};
static XPoint seg0_570[] = {
    {0,-6},{-2,-6},{-4,-7},{-5,-9},{-4,-11},{-1,-12},{2,-12},
    {6,-11},{9,-11},{11,-12},
};
static XPoint seg1_570[] = {
    {6,-11},{4,-4},{2,2},{0,6},{-2,8},{-4,9},{-6,9},{-8,8},
    {-9,6},{-9,4},{-8,3},{-6,3},{-4,4},
};
```

continues

Listing C.4 Continued

```
static XPoint *char570[] = {
    seg0_570,seg1_570,
    NULL,
};
static int char_p570[] = {
    XtNumber(seg0_570),XtNumber(seg1_570),
};
static XPoint seg0_571[] = {
    {-8,-5},{-10,-6},{-11,-8},{-11,-9},{-10,-11},{-8,-12},{-7,-12},
    {-5,-11},{-4,-9},{-4,-7},{-5,-3},{-6,0},{-7,4},
    {-7,6},{-6,8},{-4,9},{-2,9},{0,8},{1,7},{3,3},{6,-5},{8,-12},
};
static XPoint seg1_571[] = {
    {6,-5},{5,-1},{4,5},{4,8},{5,9},{6,9},{8,8},{9,7},
    {11,4},
};
static XPoint *char571[] = {
    seg0_571,seg1_571,
    NULL,
};
static int char_p571[] = {
    XtNumber(seg0_571),XtNumber(seg1_571),
};
static XPoint seg0_572[] = {
    {-7,-5},{-9,-6},{-10,-8},{-10,-9},{-9,-11},{-7,-12},
    {-6,-12},{-4,-11},{-3,-9},{-3,-7},{-4,-3},{-5,0},{-6,4},
    {-6,7},{-5,9},{-3,9},{-1,8},{2,5},{4,2},{6,-2},{7,-5},
    {8,-9},{8,-11},{7,-12},{6,-12},{5,-11},{4,-9},{4,-7},{5,-4},
    {7,-2},{9,-1},
};
static XPoint *char572[] = {
    seg0_572,
    NULL,
};
static int char_p572[] = {
    XtNumber(seg0_572),
};
static XPoint seg0_573[] = {
    {-10,-5},{-12,-6},{-13,-8},{-13,-9},{-12,-11},{-10,-12},
    {-9,-12},{-7,-11},{-6,-9},{-6,-6},{-7,9},
};
static XPoint seg1_573[] = {
    {3,-12},{-7,9},
};
static XPoint seg2_573[] = {
    {3,-12},{1,9},
};
static XPoint seg3_573[] = {
    {15,-12},{13,-11},{10,-8},{7,-4},{4,2},{1,9},
};
```

```
static XPoint *char573[] = {
    seg0_573,seg1_573,seg2_573,seg3_573,
    NULL,
};
static int char_p573[] = {
    XtNumber(seg0_573),XtNumber(seg1_573),XtNumber(seg2_573),
    XtNumber(seg3_573),
};
static XPoint seg0_574[] = {
    {-4,-6},{-6,-6},{-7,-7},{-7,-9},{-6,-11},{-4,-12},{-2,-12},
    {0,-11},{1,-9},{1,-6},{-1,3},{-1,6},{0,8},{2,9},{4,9},{6,8},
    {7,6},{7,4},{6,3},{4,3},
};
static XPoint seg1_574[] = {
    {11,-9},{11,-11},{10,-12},{8,-12},{6,-11},{4,-9},{2,-6},{-2,3},
    {-4,6},{-6,8},{-8,9},{-10,9},{-11,8},{-11,6},
};
static XPoint *char574[] = {
    seg0_574,seg1_574,
    NULL,
};
static int char_p574[] = {
    XtNumber(seg0_574),XtNumber(seg1_574),
};
static XPoint seg0_575[] = {
    {-7,-5},{-9,-6},{-10,-8},{-10,-9},{-9,-11},{-7,-12},{-6,-12},
    {-4,-11},{-3,-9},{-3,-7},{-4,-3},{-5,0},{-6,4},{-6,6},{-5,8},
    {-4,9},{-2,9},{0,8},{2,6},{4,3},{5,1},{7,-5},
};
static XPoint seg1_575[] = {
    {9,-12},{7,-5},{4,5},{2,11},{0,16},{-2,20},{-4,21},{-5,20},
    {-5,18},{-4,15},{-2,12},{1,9},{4,7},{9,4},
};
static XPoint *char575[] = {
    seg0_575,seg1_575,
    NULL,
};
static int char_p575[] = {
    XtNumber(seg0_575),XtNumber(seg1_575),
};
static XPoint seg0_576[] = {
    {3,-6},{2,-4},{1,-3},{-1,-2},{-3,-2},{-4,-4},{-4,-6},
    {-3,-9},{-1,-11},{2,-12},{5,-12},{7,-11},{8,-9},{8,-5},{7,-2},
    {5,2},{2,5},{-2,8},{-4,9},{-7,9},{-8,8},{-8,6},{-7,5},{-4,5},
    {-2,6},{-1,7},{0,9},{0,12},{-1,15},{-2,17},{-4,20},{-6,21},
    {-7,20},{-7,18},{-6,15},{-4,12},{-1,9},{2,7},{8,4},
};
static XPoint *char576[] = {
    seg0_576,
    NULL,
};
static int char_p576[] = {
    XtNumber(seg0_576),
};
```

continues

Listing C.4 Continued

```
static XPoint seg0_601[] = {
    {6,-5},{6,9},
};
static XPoint seg1_601[] = {
    {6,-2},{4,-4},{2,-5},{-1,-5},{-3,-4},{-5,-2},{-6,1},{-6,3},
    {-5,6},{-3,8},{-1,9},{2,9},{4,8},{6,6},
};
static XPoint *char601[] = {
    seg0_601,seg1_601,
    NULL,
};
static int char_p601[] = {
    XtNumber(seg0_601),XtNumber(seg1_601),
};
static XPoint seg0_602[] = {
    {-6,-12},{-6,9},
};
static XPoint seg1_602[] = {
    {-6,-2},{-4,-4},{-2,-5},{1,-5},{3,-4},{5,-2},{6,1},{6,3},
    {5,6},{3,8},{1,9},{-2,9},{-4,8},{-6,6},
};
static XPoint *char602[] = {
    seg0_602,seg1_602,
    NULL,
};
static int char_p602[] = {
    XtNumber(seg0_602),XtNumber(seg1_602),
};
static XPoint seg0_603[] = {
    {6,-2},{4,-4},{2,-5},{-1,-5},{-3,-4},{-5,-2},{-6,1},
    {-6,3},{-5,6},{-3,8},{-1,9},{2,9},{4,8},{6,6},
};
static XPoint *char603[] = {
    seg0_603,
    NULL,
};
static int char_p603[] = {
    XtNumber(seg0_603),
};
static XPoint seg0_604[] = {
    {6,-12},{6,9},
};
static XPoint seg1_604[] = {
    {6,-2},{4,-4},{2,-5},{-1,-5},{-3,-4},{-5,-2},{-6,1},{-6,3},
    {-5,6},{-3,8},{-1,9},{2,9},{4,8},{6,6},
};
static XPoint *char604[] = {
    seg0_604,seg1_604,
    NULL,
};
```

```
static int char_p604[] = {
    XtNumber(seg0_604),XtNumber(seg1_604),
};
static XPoint seg0_605[] = {
    {-6,1},{6,1},{6,-1},{5,-3},{4,-4},{2,-5},{-1,-5},{-3,-4},
    {-5,-2},{-6,1},{-6,3},{-5,6},{-3,8},{-1,9},{2,9},{4,8},{6,6},
};
static XPoint *char605[] = {
    seg0_605,
    NULL,
};
static int char_p605[] = {
    XtNumber(seg0_605),
};
static XPoint seg0_606[] = {
    {5,-12},{3,-12},{1,-11},{0,-8},
    {0,9},
};
static XPoint seg1_606[] = {
    {-3,-5},{4,-5},
};
static XPoint *char606[] = {
    seg0_606,seg1_606,
    NULL,
};
static int char_p606[] = {
    XtNumber(seg0_606),XtNumber(seg1_606),
};
static XPoint seg0_607[] = {
    {6,-5},{6,11},{5,14},{4,15},{2,16},{-1,16},
    {-3,15},
};
static XPoint seg1_607[] = {
    {6,-2},{4,-4},{2,-5},{-1,-5},{-3,-4},{-5,-2},{-6,1},{-6,3},
    {-5,6},{-3,8},{-1,9},{2,9},{4,8},{6,6},
};
static XPoint *char607[] = {
    seg0_607,seg1_607,
    NULL,
};
static int char_p607[] = {
    XtNumber(seg0_607),XtNumber(seg1_607),
};
static XPoint seg0_608[] = {
    {-5,-12},{-5,9},
};
static XPoint seg1_608[] = {
    {-5,-1},{-2,-4},{0,-5},{3,-5},{5,-4},{6,-1},{6,9},
};
static XPoint *char608[] = {
    seg0_608,seg1_608,
    NULL,
};
```

continues

Listing C.4 Continued

```
static int char_p608[] = {
    XtNumber(seg0_608),XtNumber(seg1_608),
};
static XPoint seg0_609[] = {
    {-1,-12},
    {0,-11},{1,-12},{0,-13},{-1,-12},
};
static XPoint seg1_609[] = {
    {0,-5},{0,9},
};
static XPoint *char609[] = {
    seg0_609,seg1_609,
    NULL,
};
static int char_p609[] = {
    XtNumber(seg0_609),XtNumber(seg1_609),
};
static XPoint seg0_610[] = {
    {0,-12},{1,-11},{2,-12},{1,-13},{0,-12},
};
static XPoint seg1_610[] = {
    {1,-5},{1,12},{0,15},{-2,16},{-4,16},
};
static XPoint *char610[] = {
    seg0_610,seg1_610,
    NULL,
};
static int char_p610[] = {
    XtNumber(seg0_610),XtNumber(seg1_610),
};
static XPoint seg0_611[] = {
    {-5,-12},{-5,9},
};
static XPoint seg1_611[] = {
    {5,-5},{-5,5},
};
static XPoint seg2_611[] = {
    {-1,1},{6,9},
};
static XPoint *char611[] = {
    seg0_611,seg1_611,seg2_611,
    NULL,
};
static int char_p611[] = {
    XtNumber(seg0_611),XtNumber(seg1_611),XtNumber(seg2_611),
};
static XPoint seg0_612[] = {
    {0,-12},{0,9},
};
```

```
static XPoint *char612[] = {
    seg0_612,
    NULL,
};
static int char_p612[] = {
    XtNumber(seg0_612),
};
static XPoint seg0_613[] = {
    {-11,-5},{-11,9},
};
static XPoint seg1_613[] = {
    {-11,-1},{-8,-4},{-6,-5},{-3,-5},{-1,-4},{0,-1},{0,9},
};
static XPoint seg2_613[] = {
    {0,-1},{3,-4},{5,-5},{8,-5},{10,-4},{11,-1},{11,9},
};
static XPoint *char613[] = {
    seg0_613,seg1_613,seg2_613,
    NULL,
};
static int char_p613[] = {
    XtNumber(seg0_613),XtNumber(seg1_613),XtNumber(seg2_613),
};
static XPoint seg0_614[] = {
    {-5,-5},
    {-5,9},
};
static XPoint seg1_614[] = {
    {-5,-1},{-2,-4},{0,-5},{3,-5},{5,-4},{6,-1},{6,9},
};
static XPoint *char614[] = {
    seg0_614,seg1_614,
    NULL,
};
static int char_p614[] = {
    XtNumber(seg0_614),XtNumber(seg1_614),
};
static XPoint seg0_615[] = {
    {-1,-5},{-3,-4},{-5,-2},{-6,1},{-6,3},{-5,6},{-3,8},
    {-1,9},{2,9},{4,8},{6,6},{7,3},{7,1},{6,-2},{4,-4},
    {2,-5},{-1,-5},
};
static XPoint *char615[] = {
    seg0_615,
    NULL,
};
static int char_p615[] = {
    XtNumber(seg0_615),
};
static XPoint seg0_616[] = {
    {-6,-5},{-6,16},
};
```

continues

Listing C.4 Continued

```
static XPoint seg1_616[] = {
    {-6,-2},{-4,-4},{-2,-5},{1,-5},{3,-4},{5,-2},{6,1},{6,3},
    {5,6},{3,8},{1,9},{-2,9},{-4,8},{-6,6},
};
static XPoint *char616[] = {
    seg0_616,seg1_616,
    NULL,
};
static int char_p616[] = {
    XtNumber(seg0_616),XtNumber(seg1_616),
};
static XPoint seg0_617[] = {
    {6,-5},{6,16},
};
static XPoint seg1_617[] = {
    {6,-2},{4,-4},{2,-5},{-1,-5},{-3,-4},{-5,-2},{-6,1},{-6,3},
    {-5,6},{-3,8},{-1,9},{2,9},{4,8},{6,6},
};
static XPoint *char617[] = {
    seg0_617,seg1_617,
    NULL,
};
static int char_p617[] = {
    XtNumber(seg0_617),XtNumber(seg1_617),
};
static XPoint seg0_618[] = {
    {-3,-5},{-3,9},
};
static XPoint seg1_618[] = {
    {-3,1},{-2,-2},{0,-4},{2,-5},{5,-5},
};
static XPoint *char618[] = {
    seg0_618,seg1_618,
    NULL,
};
static int char_p618[] = {
    XtNumber(seg0_618),XtNumber(seg1_618),
};
static XPoint seg0_619[] = {
    {6,-2},{5,-4},{2,-5},{-1,-5},{-4,-4},{-5,-2},{-4,0},{-2,1},
    {3,2},{5,3},{6,5},{6,6},{5,8},{2,9},{-1,9},{-4,8},{-5,6},
};
static XPoint *char619[] = {
    seg0_619,
    NULL,
};
static int char_p619[] = {
    XtNumber(seg0_619),
};
static XPoint seg0_620[] = {
    {0,-12},{0,5},{1,8},{3,9},
    {5,9},
};
static XPoint seg1_620[] = {
    {-3,-5},{4,-5},
};
static XPoint *char620[] = {
    seg0_620,seg1_620,
    NULL,
};
static int char_p620[] = {
    XtNumber(seg0_620),XtNumber(seg1_620),
};
static XPoint seg0_621[] = {
    {-5,-5},{-5,5},{-4,8},{-2,9},{1,9},{3,8},
    {6,5},
};
static XPoint seg1_621[] = {
    {6,-5},{6,9},
};
static XPoint *char621[] = {
    seg0_621,seg1_621,
    NULL,
};
static int char_p621[] = {
    XtNumber(seg0_621),XtNumber(seg1_621),
};
static XPoint seg0_622[] = {
    {-6,-5},{0,9},
};
static XPoint seg1_622[] = {
    {6,-5},{0,9},
};
static XPoint *char622[] = {
    seg0_622,seg1_622,
    NULL,
};
static int char_p622[] = {
    XtNumber(seg0_622),XtNumber(seg1_622),
};
static XPoint seg0_623[] = {
    {-8,-5},{-4,9},
};
static XPoint seg1_623[] = {
    {0,-5},{-4,9},
};
static XPoint seg2_623[] = {
    {0,-5},{4,9},
};
static XPoint seg3_623[] = {
    {8,-5},{4,9},
};
static XPoint *char623[] = {
    seg0_623,seg1_623,seg2_623,seg3_623,
```

continues

Listing C.4 Continued

```
    NULL,
};
static int char_p623[] = {
    XtNumber(seg0_623),XtNumber(seg1_623),XtNumber(seg2_623),
    XtNumber(seg3_623),
};
static XPoint seg0_624[] = {
    {-5,-5},{6,9},
};
static XPoint seg1_624[] = {
    {6,-5},{-5,9},
};
static XPoint *char624[] = {
    seg0_624,seg1_624,
    NULL,
};
static int char_p624[] = {
    XtNumber(seg0_624),XtNumber(seg1_624),
};
static XPoint seg0_625[] = {
    {-6,-5},{0,9},
};
static XPoint seg1_625[] = {
    {6,-5},{0,9},{-2,13},{-4,15},{-6,16},{-7,16},
};
static XPoint *char625[] = {
    seg0_625,seg1_625,
    NULL,
};
static int char_p625[] = {
    XtNumber(seg0_625),XtNumber(seg1_625),
};
static XPoint seg0_626[] = {
    {6,-5},{-5,9},
};
static XPoint seg1_626[] = {
    {-5,-5},{6,-5},
};
static XPoint seg2_626[] = {
    {-5,9},{6,9},
};
static XPoint *char626[] = {
    seg0_626,seg1_626,seg2_626,
    NULL,
};
static int char_p626[] = {
    XtNumber(seg0_626),XtNumber(seg1_626),XtNumber(seg2_626),
};
static XPoint seg0_651[] = {
    {3,3},{2,1},{0,0},{-2,0},{-4,1},{-5,2},{-6,4},{-6,6},{-5,8},
    {-3,9},{-1,9},{1,8},{2,6},{4,0},{3,5},
    {3,8},{4,9},{5,9},{7,8},{8,7},{10,4},
};
static XPoint *char651[] = {
    seg0_651,
    NULL,
};
static int char_p651[] = {
    XtNumber(seg0_651),
};
static XPoint seg0_652[] = {
    {-5,4},{-3,1},{0,-4},{1,-6},{2,-9},{2,-11},{1,-12},{-1,-11},
    {-2,-9},{-3,-5},{-4,2},{-4,8},{-3,9},{-2,9},{0,8},{2,6},
    {3,3},{3,0},{4,4},{5,5},{7,5},{9,4},
};
static XPoint *char652[] = {
    seg0_652,
    NULL,
};
static int char_p652[] = {
    XtNumber(seg0_652),
};
static XPoint seg0_653[] = {
    {2,2},{2,1},{1,0},{-1,0},{-3,1},{-4,2},{-5,4},{-5,6},
    {-4,8},{-2,9},{1,9},{4,7},{6,4},
};
static XPoint *char653[] = {
    seg0_653,
    NULL,
};
static int char_p653[] = {
    XtNumber(seg0_653),
};
static XPoint seg0_654[] = {
    {3,3},{2,1},{0,0},{-2,0},
    {-4,1},{-5,2},{-6,4},{-6,6},{-5,8},{-3,9},{-1,9},{1,8},{2,6},
    {8,-12},
};
static XPoint seg1_654[] = {
    {4,0},{3,5},{3,8},{4,9},{5,9},{7,8},{8,7},{10,4},
};
static XPoint *char654[] = {
    seg0_654,seg1_654,
    NULL,
};
static int char_p654[] = {
    XtNumber(seg0_654),XtNumber(seg1_654),
};
static XPoint seg0_655[] = {
    {-3,7},{-1,6},{0,5},{1,3},{1,1},{0,0},{-1,0},{-3,1},{-4,3},
    {-4,6},{-3,8},{-1,9},{1,9},{3,8},{4,7},{6,4},
};
```

continues

Listing C.4 Continued

```
static XPoint *char655[] = {
    seg0_655,
    NULL,
};
static int char_p655[] = {
    XtNumber(seg0_655),
};
static XPoint seg0_656[] = {
    {-3,4},{1,-1},{3,-4},{4,-6},{5,-9},{5,-11},{4,-12},{2,-11},
    {1,-9},{-1,-1},{-4,8},{-7,15},{-8,18},{-8,20},{-7,21},{-5,20},
    {-4,17},{-3,8},{-2,9},{0,9},{2,8},{3,7},{5,4},
};
static XPoint *char656[] = {
    seg0_656,
    NULL,
};
static int char_p656[] = {
    XtNumber(seg0_656),
};
static XPoint seg0_657[] = {
    {3,3},{2,1},{0,0},{-2,0},{-4,1},{-5,2},
    {-6,4},{-6,6},{-5,8},{-3,9},{-1,9},{1,8},{2,7},
};
static XPoint seg1_657[] = {
    {4,0},{2,7},{-2,18},{-3,20},{-5,21},{-6,20},{-6,18},{-5,15},
    {-2,12},{1,10},{3,9},{6,7},{9,4},
};
static XPoint *char657[] = {
    seg0_657,seg1_657,
    NULL,
};
static int char_p657[] = {
    XtNumber(seg0_657),XtNumber(seg1_657),
};
static XPoint seg0_658[] = {
    {-5,4},{-3,1},{0,-4},{1,-6},
    {2,-9},{2,-11},{1,-12},{-1,-11},{-2,-9},{-3,-5},{-4,1},{-5,9},
};
static XPoint seg1_658[] = {
    {-5,9},{-4,6},{-3,4},{-1,1},{1,0},{3,0},{4,1},{4,3},
    {3,6},{3,8},{4,9},{5,9},{7,8},{8,7},{10,4},
};
static XPoint *char658[] = {
    seg0_658,seg1_658,
    NULL,
};
static int char_p658[] = {
    XtNumber(seg0_658),XtNumber(seg1_658),
};
static XPoint seg0_659[] = {
    {1,-5},{1,-4},
    {2,-4},{2,-5},{1,-5},
};
static XPoint seg1_659[] = {
    {-2,4},{0,0},{-2,6},{-2,8},{-1,9},{0,9},{2,8},{3,7},
    {5,4},
};
static XPoint *char659[] = {
    seg0_659,seg1_659,
    NULL,
};
static int char_p659[] = {
    XtNumber(seg0_659),XtNumber(seg1_659),
};
static XPoint seg0_660[] = {
    {1,-5},{1,-4},{2,-4},{2,-5},{1,-5},
};
static XPoint seg1_660[] = {
    {-2,4},{0,0},{-6,18},{-7,20},{-9,21},{-10,20},{-10,18},{-9,15},
    {-6,12},{-3,10},{-1,9},{2,7},{5,4},
};
static XPoint *char660[] = {
    seg0_660,seg1_660,
    NULL,
};
static int char_p660[] = {
    XtNumber(seg0_660),XtNumber(seg1_660),
};
static XPoint seg0_661[] = {
    {-5,4},{-3,1},{0,-4},{1,-6},
    {2,-9},{2,-11},{1,-12},{-1,-11},{-2,-9},{-3,-5},{-4,1},{-5,9},
};
static XPoint seg1_661[] = {
    {-5,9},{-4,6},{-3,4},{-1,1},{1,0},{3,0},{4,1},{4,3},
    {2,4},{-1,4},
};
static XPoint seg2_661[] = {
    {-1,4},{1,5},{2,8},{3,9},{4,9},{6,8},{7,7},{9,4},
};
static XPoint *char661[] = {
    seg0_661,seg1_661,seg2_661,
    NULL,
};
static int char_p661[] = {
    XtNumber(seg0_661),XtNumber(seg1_661),XtNumber(seg2_661),
};
static XPoint seg0_662[] = {
    {-3,4},{-1,1},{2,-4},{3,-6},{4,-9},{4,-11},{3,-12},{1,-11},
    {0,-9},{-1,-5},{-2,2},{-2,8},{-1,9},{0,9},{2,8},{3,7},{5,4},
};
static XPoint *char662[] = {
    seg0_662,
    NULL,
};
```

continues

Listing C.4 Continued

```
static int char_p662[] = {
    XtNumber(seg0_662),
};
static XPoint seg0_663[] = {
    {-13,4},
    {-11,1},{-9,0},{-8,1},{-8,2},{-9,6},{-10,9},
};
static XPoint seg1_663[] = {
    {-9,6},{-8,4},{-6,1},{-4,0},{-2,0},{-1,1},{-1,2},{-2,6},
    {-3,9},
};
static XPoint seg2_663[] = {
    {-2,6},{-1,4},{1,1},{3,0},{5,0},{6,1},{6,3},{5,6},
    {5,8},{6,9},{7,9},{9,8},{10,7},{12,4},
};
static XPoint *char663[] = {
    seg0_663,seg1_663,seg2_663,
    NULL,
};
static int char_p663[] = {
    XtNumber(seg0_663),XtNumber(seg1_663),XtNumber(seg2_663),
};
static XPoint seg0_664[] = {
    {-8,4},{-6,1},{-4,0},
    {-3,1},{-3,2},{-4,6},{-5,9},
};
static XPoint seg1_664[] = {
    {-4,6},{-3,4},{-1,1},{1,0},{3,0},{4,1},{4,3},{3,6},
    {3,8},{4,9},{5,9},{7,8},{8,7},{10,4},
};
static XPoint *char664[] = {
    seg0_664,seg1_664,
    NULL,
};
static int char_p664[] = {
    XtNumber(seg0_664),XtNumber(seg1_664),
};
static XPoint seg0_665[] = {
    {0,0},{-2,0},{-4,1},
    {-5,2},{-6,4},{-6,6},{-5,8},{-3,9},{-1,9},{1,8},{2,7},{3,5},
    {3,3},{2,1},{0,0},{-1,1},{-1,3},{0,5},{2,6},{5,6},{7,5},
    {8,4},
};
static XPoint *char665[] = {
    seg0_665,
    NULL,
};
static int char_p665[] = {
    XtNumber(seg0_665),
};
```

```
static XPoint seg0_666[] = {
    {-7,4},{-5,1},{-4,-1},{-5,3},{-11,21},
};
static XPoint seg1_666[] = {
    {-5,3},{-4,1},{-2,0},{0,0},{2,1},{3,3},{3,5},{2,7},
    {1,8},{-1,9},
};
static XPoint seg2_666[] = {
    {-5,8},{-3,9},{0,9},{3,8},{5,7},{8,4},
};
static XPoint *char666[] = {
    seg0_666,seg1_666,seg2_666,
    NULL,
};
static int char_p666[] = {
    XtNumber(seg0_666),XtNumber(seg1_666),XtNumber(seg2_666),
};
static XPoint seg0_667[] = {
    {3,3},{2,1},{0,0},{-2,0},{-4,1},{-5,2},{-6,4},{-6,6},
    {-5,8},{-3,9},{-1,9},{1,8},
};
static XPoint seg1_667[] = {
    {4,0},{3,3},{1,8},{-2,15},{-3,18},{-3,20},{-2,21},{0,20},
    {1,17},{1,10},{3,9},{6,7},{9,4},
};
static XPoint *char667[] = {
    seg0_667,seg1_667,
    NULL,
};
static int char_p667[] = {
    XtNumber(seg0_667),XtNumber(seg1_667),
};
static XPoint seg0_668[] = {
    {-5,4},{-3,1},{-2,-1},{-2,1},
    {1,1},{2,2},{2,4},{1,7},{1,8},{2,9},{3,9},{5,8},{6,7},
    {8,4},
};
static XPoint *char668[] = {
    seg0_668,
    NULL,
};
static int char_p668[] = {
    XtNumber(seg0_668),
};
static XPoint seg0_669[] = {
    {-4,4},{-2,1},{-1,-1},{-1,1},{1,4},{2,6},{2,8},{0,9},
};
static XPoint seg1_669[] = {
    {-4,8},{-2,9},{2,9},{4,8},{5,7},{7,4},
};
static XPoint *char669[] = {
    seg0_669,seg1_669,
    NULL,
};
```

continues

Listing C.4 Continued

```
static int char_p669[] = {
    XtNumber(seg0_669),XtNumber(seg1_669),
};
static XPoint seg0_670[] = {
    {-3,4},{-1,1},
    {1,-3},
};
static XPoint seg1_670[] = {
    {4,-12},{-2,6},{-2,8},{-1,9},{1,9},{3,8},{4,7},{6,4},
};
static XPoint seg2_670[] = {
    {-2,-4},{5,-4},
};
static XPoint *char670[] = {
    seg0_670,seg1_670,seg2_670,
    NULL,
};
static int char_p670[] = {
    XtNumber(seg0_670),XtNumber(seg1_670),XtNumber(seg2_670),
};
static XPoint seg0_671[] = {
    {-6,4},{-4,0},{-6,6},{-6,8},{-5,9},{-3,9},
    {-1,8},{1,6},{3,3},
};
static XPoint seg1_671[] = {
    {4,0},{2,6},{2,8},{3,9},{4,9},{6,8},{7,7},{9,4},
};
static XPoint *char671[] = {
    seg0_671,seg1_671,
    NULL,
};
static int char_p671[] = {
    XtNumber(seg0_671),XtNumber(seg1_671),
};
static XPoint seg0_672[] = {
    {-6,4},{-4,0},{-5,5},{-5,8},{-4,9},{-3,9},{0,8},{2,6},{3,3},
    {3,0},
};
static XPoint seg1_672[] = {
    {3,0},{4,4},{5,5},{7,5},{9,4},
};
static XPoint *char672[] = {
    seg0_672,seg1_672,
    NULL,
};
static int char_p672[] = {
    XtNumber(seg0_672),XtNumber(seg1_672),
};
static XPoint seg0_673[] = {
    {-6,0},{-8,2},{-9,5},
    {-9,7},{-8,9},{-6,9},{-4,8},{-2,6},
};
static XPoint seg1_673[] = {
    {0,0},{-2,6},{-2,8},{-1,9},{1,9},{3,8},{5,6},{6,3},
    {6,0},
};
static XPoint seg2_673[] = {
    {6,0},{7,4},{8,5},{10,5},{12,4},
};
static XPoint *char673[] = {
    seg0_673,seg1_673,seg2_673,
    NULL,
};
static int char_p673[] = {
    XtNumber(seg0_673),XtNumber(seg1_673),XtNumber(seg2_673),
};
static XPoint seg0_674[] = {
    {-8,4},{-6,1},{-4,0},
    {-2,0},{-1,1},{-1,8},{0,9},{3,9},{6,7},{8,4},
};
static XPoint seg1_674[] = {
    {5,1},{4,0},{2,0},{1,1},{-3,8},{-4,9},{-6,9},{-7,8},
};
static XPoint *char674[] = {
    seg0_674,seg1_674,
    NULL,
};
static int char_p674[] = {
    XtNumber(seg0_674),XtNumber(seg1_674),
};
static XPoint seg0_675[] = {
    {-6,4},{-4,0},{-6,6},{-6,8},{-5,9},{-3,9},{-1,8},{1,6},{3,3},
};
static XPoint seg1_675[] = {
    {4,0},{-2,18},{-3,20},{-5,21},{-6,20},{-6,18},{-5,15},{-2,12},
    {1,10},{3,9},{6,7},{9,4},
};
static XPoint *char675[] = {
    seg0_675,seg1_675,
    NULL,
};
static int char_p675[] = {
    XtNumber(seg0_675),XtNumber(seg1_675),
};
static XPoint seg0_676[] = {
    {-6,4},{-4,1},{-2,0},{0,0},{2,2},{2,4},{1,6},{-1,8},{-4,9},
    {-2,10},{-1,12},{-1,15},{-2,18},{-3,20},{-5,21},{-6,20},
    {-6,18},{-5,15},{-2,12},{1,10},{5,7},{8,4},
};
static XPoint *char676[] = {
    seg0_676,
    NULL,
};
```

continues

Listing C.4 Continued

```
static int char_p676[] = {
    XtNumber(seg0_676),
};
static XPoint seg0_699[] = {
    {-8,8},
};
static XPoint *char699[] = {
    seg0_699,
    NULL,
};
static int char_p699[] = {
    XtNumber(seg0_699),
};
static XPoint seg0_700[] = {
    {-1,-12},{-4,-11},{-6,-8},{-7,-3},{-7,0},{-6,5},{-4,8},{-1,9},
    {1,9},{4,8},{6,5},{7,0},{7,-3},{6,-8},{4,-11},{1,-12},{-1,-12},
};
static XPoint *char700[] = {
    seg0_700,
    NULL,
};
static int char_p700[] = {
    XtNumber(seg0_700),
};
static XPoint seg0_701[] = {
    {-4,-8},
    {-2,-9},{1,-12},{1,9},
};
static XPoint *char701[] = {
    seg0_701,
    NULL,
};
static int char_p701[] = {
    XtNumber(seg0_701),
};
static XPoint seg0_702[] = {
    {-6,-7},{-6,-8},{-5,-10},{-4,-11},{-2,-12},{2,-12},
    {4,-11},{5,-10},{6,-8},{6,-6},{5,-4},{3,-1},{-7,9},{7,9},
};
static XPoint *char702[] = {
    seg0_702,
    NULL,
};
static int char_p702[] = {
    XtNumber(seg0_702),
};
static XPoint seg0_703[] = {
    {-5,-12},{6,-12},{0,-4},{3,-4},{5,-3},{6,-2},{7,1},{7,3},
    {6,6},{4,8},{1,9},{-2,9},{-5,8},{-6,7},{-7,5},
};
```

```
static XPoint *char703[] = {
    seg0_703,
    NULL,
};
static int char_p703[] = {
    XtNumber(seg0_703),
};
static XPoint seg0_704[] = {
    {3,-12},{-7,2},{8,2},
};
static XPoint seg1_704[] = {
    {3,-12},{3,9},
};
static XPoint *char704[] = {
    seg0_704,seg1_704,
    NULL,
};
static int char_p704[] = {
    XtNumber(seg0_704),XtNumber(seg1_704),
};
static XPoint seg0_705[] = {
    {5,-12},{-5,-12},{-6,-3},{-5,-4},{-2,-5},{1,-5},
    {4,-4},{6,-2},{7,1},{7,3},{6,6},{4,8},{1,9},{-2,9},{-5,8},
    {-6,7},{-7,5},
};
static XPoint *char705[] = {
    seg0_705,
    NULL,
};
static int char_p705[] = {
    XtNumber(seg0_705),
};
static XPoint seg0_706[] = {
    {6,-9},{5,-11},{2,-12},{0,-12},{-3,-11},{-5,-8},{-6,-3},
    {-6,2},{-5,6},{-3,8},{0,9},{1,9},{4,8},{6,6},{7,3},{7,2},
    {6,-1},{4,-3},{1,-4},{0,-4},{-3,-3},{-5,-1},{-6,2},
};
static XPoint *char706[] = {
    seg0_706,
    NULL,
};
static int char_p706[] = {
    XtNumber(seg0_706),
};
static XPoint seg0_707[] = {
    {7,-12},{-3,9},
};
static XPoint seg1_707[] = {
    {-7,-12},{7,-12},
};
static XPoint *char707[] = {
    seg0_707,seg1_707,
    NULL,
};
```

continues

Listing C.4 Continued

```
static int char_p707[] = {
    XtNumber(seg0_707),XtNumber(seg1_707),
};
static XPoint seg0_708[] = {
    {-2,-12},{-5,-11},{-6,-9},{-6,-7},{-5,-5},{-3,-4},
    {1,-3},{4,-2},{6,0},{7,2},{7,5},{6,7},{5,8},{2,9},{-2,9},
    {-5,8},{-6,7},{-7,5},{-7,2},{-6,0},{-4,-2},{-1,-3},{3,-4},
    {5,-5},{6,-7},{6,-9},{5,-11},{2,-12},{-2,-12},
};
static XPoint *char708[] = {
    seg0_708,
    NULL,
};
static int char_p708[] = {
    XtNumber(seg0_708),
};
static XPoint seg0_709[] = {
    {6,-5},{5,-2},{3,0},{0,1},{-1,1},{-4,0},{-6,-2},{-7,-5},
    {-7,-6},{-6,-9},{-4,-11},{-1,-12},{0,-12},{3,-11},{5,-9},
    {6,-5},{6,0},{5,5},{3,8},{0,9},{-2,9},{-5,8},{-6,6},
};
static XPoint *char709[] = {
    seg0_709,
    NULL,
};
static int char_p709[] = {
    XtNumber(seg0_709),
};
static XPoint seg0_710[] = {
    {0,7},{-1,8},{0,9},{1,8},{0,7},
};
static XPoint *char710[] = {
    seg0_710,
    NULL,
};
static int char_p710[] = {
    XtNumber(seg0_710),
};
static XPoint seg0_711[] = {
    {1,8},{0,9},{-1,8},
    {0,7},{1,8},{1,10},{0,12},{-1,13},
};
static XPoint *char711[] = {
    seg0_711,
    NULL,
};
static int char_p711[] = {
    XtNumber(seg0_711),
};
```

```
static XPoint seg0_712[] = {
    {0,-5},{-1,-4},{0,-3},{1,-4},
    {0,-5},
};
static XPoint seg1_712[] = {
    {0,7},{-1,8},{0,9},{1,8},{0,7},
};
static XPoint *char712[] = {
    seg0_712,seg1_712,
    NULL,
};
static int char_p712[] = {
    XtNumber(seg0_712),XtNumber(seg1_712),
};
static XPoint seg0_713[] = {
    {0,-5},{-1,-4},{0,-3},
    {1,-4},{0,-5},
};
static XPoint seg1_713[] = {
    {1,8},{0,9},{-1,8},{0,7},{1,8},{1,10},{0,12},{-1,13},
};
static XPoint *char713[] = {
    seg0_713,seg1_713,
    NULL,
};
static int char_p713[] = {
    XtNumber(seg0_713),XtNumber(seg1_713),
};
static XPoint seg0_714[] = {
    {0,-12},{0,2},
};
static XPoint seg1_714[] = {
    {0,7},{-1,8},{0,9},{1,8},{0,7},
};
static XPoint *char714[] = {
    seg0_714,seg1_714,
    NULL,
};
static int char_p714[] = {
    XtNumber(seg0_714),XtNumber(seg1_714),
};
static XPoint seg0_715[] = {
    {-6,-7},{-6,-8},{-5,-10},{-4,-11},{-2,-12},{2,-12},{4,-11},
    {5,-10},{6,-8},{6,-6},{5,-4},{4,-3},{0,-1},{0,2},
};
static XPoint seg1_715[] = {
    {0,7},{-1,8},{0,9},{1,8},{0,7},
};
static XPoint *char715[] = {
    seg0_715,seg1_715,
    NULL,
};
static int char_p715[] = {
    XtNumber(seg0_715),XtNumber(seg1_715),
};
```

continues

Listing C.4 Continued

```
static XPoint seg0_717[] = {
    {-4,-12},
    {-4,-5},
};
static XPoint seg1_717[] = {
    {4,-12},{4,-5},
};
static XPoint *char717[] = {
    seg0_717,seg1_717,
    NULL,
};
static int char_p717[] = {
    XtNumber(seg0_717),XtNumber(seg1_717),
};
static XPoint seg0_718[] = {
    {-1,-12},{-3,-11},{-4,-9},{-4,-7},{-3,-5},{-1,-4},
    {1,-4},{3,-5},{4,-7},{4,-9},{3,-11},{1,-12},{-1,-12},
};
static XPoint *char718[] = {
    seg0_718,
    NULL,
};
static int char_p718[] = {
    XtNumber(seg0_718),
};
static XPoint seg0_719[] = {
    {-2,-16},{-2,13},
};
static XPoint seg1_719[] = {
    {2,-16},{2,13},
};
static XPoint seg2_719[] = {
    {7,-9},{5,-11},{2,-12},{-2,-12},{-5,-11},{-7,-9},{-7,-7},
    {-6,-5},{-5,-4},{-3,-3},{3,-1},{5,0},{6,1},{7,3},{7,6},
    {5,8},{2,9},{-2,9},{-5,8},{-7,6},
};
static XPoint *char719[] = {
    seg0_719,seg1_719,seg2_719,
    NULL,
};
static int char_p719[] = {
    XtNumber(seg0_719),XtNumber(seg1_719),XtNumber(seg2_719),
};
static XPoint seg0_720[] = {
    {9,-16},{-9,16},
};
static XPoint *char720[] = {
    seg0_720,
    NULL,
};
```

```
static int char_p720[] = {
    XtNumber(seg0_720),
};
static XPoint seg0_721[] = {
    {4,-16},{2,-14},{0,-11},{-2,-7},
    {-3,-2},{-3,2},{-2,7},{0,11},{2,14},{4,16},
};
static XPoint *char721[] = {
    seg0_721,
    NULL,
};
static int char_p721[] = {
    XtNumber(seg0_721),
};
static XPoint seg0_722[] = {
    {-4,-16},{-2,-14},{0,-11},
    {2,-7},{3,-2},{3,2},{2,7},{0,11},{-2,14},{-4,16},
};
static XPoint *char722[] = {
    seg0_722,
    NULL,
};
static int char_p722[] = {
    XtNumber(seg0_722),
};
static XPoint seg0_723[] = {
    {0,-16},{0,16},
};
static XPoint *char723[] = {
    seg0_723,
    NULL,
};
static int char_p723[] = {
    XtNumber(seg0_723),
};
static XPoint seg0_724[] = {
    {-9,0},{9,0},
};
static XPoint *char724[] = {
    seg0_724,
    NULL,
};
static int char_p724[] = {
    XtNumber(seg0_724),
};
static XPoint seg0_725[] = {
    {0,-9},{0,9},
};
static XPoint seg1_725[] = {
    {-9,0},{9,0},
};
```

C

continues

Listing C.4 Continued

```
static XPoint *char725[] = {
    seg0_725,seg1_725,
    NULL,
};
static int char_p725[] = {
    XtNumber(seg0_725),XtNumber(seg1_725),
};
static XPoint seg0_726[] = {
    {-9,-3},{9,-3},
};
static XPoint seg1_726[] = {
    {-9,3},{9,3},
};
static XPoint *char726[] = {
    seg0_726,seg1_726,
    NULL,
};
static int char_p726[] = {
    XtNumber(seg0_726),XtNumber(seg1_726),
};
static XPoint seg0_730[] = {
    {1,-12},
    {0,-11},{-1,-9},{-1,-7},{0,-6},{1,-7},{0,-8},
};
static XPoint *char730[] = {
    seg0_730,
    NULL,
};
static int char_p730[] = {
    XtNumber(seg0_730),
};
static XPoint seg0_731[] = {
    {0,-10},{-1,-11},{0,-12},
    {1,-11},{1,-9},{0,-7},{-1,-6},
};
static XPoint *char731[] = {
    seg0_731,
    NULL,
};
static int char_p731[] = {
    XtNumber(seg0_731),
};
static XPoint seg0_733[] = {
    {1,-16},{-6,16},
};
static XPoint seg1_733[] = {
    {7,-16},{0,16},
};
static XPoint seg2_733[] = {
    {-6,-3},{8,-3},
};
```

```
static XPoint seg3_733[] = {
    {-7,3},{7,3},
};
static XPoint *char733[] = {
    seg0_733,seg1_733,seg2_733,seg3_733,
    NULL,
};
static int char_p733[] = {
    XtNumber(seg0_733),XtNumber(seg1_733),XtNumber(seg2_733),
    XtNumber(seg3_733),
};
static XPoint seg0_734[] = {
    {10,-3},{10,-4},{9,-5},{8,-5},{7,-4},{6,-2},{4,3},{2,6},{0,8},
    {-2,9},{-6,9},{-8,8},{-9,7},{-10,5},{-10,3},{-9,1},{-8,0},
    {-1,-4},{0,-5},{1,-7},{1,-9},{0,-11},{-2,-12},{-4,-11},
    {-5,-9},{-5,-7},{-4,-4},{-2,-1},{3,6},{5,8},{7,9},{9,9},{10,8},
    {10,7},
};
static XPoint *char734[] = {
    seg0_734,
    NULL,
};
static int char_p734[] = {
    XtNumber(seg0_734),
};
static XPoint seg0_804[] = {
    {-7,-12},{7,12},
};
static XPoint *char804[] = {
    seg0_804,
    NULL,
};
static int char_p804[] = {
    XtNumber(seg0_804),
};
static XPoint seg0_834[] = {
    {-14,0},{14,0},
};
static XPoint seg1_834[] = {
    {-14,0},{0,16},
};
static XPoint seg2_834[] = {
    {14,0},{0,16},
};
static XPoint *char834[] = {
    seg0_834,seg1_834,seg2_834,
    NULL,
};
static int char_p834[] = {
    XtNumber(seg0_834),XtNumber(seg1_834),XtNumber(seg2_834),
};
static XPoint seg0_840[] = {
    {-1,-7},{-4,-6},{-6,-4},{-7,-1},{-7,1},{-6,4},
```

continues

Listing C.4 Continued

```
    {-4,6},{-1,7},{1,7},{4,6},{6,4},{7,1},{7,-1},{6,-4},{4,-6},
    {1,-7},{-1,-7},
};
static XPoint *char840[] = {
    seg0_840,
    NULL,
};
static int char_p840[] = {
    XtNumber(seg0_840),
};
static XPoint seg0_844[] = {
    {0,-9},{-2,-3},
    {-8,-3},{-3,1},{-5,7},{0,3},{5,7},{3,1},{8,-3},{2,-3},{0,-9},
};
static XPoint *char844[] = {
    seg0_844,
    NULL,
};
static int char_p844[] = {
    XtNumber(seg0_844),
};
static XPoint seg0_845[] = {
    {0,-7},{0,7},
};
static XPoint seg1_845[] = {
    {-7,0},{7,0},
};
static XPoint *char845[] = {
    seg0_845,seg1_845,
    NULL,
};
static int char_p845[] = {
    XtNumber(seg0_845),XtNumber(seg1_845),
};
static XPoint seg0_847[] = {
    {0,-6},{0,6},
};
static XPoint seg1_847[] = {
    {-5,-3},{5,3},
};
static XPoint seg2_847[] = {
    {5,-3},{-5,3},
};
static XPoint *char847[] = {
    seg0_847,seg1_847,seg2_847,
    NULL,
};
static int char_p847[] = {
    XtNumber(seg0_847),XtNumber(seg1_847),XtNumber(seg2_847),
};
```

```
static XPoint seg0_850[] = {
    {-1,-4},{-3,-3},{-4,-1},{-4,1},{-3,3},{-1,4},
    {1,4},{3,3},{4,1},{4,-1},{3,-3},{1,-4},{-1,-4},
};
static XPoint seg1_850[] = {
    {-3,-1},{-3,1},
};
static XPoint seg2_850[] = {
    {-2,-2},{-2,2},
};
static XPoint seg3_850[] = {
    {-1,-3},{-1,3},
};
static XPoint seg4_850[] = {
    {0,-3},{0,3},
};
static XPoint seg5_850[] = {
    {1,-3},{1,3},
};
static XPoint seg6_850[] = {
    {2,-2},{2,2},
};
static XPoint seg7_850[] = {
    {3,-1},{3,1},
};
static XPoint *char850[] = {
    seg0_850,seg1_850,seg2_850,seg3_850,seg4_850,seg5_850,
    seg6_850,seg7_850,
    NULL,
};
static int char_p850[] = {
    XtNumber(seg0_850),XtNumber(seg1_850),XtNumber(seg2_850),
    XtNumber(seg3_850),XtNumber(seg4_850),XtNumber(seg5_850),
    XtNumber(seg6_850),XtNumber(seg7_850),
};
static XPoint seg0_855[] = {
    {6,0},{-3,-5},{-3,5},{6,0},
};
static XPoint seg1_855[] = {
    {3,0},{-2,-3},
};
static XPoint seg2_855[] = {
    {3,0},{-2,3},
};
static XPoint seg3_855[] = {
    {0,0},{-2,-1},
};
static XPoint seg4_855[] = {
    {0,0},{-2,1},
};
static XPoint *char855[] = {
    seg0_855,seg1_855,seg2_855,seg3_855,seg4_855,
    NULL,
};
```

continues

Listing C.4 Continued

```
static int char_p855[] = {
    XtNumber(seg0_855),XtNumber(seg1_855),XtNumber(seg2_855),
    XtNumber(seg3_855),XtNumber(seg4_855),
};
static XPoint seg0_866[] = {
    {-2,-6},{-2,-2},{-6,-2},{-6,2},
    {-2,2},{-2,6},{2,6},{2,2},{6,2},{6,-2},{2,-2},{2,-6},{-2,-6},
};
static XPoint *char866[] = {
    seg0_866,
    NULL,
};
static int char_p866[] = {
    XtNumber(seg0_866),
};
static XPoint seg0_999[] = {
    {-8,11},{8,11},
};
static XPoint *char999[] = {
    seg0_999,
    NULL,
};
static int char_p999[] = {
    XtNumber(seg0_999),
};
static XPoint seg0_2219[] = {
    {0,-12},{0,0},
};
static XPoint seg1_2219[] = {
    {-5,-9},{5,-3},
};
static XPoint seg2_2219[] = {
    {5,-9},{-5,-3},
};
static XPoint *char2219[] = {
    seg0_2219,seg1_2219,seg2_2219,
    NULL,
};
static int char_p2219[] = {
    XtNumber(seg0_2219),XtNumber(seg1_2219),XtNumber(seg2_2219),
};
static XPoint seg0_2223[] = {
    {-3,-16},{-3,16},
};
static XPoint seg1_2223[] = {
    {-2,-16},{-2,16},
};
static XPoint seg2_2223[] = {
    {-3,-16},{4,-16},
};
```

```
static XPoint seg3_2223[] = {
    {-3,16},{4,16},
};
static XPoint *char2223[] = {
    seg0_2223,seg1_2223,seg2_2223,seg3_2223,
    NULL,
};
static int char_p2223[] = {
    XtNumber(seg0_2223),XtNumber(seg1_2223),XtNumber(seg2_2223),
    XtNumber(seg3_2223),
};
static XPoint seg0_2224[] = {
    {2,-16},{2,16},
};
static XPoint seg1_2224[] = {
    {3,-16},{3,16},
};
static XPoint seg2_2224[] = {
    {-4,-16},{3,-16},
};
static XPoint seg3_2224[] = {
    {-4,16},{3,16},
};
static XPoint *char2224[] = {
    seg0_2224,seg1_2224,seg2_2224,seg3_2224,
    NULL,
};
static int char_p2224[] = {
    XtNumber(seg0_2224),XtNumber(seg1_2224),XtNumber(seg2_2224),
    XtNumber(seg3_2224),
};
static XPoint seg0_2225[] = {
    {2,-16},{0,-15},{-1,-14},{-2,-12},{-2,-10},{-1,-8},
    {0,-7},{1,-5},{1,-3},{-1,-1},
};
static XPoint seg1_2225[] = {
    {0,-15},{-1,-13},{-1,-11},{0,-9},{1,-8},{2,-6},{2,-4},{1,-2},
    {-3,0},{1,2},{2,4},{2,6},{1,8},{0,9},{-1,11},{-1,13},{0,15},
};
static XPoint seg2_2225[] = {
    {-1,1},{1,3},{1,5},{0,7},{-1,8},{-2,10},{-2,12},{-1,14},
    {0,15},{2,16},
};
static XPoint *char2225[] = {
    seg0_2225,seg1_2225,seg2_2225,
    NULL,
};
static int char_p2225[] = {
    XtNumber(seg0_2225),XtNumber(seg1_2225),XtNumber(seg2_2225),
};
static XPoint seg0_2226[] = {
    {-2,-16},{0,-15},{1,-14},{2,-12},{2,-10},{1,-8},{0,-7},
    {-1,-5},{-1,-3},{1,-1},
};
```

continues

Listing C.4 Continued

```
static XPoint seg1_2226[] = {
    {0,-15},{1,-13},{1,-11},{0,-9},{-1,-8},{-2,-6},{-2,-4},{-1,-2},
    {3,0},{-1,2},{-2,4},{-2,6},{-1,8},{0,9},{1,11},{1,13},{0,15},
};
static XPoint seg2_2226[] = {
    {1,1},{-1,3},{-1,5},{0,7},{1,8},{2,10},{2,12},{1,14},
    {0,15},{-2,16},
};
static XPoint *char2226[] = {
    seg0_2226,seg1_2226,seg2_2226,
    NULL,
};
static int char_p2226[] = {
    XtNumber(seg0_2226),XtNumber(seg1_2226),XtNumber(seg2_2226),
};
static XPoint seg0_2229[] = {
    {0,-16},
    {0,16},
};
static XPoint *char2229[] = {
    seg0_2229,
    NULL,
};
static int char_p2229[] = {
    XtNumber(seg0_2229),
};
static XPoint seg0_2241[] = {
    {8,-9},{-8,0},{8,9},
};
static XPoint *char2241[] = {
    seg0_2241,
    NULL,
};
static int char_p2241[] = {
    XtNumber(seg0_2241),
};
static XPoint seg0_2242[] = {
    {-8,-9},{8,0},{-8,9},
};
static XPoint *char2242[] = {
    seg0_2242,
    NULL,
};
static int char_p2242[] = {
    XtNumber(seg0_2242),
};
static XPoint seg0_2246[] = {
    {-9,3},{-9,1},{-8,-2},{-6,-3},
    {-4,-3},{-2,-2},{2,1},{4,2},{6,2},{8,1},{9,-1},
};
```

```
static XPoint seg1_2246[] = {
    {-9,1},{-8,-1},{-6,-2},{-4,-2},{-2,-1},{2,2},{4,3},{6,3},
    {8,2},{9,-1},{9,-3},
};
static XPoint *char2246[] = {
    seg0_2246,seg1_2246,
    NULL,
};
static int char_p2246[] = {
    XtNumber(seg0_2246),XtNumber(seg1_2246),
};
static XPoint seg0_2262[] = {
    {-2,-6},{0,-9},{2,-6},
};
static XPoint seg1_2262[] = {
    {-5,-3},{0,-8},{5,-3},
};
static XPoint seg2_2262[] = {
    {0,-8},{0,9},
};
static XPoint *char2262[] = {
    seg0_2262,seg1_2262,seg2_2262,
    NULL,
};
static int char_p2262[] = {
    XtNumber(seg0_2262),XtNumber(seg1_2262),XtNumber(seg2_2262),
};
static XPoint seg0_2271[] = {
    {9,-12},{-9,9},
};
static XPoint seg1_2271[] = {
    {-4,-12},{-2,-10},{-2,-8},{-3,-6},{-5,-5},{-7,-5},{-9,-7},
    {-9,-9},{-8,-11},{-6,-12},{-4,-12},{-2,-11},{1,-10},{4,-10},
    {7,-11},{9,-12},
};
static XPoint seg2_2271[] = {
    {5,2},{3,3},{2,5},{2,7},{4,9},{6,9},{8,8},{9,6},
    {9,4},{7,2},{5,2},
};
static XPoint *char2271[] = {
    seg0_2271,seg1_2271,seg2_2271,
    NULL,
};
static int char_p2271[] = {
    XtNumber(seg0_2271),XtNumber(seg1_2271),XtNumber(seg2_2271),
};
static XPoint seg0_2273[] = {
    {5,-4},{4,-6},{2,-7},{-1,-7},{-3,-6},{-4,-5},{-5,-2},
    {-5,1},{-4,3},{-2,4},{1,4},{3,3},{4,1},
};
static XPoint seg1_2273[] = {
    {-1,-7},{-3,-5},{-4,-2},{-4,1},{-3,3},{-2,4},
};
```

continues

Listing C.4 Continued

```
static XPoint seg2_2273[] = {
    {5,-7},{4,1},{4,3},{6,4},{8,4},{10,2},{11,-1},{11,-3},
    {10,-6},{9,-8},{7,-10},{5,-11},{2,-12},{-1,-12},{-4,-11},
    {-6,-10},{-8,-8},{-9,-6},{-10,-3},{-10,0},{-9,3},{-8,5},
    {-6,7},{-4,8},{-1,9},{2,9},{5,8},{7,7},{8,6},
};
static XPoint seg3_2273[] = {
    {6,-7},{5,1},{5,3},{6,4},
};
static XPoint *char2273[] = {
    seg0_2273,seg1_2273,seg2_2273,seg3_2273,
    NULL,
};
static int char_p2273[] = {
    XtNumber(seg0_2273),XtNumber(seg1_2273),XtNumber(seg2_2273),
    XtNumber(seg3_2273),
};
static XPoint seg0_2275[] = {
    {1,-12},{-6,16},
};
static XPoint seg1_2275[] = {
    {7,-12},{0,16},
};
static XPoint seg2_2275[] = {
    {-6,-1},{8,-1},
};
static XPoint seg3_2275[] = {
    {-7,5},{7,5},
};
static XPoint *char2275[] = {
    seg0_2275,seg1_2275,seg2_2275,seg3_2275,
    NULL,
};
static int char_p2275[] = {
    XtNumber(seg0_2275),XtNumber(seg1_2275),XtNumber(seg2_2275),
    XtNumber(seg3_2275),
};
static XPoint seg0_2750[] = {
    {2,-12},{-1,-11},{-3,-9},{-5,-6},{-6,-3},{-7,1},{-7,4},
    {-6,7},{-5,8},{-3,9},{-1,9},{2,8},{4,6},{6,3},{7,0},{8,-4},
    {8,-7},{7,-10},{6,-11},{4,-12},{2,-12},
};
static XPoint seg1_2750[] = {
    {2,-12},{0,-11},{-2,-9},{-4,-6},{-5,-3},{-6,1},{-6,4},{-5,7},
    {-3,9},
};
static XPoint seg2_2750[] = {
    {-1,9},{1,8},{3,6},{5,3},{6,0},{7,-4},{7,-7},{6,-10},
    {4,-12},
};
```

```
static XPoint *char2750[] = {
    seg0_2750,seg1_2750,seg2_2750,
    NULL,
};
static int char_p2750[] = {
    XtNumber(seg0_2750),XtNumber(seg1_2750),XtNumber(seg2_2750),
};
static XPoint seg0_2751[] = {
    {2,-8},{-3,9},
};
static XPoint seg1_2751[] = {
    {4,-12},{-2,9},
};
static XPoint seg2_2751[] = {
    {4,-12},{1,-9},{-2,-7},{-4,-6},
};
static XPoint seg3_2751[] = {
    {3,-9},{-1,-7},{-4,-6},
};
static XPoint *char2751[] = {
    seg0_2751,seg1_2751,seg2_2751,seg3_2751,
    NULL,
};
static int char_p2751[] = {
    XtNumber(seg0_2751),XtNumber(seg1_2751),XtNumber(seg2_2751),
    XtNumber(seg3_2751),
};
static XPoint seg0_2752[] = {
    {-3,-8},{-2,-7},{-3,-6},{-4,-7},{-4,-8},{-3,-10},{-2,-11},
    {1,-12},{4,-12},{7,-11},{8,-9},{8,-7},{7,-5},{5,-3},
    {2,-1},{-2,1},{-5,3},{-7,5},{-9,9},
};
static XPoint seg1_2752[] = {
    {4,-12},{6,-11},{7,-9},{7,-7},{6,-5},{4,-3},{-2,1},
};
static XPoint seg2_2752[] = {
    {-8,7},{-7,6},{-5,6},{0,8},{3,8},{5,7},{6,5},
};
static XPoint seg3_2752[] = {
    {-5,6},{0,9},{3,9},{5,8},{6,5},
};
static XPoint *char2752[] = {
    seg0_2752,seg1_2752,seg2_2752,seg3_2752,
    NULL,
};
static int char_p2752[] = {
    XtNumber(seg0_2752),XtNumber(seg1_2752),XtNumber(seg2_2752),
    XtNumber(seg3_2752),
};
static XPoint seg0_2753[] = {
    {-3,-8},{-2,-7},{-3,-6},{-4,-7},{-4,-8},{-3,-10},{-2,-11},
    {1,-12},{4,-12},{7,-11},{8,-9},{8,-7},{7,-5},{4,-3},{1,-2},
};
```

continues

Listing C.4 Continued

```
static XPoint seg1_2753[] = {
    {4,-12},{6,-11},{7,-9},{7,-7},{6,-5},{4,-3},
};
static XPoint seg2_2753[] = {
    {-1,-2},{1,-2},{4,-1},{5,0},{6,2},{6,5},{5,7},{4,8},
    {1,9},{-3,9},{-6,8},{-7,7},{-8,5},{-8,4},{-7,3},{-6,4},{-7,5},
};
static XPoint seg3_2753[] = {
    {1,-2},{3,-1},{4,0},{5,2},{5,5},{4,7},{3,8},{1,9},
};
static XPoint *char2753[] = {
    seg0_2753,seg1_2753,seg2_2753,seg3_2753,
    NULL,
};
static int char_p2753[] = {
    XtNumber(seg0_2753),XtNumber(seg1_2753),XtNumber(seg2_2753),
    XtNumber(seg3_2753),
};
static XPoint seg0_2754[] = {
    {6,-11},{0,9},
};
static XPoint seg1_2754[] = {
    {7,-12},{1,9},
};
static XPoint seg2_2754[] = {
    {7,-12},{-8,3},{8,3},
};
static XPoint *char2754[] = {
    seg0_2754,seg1_2754,seg2_2754,
    NULL,
};
static int char_p2754[] = {
    XtNumber(seg0_2754),XtNumber(seg1_2754),XtNumber(seg2_2754),
};
static XPoint seg0_2755[] = {
    {-1,-12},{-6,-2},
};
static XPoint seg1_2755[] = {
    {-1,-12},{9,-12},
};
static XPoint seg2_2755[] = {
    {-1,-11},{4,-11},{9,-12},
};
static XPoint seg3_2755[] = {
    {-6,-2},{-5,-3},{-2,-4},{1,-4},{4,-3},{5,-2},{6,0},{6,3},
    {5,6},{3,8},{0,9},{-3,9},{-6,8},{-7,7},{-8,5},{-8,4},{-7,3},
    {-6,4},{-7,5},
};
```

```
static XPoint seg4_2755[] = {
    {1,-4},{3,-3},{4,-2},{5,0},{5,3},{4,6},{2,8},{0,9},
};
static XPoint *char2755[] = {
    seg0_2755,seg1_2755,seg2_2755,seg3_2755,seg4_2755,
    NULL,
};
static int char_p2755[] = {
    XtNumber(seg0_2755),XtNumber(seg1_2755),XtNumber(seg2_2755),
    XtNumber(seg3_2755),XtNumber(seg4_2755),
};
static XPoint seg0_2756[] = {
    {7,-9},{6,-8},{7,-7},{8,-8},{8,-9},{7,-11},{5,-12},{2,-12},
    {-1,-11},{-3,-9},{-5,-6},{-6,-3},{-7,1},{-7,5},{-6,7},{-5,8},
    {-3,9},{0,9},{3,8},{5,6},{6,4},{6,1},{5,-1},{4,-2},{2,-3},
    {-1,-3},{-3,-2},{-5,0},{-6,2},
};
static XPoint seg1_2756[] = {
    {2,-12},{0,-11},{-2,-9},{-4,-6},{-5,-3},{-6,1},{-6,6},{-5,8},
};
static XPoint seg2_2756[] = {
    {0,9},{2,8},{4,6},{5,4},{5,0},{4,-2},
};
static XPoint *char2756[] = {
    seg0_2756,seg1_2756,seg2_2756,
    NULL,
};
static int char_p2756[] = {
    XtNumber(seg0_2756),XtNumber(seg1_2756),XtNumber(seg2_2756),
};
static XPoint seg0_2757[] = {
    {-4,-12},{-6,-6},
};
static XPoint seg1_2757[] = {
    {9,-12},{8,-9},{6,-6},{1,0},{-1,3},{-2,5},{-3,9},
};
static XPoint seg2_2757[] = {
    {6,-6},{0,0},{-2,3},{-3,5},{-4,9},
};
static XPoint seg3_2757[] = {
    {-5,-9},{-2,-12},{0,-12},{5,-9},
};
static XPoint seg4_2757[] = {
    {-4,-10},{-2,-11},{0,-11},{5,-9},{7,-9},{8,-10},{9,-12},
};
static XPoint *char2757[] = {
    seg0_2757,seg1_2757,seg2_2757,seg3_2757,seg4_2757,
    NULL,
};
static int char_p2757[] = {
    XtNumber(seg0_2757),XtNumber(seg1_2757),XtNumber(seg2_2757),
    XtNumber(seg3_2757),XtNumber(seg4_2757),
};
```

continues

Listing C.4 Continued

```
static XPoint seg0_2758[] = {
    {1,-12},{-2,-11},{-3,-10},{-4,-8},{-4,-5},{-3,-3},{-1,-2},
    {2,-2},{6,-3},{7,-4},{8,-6},{8,-9},{7,-11},{4,-12},{1,-12},
};
static XPoint seg1_2758[] = {
    {1,-12},{-1,-11},{-2,-10},{-3,-8},{-3,-5},{-2,-3},{-1,-2},
};
static XPoint seg2_2758[] = {
    {2,-2},{5,-3},{6,-4},{7,-6},{7,-9},{6,-11},{4,-12},
};
static XPoint seg3_2758[] = {
    {-1,-2},{-5,-1},{-7,1},{-8,3},{-8,6},{-7,8},{-4,9},{0,9},
    {4,8},{5,7},{6,5},{6,2},{5,0},{4,-1},{2,-2},
};
static XPoint seg4_2758[] = {
    {-1,-2},{-4,-1},{-6,1},{-7,3},{-7,6},{-6,8},{-4,9},
};
static XPoint seg5_2758[] = {
    {0,9},{3,8},{4,7},{5,5},{5,1},{4,-1},
};
static XPoint *char2758[] = {
    seg0_2758,seg1_2758,seg2_2758,seg3_2758,seg4_2758,seg5_2758,
    NULL,
};
static int char_p2758[] = {
    XtNumber(seg0_2758),XtNumber(seg1_2758),XtNumber(seg2_2758),
    XtNumber(seg3_2758),XtNumber(seg4_2758),XtNumber(seg5_2758),
};
static XPoint seg0_2759[] = {
    {7,-5},{6,-3},{4,-1},{2,0},{-1,0},{-3,-1},{-4,-2},{-5,-4},
    {-5,-7},{-4,-9},{-2,-11},{1,-12},{4,-12},{6,-11},{7,-10},
    {8,-8},{8,-4},{7,0},{6,3},{4,6},{2,8},{-1,9},{-4,9},{-6,8},
    {-7,6},{-7,5},{-6,4},{-5,5},{-6,6},
};
static XPoint seg1_2759[] = {
    {-3,-1},{-4,-3},{-4,-7},{-3,-9},{-1,-11},{1,-12},
};
static XPoint seg2_2759[] = {
    {6,-11},{7,-9},{7,-4},{6,0},{5,3},{3,6},{1,8},{-1,9},
};
static XPoint *char2759[] = {
    seg0_2759,seg1_2759,seg2_2759,
    NULL,
};
static int char_p2759[] = {
    XtNumber(seg0_2759),XtNumber(seg1_2759),XtNumber(seg2_2759),
};
static XPoint seg0_2761[] = {
    {-2,9},{-3,8},{-2,7},{-1,8},
    {-1,9},{-2,11},{-4,13},
};
```

```
static XPoint *char2761[] = {
    seg0_2761,
    NULL,
};
static int char_p2761[] = {
    XtNumber(seg0_2761),
};
static XPoint seg0_2762[] = {
    {1,-5},{0,-4},{1,-3},{2,-4},{1,-5},
};
static XPoint seg1_2762[] = {
    {-2,7},{-3,8},{-2,9},{-1,8},
};
static XPoint *char2762[] = {
    seg0_2762,seg1_2762,
    NULL,
};
static int char_p2762[] = {
    XtNumber(seg0_2762),XtNumber(seg1_2762),
};
static XPoint seg0_2763[] = {
    {1,-5},{0,-4},{1,-3},{2,-4},
    {1,-5},
};
static XPoint seg1_2763[] = {
    {-2,9},{-3,8},{-2,7},{-1,8},{-1,9},{-2,11},{-4,13},
};
static XPoint *char2763[] = {
    seg0_2763,seg1_2763,
    NULL,
};
static int char_p2763[] = {
    XtNumber(seg0_2763),XtNumber(seg1_2763),
};
static XPoint seg0_2764[] = {
    {3,-12},
    {2,-11},{0,1},
};
static XPoint seg1_2764[] = {
    {3,-11},{0,1},
};
static XPoint seg2_2764[] = {
    {3,-12},{4,-11},{0,1},
};
static XPoint seg3_2764[] = {
    {-2,7},{-3,8},{-2,9},{-1,8},{-2,7},
};
static XPoint *char2764[] = {
    seg0_2764,seg1_2764,seg2_2764,seg3_2764,
    NULL,
};
static int char_p2764[] = {
    XtNumber(seg0_2764),XtNumber(seg1_2764),XtNumber(seg2_2764),
```

continues

Listing C.4 Continued

```
    XtNumber(seg3_2764),
};
static XPoint seg0_2765[] = {
    {-3,-8},{-2,-7},{-3,-6},{-4,-7},{-4,-8},{-3,-10},{-2,-11},
    {1,-12},{5,-12},{8,-11},{9,-9},{9,-7},{8,-5},{7,-4},{1,-2},
    {-1,-1},{-1,1},{0,2},{2,2},
};
static XPoint seg1_2765[] = {
    {5,-12},{7,-11},{8,-9},{8,-7},{7,-5},{6,-4},{4,-3},
};
static XPoint seg2_2765[] = {
    {-2,7},{-3,8},{-2,9},{-1,8},{-2,7},
};
static XPoint *char2765[] = {
    seg0_2765,seg1_2765,seg2_2765,
    NULL,
};
static int char_p2765[] = {
    XtNumber(seg0_2765),XtNumber(seg1_2765),XtNumber(seg2_2765),
};
static XPoint seg0_2766[] = {
    {4,-12},{2,-10},{1,-8},
    {1,-7},{2,-6},{3,-7},{2,-8},
};
static XPoint *char2766[] = {
    seg0_2766,
    NULL,
};
static int char_p2766[] = {
    XtNumber(seg0_2766),
};
static XPoint seg0_2767[] = {
    {3,-10},{2,-11},{3,-12},{4,-11},{4,-10},
    {3,-8},{1,-6},
};
static XPoint *char2767[] = {
    seg0_2767,
    NULL,
};
static int char_p2767[] = {
    XtNumber(seg0_2767),
};
static XPoint seg0_2768[] = {
    {10,-4},{9,-3},{10,-2},{11,-3},{11,-4},{10,-5},{9,-5},
    {7,-4},{5,-2},{0,6},{-2,8},{-4,9},{-7,9},{-10,8},{-11,6},
    {-11,4},{-10,2},{-9,1},{-7,0},{-2,-2},{0,-3},{2,-5},{3,-7},
    {3,-9},{2,-11},{0,-12},{-2,-11},{-3,-9},{-3,-6},{-2,0},{-1,3},
    {1,6},{3,8},{5,9},{7,9},{8,7},{8,6},
};
```

```
static XPoint seg1_2768[] = {
    {-7,9},{-9,8},{-10,6},{-10,4},{-9,2},{-8,1},{-2,-2},
};
static XPoint seg2_2768[] = {
    {-3,-6},{-2,-1},{-1,2},{1,5},{3,7},{5,8},{7,8},{8,7},
};
static XPoint *char2768[] = {
    seg0_2768,seg1_2768,seg2_2768,
    NULL,
};
static int char_p2768[] = {
    XtNumber(seg0_2768),XtNumber(seg1_2768),XtNumber(seg2_2768),
};
static XPoint seg0_2769[] = {
    {2,-16},{-6,13},
};
static XPoint seg1_2769[] = {
    {7,-16},{-1,13},
};
static XPoint seg2_2769[] = {
    {8,-8},{7,-7},{8,-6},{9,-7},{9,-8},{8,-10},{7,-11},{4,-12},
    {0,-12},{-3,-11},{-5,-9},{-5,-7},{-4,-5},{-3,-4},{4,0},{6,2},
};
static XPoint seg3_2769[] = {
    {-5,-7},{-3,-5},{4,-1},{5,0},{6,2},{6,5},{5,7},{4,8},
    {1,9},{-3,9},{-6,8},{-7,7},{-8,5},{-8,4},{-7,3},{-6,4},{-7,5},
};
static XPoint *char2769[] = {
    seg0_2769,seg1_2769,seg2_2769,seg3_2769,
    NULL,
};
static int char_p2769[] = {
    XtNumber(seg0_2769),XtNumber(seg1_2769),XtNumber(seg2_2769),
    XtNumber(seg3_2769),
};
static XPoint seg0_2770[] = {
    {13,-16},{-13,16},
};
static XPoint *char2770[] = {
    seg0_2770,
    NULL,
};
static int char_p2770[] = {
    XtNumber(seg0_2770),
};
static XPoint seg0_2771[] = {
    {8,-16},{4,-13},{1,-10},{-1,-7},{-3,-3},{-4,2},{-4,6},
    {-3,11},{-2,14},{-1,16},
};
static XPoint seg1_2771[] = {
    {4,-13},{1,-9},{-1,-5},{-2,-2},{-3,3},{-3,8},{-2,13},{-1,16},
};
static XPoint *char2771[] = {
    seg0_2771,seg1_2771,
```

continues

Listing C.4 Continued

```
    NULL,
};
static int char_p2771[] = {
    XtNumber(seg0_2771),XtNumber(seg1_2771),
};
static XPoint seg0_2772[] = {
    {1,-16},{2,-14},{3,-11},{4,-6},{4,-2},{3,3},{1,7},
    {-1,10},{-4,13},{-8,16},
};
static XPoint seg1_2772[] = {
    {1,-16},{2,-13},{3,-8},{3,-3},{2,2},{1,5},{-1,9},{-4,13},
};
static XPoint *char2772[] = {
    seg0_2772,seg1_2772,
    NULL,
};
static int char_p2772[] = {
    XtNumber(seg0_2772),XtNumber(seg1_2772),
};
static XPoint seg0_2773[] = {
    {2,-12},{2,0},
};
static XPoint seg1_2773[] = {
    {-3,-9},{7,-3},
};
static XPoint seg2_2773[] = {
    {7,-9},{-3,-3},
};
static XPoint *char2773[] = {
    seg0_2773,seg1_2773,seg2_2773,
    NULL,
};
static int char_p2773[] = {
    XtNumber(seg0_2773),XtNumber(seg1_2773),XtNumber(seg2_2773),
};
static XPoint seg0_2778[] = {
    {-2,-12},{-4,-5},
};
static XPoint seg1_2778[] = {
    {-1,-12},{-4,-5},
};
static XPoint seg2_2778[] = {
    {7,-12},{5,-5},
};
static XPoint seg3_2778[] = {
    {8,-12},{5,-5},
};
static XPoint *char2778[] = {
    seg0_2778,seg1_2778,seg2_2778,seg3_2778,
    NULL,
};
```

```
static int char_p2778[] = {
    XtNumber(seg0_2778),XtNumber(seg1_2778),XtNumber(seg2_2778),
    XtNumber(seg3_2778),
};
static XPoint seg0_3001[] = {
    {0,-12},{-7,8},
};
static XPoint seg1_3001[] = {
    {-1,-9},{5,9},
};
static XPoint seg2_3001[] = {
    {0,-9},{6,9},
};
static XPoint seg3_3001[] = {
    {0,-12},{7,9},
};
static XPoint seg4_3001[] = {
    {-5,3},{4,3},
};
static XPoint seg5_3001[] = {
    {-9,9},{-3,9},
};
static XPoint seg6_3001[] = {
    {2,9},{9,9},
};
static XPoint seg7_3001[] = {
    {-7,8},{-8,9},
};
static XPoint seg8_3001[] = {
    {-7,8},{-5,9},
};
static XPoint seg9_3001[] = {
    {5,8},{3,9},
};
static XPoint seg10_3001[] = {
    {5,7},{4,9},
};
static XPoint seg11_3001[] = {
    {6,7},{8,9},
};
static XPoint *char3001[] = {
    seg0_3001,seg1_3001,seg2_3001,seg3_3001,seg4_3001,
    seg5_3001,seg6_3001,seg7_3001,seg8_3001,seg9_3001,
    seg10_3001,seg11_3001,
    NULL,
};
static int char_p3001[] = {
    XtNumber(seg0_3001),XtNumber(seg1_3001), XtNumber(seg2_3001),
    XtNumber(seg3_3001),XtNumber(seg4_3001), XtNumber(seg5_3001),
    XtNumber(seg6_3001),XtNumber(seg7_3001), XtNumber(seg8_3001),
    XtNumber(seg9_3001),XtNumber(seg10_3001),XtNumber(seg11_3001),
};
```

continues

Listing C.4 Continued

```
static XPoint seg0_3002[] = {
    {-6,-12},{-6,9},
};
static XPoint seg1_3002[] = {
    {-5,-11},{-5,8},
};
static XPoint seg2_3002[] = {
    {-4,-12},{-4,9},
};
static XPoint seg3_3002[] = {
    {-9,-12},{3,-12},{6,-11},{7,-10},{8,-8},{8,-6},{7,-4},{6,-3},
    {3,-2},
};
static XPoint seg4_3002[] = {
    {6,-10},{7,-8},{7,-6},{6,-4},
};
static XPoint seg5_3002[] = {
    {3,-12},{5,-11},{6,-9},{6,-5},{5,-3},{3,-2},
};
static XPoint seg6_3002[] = {
    {-4,-2},{3,-2},{6,-1},{7,0},{8,2},{8,5},{7,7},{6,8},
    {3,9},{-9,9},
};
static XPoint seg7_3002[] = {
    {6,0},{7,2},{7,5},{6,7},
};
static XPoint seg8_3002[] = {
    {3,-2},{5,-1},{6,1},{6,6},{5,8},{3,9},
};
static XPoint seg9_3002[] = {
    {-8,-12},{-6,-11},
};
static XPoint seg10_3002[] = {
    {-7,-12},{-6,-10},
};
static XPoint seg11_3002[] = {
    {-3,-12},{-4,-10},
};
static XPoint seg12_3002[] = {
    {-2,-12},{-4,-11},
};
static XPoint seg13_3002[] = {
    {-6,8},{-8,9},
};
static XPoint seg14_3002[] = {
    {-6,7},{-7,9},
};
static XPoint seg15_3002[] = {
    {-4,7},{-3,9},
};
```

```
static XPoint seg16_3002[] = {
    {-4,8},{-2,9},
};
static XPoint *char3002[] = {
    seg0_3002,seg1_3002,seg2_3002,seg3_3002,seg4_3002,
    seg5_3002,seg6_3002,seg7_3002,seg8_3002,seg9_3002,
    seg10_3002,seg11_3002,seg12_3002,seg13_3002,seg14_3002,
    seg15_3002,seg16_3002,
    NULL,
};
static int char_p3002[] = {
    XtNumber(seg0_3002), XtNumber(seg1_3002), XtNumber(seg2_3002),
    XtNumber(seg3_3002), XtNumber(seg4_3002), XtNumber(seg5_3002),
    XtNumber(seg6_3002), XtNumber(seg7_3002), XtNumber(seg8_3002),
    XtNumber(seg9_3002), XtNumber(seg10_3002),XtNumber(seg11_3002),
    XtNumber(seg12_3002),XtNumber(seg13_3002),XtNumber(seg14_3002),
    XtNumber(seg15_3002),XtNumber(seg16_3002),
};
static XPoint seg0_3003[] = {
    {6,-9},{7,-12},{7,-6},{6,-9},{4,-11},{2,-12},{-1,-12},{-4,-11},
    {-6,-9},{-7,-7},{-8,-4},{-8,1},{-7,4},{-6,6},{-4,8},
    {-1,9},{2,9},{4,8},{6,6},{7,4},
};
static XPoint seg1_3003[] = {
    {-5,-9},{-6,-7},{-7,-4},{-7,1},{-6,4},{-5,6},
};
static XPoint seg2_3003[] = {
    {-1,-12},{-3,-11},{-5,-8},{-6,-4},{-6,1},{-5,5},{-3,8},{-1,9},
};
static XPoint *char3003[] = {
    seg0_3003,seg1_3003,seg2_3003,
    NULL,
};
static int char_p3003[] = {
    XtNumber(seg0_3003),XtNumber(seg1_3003),XtNumber(seg2_3003),
};
static XPoint seg0_3004[] = {
    {-6,-12},{-6,9},
};
static XPoint seg1_3004[] = {
    {-5,-11},{-5,8},
};
static XPoint seg2_3004[] = {
    {-4,-12},{-4,9},
};
static XPoint seg3_3004[] = {
    {-9,-12},{1,-12},{4,-11},{6,-9},{7,-7},{8,-4},{8,1},{7,4},
    {6,6},{4,8},{1,9},{-9,9},
};
static XPoint seg4_3004[] = {
    {5,-9},{6,-7},{7,-4},{7,1},{6,4},{5,6},
};
```

continues

Listing C.4 Continued

```
static XPoint seg5_3004[] = {
    {1,-12},{3,-11},{5,-8},{6,-4},{6,1},{5,5},{3,8},{1,9},
};
static XPoint seg6_3004[] = {
    {-8,-12},{-6,-11},
};
static XPoint seg7_3004[] = {
    {-7,-12},{-6,-10},
};
static XPoint seg8_3004[] = {
    {-3,-12},{-4,-10},
};
static XPoint seg9_3004[] = {
    {-2,-12},{-4,-11},
};
static XPoint seg10_3004[] = {
    {-6,8},{-8,9},
};
static XPoint seg11_3004[] = {
    {-6,7},{-7,9},
};
static XPoint seg12_3004[] = {
    {-4,7},{-3,9},
};
static XPoint seg13_3004[] = {
    {-4,8},{-2,9},
};
static XPoint *char3004[] = {
    seg0_3004,seg1_3004,seg2_3004,seg3_3004,seg4_3004,
    seg5_3004,seg6_3004,seg7_3004,seg8_3004,seg9_3004,
    seg10_3004,seg11_3004,seg12_3004,seg13_3004,
    NULL,
};
static int char_p3004[] = {
    XtNumber(seg0_3004), XtNumber(seg1_3004), XtNumber(seg2_3004),
    XtNumber(seg3_3004), XtNumber(seg4_3004), XtNumber(seg5_3004),
    XtNumber(seg6_3004), XtNumber(seg7_3004), XtNumber(seg8_3004),
    XtNumber(seg9_3004), XtNumber(seg10_3004),XtNumber(seg11_3004),
    XtNumber(seg12_3004),XtNumber(seg13_3004),
};
static XPoint seg0_3005[] = {
    {-6,-12},{-6,9},
};
static XPoint seg1_3005[] = {
    {-5,-11},{-5,8},
};
static XPoint seg2_3005[] = {
    {-4,-12},{-4,9},
};
static XPoint seg3_3005[] = {
    {-9,-12},{7,-12},{7,-6},
};
```

```
static XPoint seg4_3005[] = {
    {-4,-2},{2,-2},
};
static XPoint seg5_3005[] = {
    {2,-6},{2,2},
};
static XPoint seg6_3005[] = {
    {-9,9},{7,9},{7,3},
};
static XPoint seg7_3005[] = {
    {-8,-12},{-6,-11},
};
static XPoint seg8_3005[] = {
    {-7,-12},{-6,-10},
};
static XPoint seg9_3005[] = {
    {-3,-12},{-4,-10},
};
static XPoint seg10_3005[] = {
    {-2,-12},{-4,-11},
};
static XPoint seg11_3005[] = {
    {2,-12},{7,-11},
};
static XPoint seg12_3005[] = {
    {4,-12},{7,-10},
};
static XPoint seg13_3005[] = {
    {5,-12},{7,-9},
};
static XPoint seg14_3005[] = {
    {6,-12},{7,-6},
};
static XPoint seg15_3005[] = {
    {2,-6},{1,-2},{2,2},
};
static XPoint seg16_3005[] = {
    {2,-4},{0,-2},{2,0},
};
static XPoint seg17_3005[] = {
    {2,-3},{-2,-2},{2,-1},
};
static XPoint seg18_3005[] = {
    {-6,8},{-8,9},
};
static XPoint seg19_3005[] = {
    {-6,7},{-7,9},
};
static XPoint seg20_3005[] = {
    {-4,7},{-3,9},
};
```

continues

Listing C.4 Continued

```
static XPoint seg21_3005[] = {
    {-4,8},{-2,9},
};
static XPoint seg22_3005[] = {
    {2,9},{7,8},
};
static XPoint seg23_3005[] = {
    {4,9},{7,7},
};
static XPoint seg24_3005[] = {
    {5,9},{7,6},
};
static XPoint seg25_3005[] = {
    {6,9},{7,3},
};
static XPoint *char3005[] = {
    seg0_3005,seg1_3005,seg2_3005,seg3_3005,seg4_3005,
    seg5_3005,seg6_3005,seg7_3005,seg8_3005,seg9_3005,
    seg10_3005,seg11_3005,seg12_3005,seg13_3005,seg14_3005,
    seg15_3005,seg16_3005,seg17_3005,seg18_3005,seg19_3005,
    seg20_3005,seg21_3005,seg22_3005,seg23_3005,seg24_3005,
    seg25_3005,
    NULL,
};
static int char_p3005[] = {
    XtNumber(seg0_3005), XtNumber(seg1_3005), XtNumber(seg2_3005),
    XtNumber(seg3_3005), XtNumber(seg4_3005), XtNumber(seg5_3005),
    XtNumber(seg6_3005), XtNumber(seg7_3005), XtNumber(seg8_3005),
    XtNumber(seg9_3005), XtNumber(seg10_3005),XtNumber(seg11_3005),
    XtNumber(seg12_3005),XtNumber(seg13_3005),XtNumber(seg14_3005),
    XtNumber(seg15_3005),XtNumber(seg16_3005),XtNumber(seg17_3005),
    XtNumber(seg18_3005),XtNumber(seg19_3005),XtNumber(seg20_3005),
    XtNumber(seg21_3005),XtNumber(seg22_3005),XtNumber(seg23_3005),
    XtNumber(seg24_3005),XtNumber(seg25_3005),
};
static XPoint seg0_3006[] = {
    {-6,-12},{-6,9},
};
static XPoint seg1_3006[] = {
    {-5,-11},{-5,8},
};
static XPoint seg2_3006[] = {
    {-4,-12},{-4,9},
};
static XPoint seg3_3006[] = {
    {-9,-12},{7,-12},{7,-6},
};
static XPoint seg4_3006[] = {
    {-4,-2},{2,-2},
};
static XPoint seg5_3006[] = {
    {2,-6},{2,2},
};
```

```
static XPoint seg6_3006[] = {
    {-9,9},{-1,9},
};
static XPoint seg7_3006[] = {
    {-8,-12},{-6,-11},
};
static XPoint seg8_3006[] = {
    {-7,-12},{-6,-10},
};
static XPoint seg9_3006[] = {
    {-3,-12},{-4,-10},
};
static XPoint seg10_3006[] = {
    {-2,-12},{-4,-11},
};
static XPoint seg11_3006[] = {
    {2,-12},{7,-11},
};
static XPoint seg12_3006[] = {
    {4,-12},{7,-10},
};
static XPoint seg13_3006[] = {
    {5,-12},{7,-9},
};
static XPoint seg14_3006[] = {
    {6,-12},{7,-6},
};
static XPoint seg15_3006[] = {
    {2,-6},{1,-2},{2,2},
};
static XPoint seg16_3006[] = {
    {2,-4},{0,-2},{2,0},
};
static XPoint seg17_3006[] = {
    {2,-3},{-2,-2},{2,-1},
};
static XPoint seg18_3006[] = {
    {-6,8},{-8,9},
};
static XPoint seg19_3006[] = {
    {-6,7},{-7,9},
};
static XPoint seg20_3006[] = {
    {-4,7},{-3,9},
};
static XPoint seg21_3006[] = {
    {-4,8},{-2,9},
};
static XPoint *char3006[] = {
    seg0_3006,seg1_3006,seg2_3006,seg3_3006,seg4_3006,
    seg5_3006,seg6_3006,seg7_3006,seg8_3006,seg9_3006,
    seg10_3006,seg11_3006,seg12_3006,seg13_3006,seg14_3006,
```

continues

Listing C.4 Continued

```
    seg15_3006,seg16_3006,seg17_3006,seg18_3006,seg19_3006,
    seg20_3006,seg21_3006,
    NULL,
};
static int char_p3006[] = {
    XtNumber(seg0_3006), XtNumber(seg1_3006), XtNumber(seg2_3006),
    XtNumber(seg3_3006), XtNumber(seg4_3006), XtNumber(seg5_3006),
    XtNumber(seg6_3006), XtNumber(seg7_3006), XtNumber(seg8_3006),
    XtNumber(seg9_3006), XtNumber(seg10_3006),XtNumber(seg11_3006),
    XtNumber(seg12_3006),XtNumber(seg13_3006),XtNumber(seg14_3006),
    XtNumber(seg15_3006),XtNumber(seg16_3006),XtNumber(seg17_3006),
    XtNumber(seg18_3006),XtNumber(seg19_3006),XtNumber(seg20_3006),
    XtNumber(seg21_3006),
};
static XPoint seg0_3007[] = {
    {6,-9},{7,-12},{7,-6},{6,-9},{4,-11},{2,-12},{-1,-12},{-4,-11},
    {-6,-9},{-7,-7},{-8,-4},{-8,1},{-7,4},{-6,6},{-4,8},
    {-1,9},{2,9},{4,8},{6,8},{7,9},{7,1},
};
static XPoint seg1_3007[] = {
    {-5,-9},{-6,-7},{-7,-4},{-7,1},{-6,4},{-5,6},
};
static XPoint seg2_3007[] = {
    {-1,-12},{-3,-11},{-5,-8},{-6,-4},{-6,1},{-5,5},{-3,8},{-1,9},
};
static XPoint seg3_3007[] = {
    {6,2},{6,7},
};
static XPoint seg4_3007[] = {
    {5,1},{5,7},{4,8},
};
static XPoint seg5_3007[] = {
    {2,1},{10,1},
};
static XPoint seg6_3007[] = {
    {3,1},{5,2},
};
static XPoint seg7_3007[] = {
    {4,1},{5,3},
};
static XPoint seg8_3007[] = {
    {8,1},{7,3},
};
static XPoint seg9_3007[] = {
    {9,1},{7,2},
};
static XPoint *char3007[] = {
    seg0_3007,seg1_3007,seg2_3007,seg3_3007,seg4_3007,seg5_3007,
    seg6_3007,seg7_3007,seg8_3007,seg9_3007,
    NULL,
};
```

```
static int char_p3007[] = {
    XtNumber(seg0_3007),XtNumber(seg1_3007),XtNumber(seg2_3007),
    XtNumber(seg3_3007),XtNumber(seg4_3007),XtNumber(seg5_3007),
    XtNumber(seg6_3007),XtNumber(seg7_3007),XtNumber(seg8_3007),
    XtNumber(seg9_3007),
};
static XPoint seg0_3008[] = {
    {-7,-12},{-7,9},
};
static XPoint seg1_3008[] = {
    {-6,-11},{-6,8},
};
static XPoint seg2_3008[] = {
    {-5,-12},{-5,9},
};
static XPoint seg3_3008[] = {
    {5,-12},{5,9},
};
static XPoint seg4_3008[] = {
    {6,-11},{6,8},
};
static XPoint seg5_3008[] = {
    {7,-12},{7,9},
};
static XPoint seg6_3008[] = {
    {-10,-12},{-2,-12},
};
static XPoint seg7_3008[] = {
    {2,-12},{10,-12},
};
static XPoint seg8_3008[] = {
    {-5,-2},{5,-2},
};
static XPoint seg9_3008[] = {
    {-10,9},{-2,9},
};
static XPoint seg10_3008[] = {
    {2,9},{10,9},
};
static XPoint seg11_3008[] = {
    {-9,-12},{-7,-11},
};
static XPoint seg12_3008[] = {
    {-8,-12},{-7,-10},
};
static XPoint seg13_3008[] = {
    {-4,-12},{-5,-10},
};
static XPoint seg14_3008[] = {
    {-3,-12},{-5,-11},
};
```

continues

Listing C.4 Continued

```
static XPoint seg15_3008[] = {
    {3,-12},{5,-11},
};
static XPoint seg16_3008[] = {
    {4,-12},{5,-10},
};
static XPoint seg17_3008[] = {
    {8,-12},{7,-10},
};
static XPoint seg18_3008[] = {
    {9,-12},{7,-11},
};
static XPoint seg19_3008[] = {
    {-7,8},{-9,9},
};
static XPoint seg20_3008[] = {
    {-7,7},{-8,9},
};
static XPoint seg21_3008[] = {
    {-5,7},{-4,9},
};
static XPoint seg22_3008[] = {
    {-5,8},{-3,9},
};
static XPoint seg23_3008[] = {
    {5,8},{3,9},
};
static XPoint seg24_3008[] = {
    {5,7},{4,9},
};
static XPoint seg25_3008[] = {
    {7,7},{8,9},
};
static XPoint seg26_3008[] = {
    {7,8},{9,9},
};
static XPoint *char3008[] = {
    seg0_3008,seg1_3008,seg2_3008,seg3_3008,seg4_3008,
    seg5_3008,seg6_3008,seg7_3008,seg8_3008,seg9_3008,
    seg10_3008,seg11_3008,seg12_3008,seg13_3008,seg14_3008,
    seg15_3008,seg16_3008,seg17_3008,seg18_3008,seg19_3008,
    seg20_3008,seg21_3008,seg22_3008,seg23_3008,seg24_3008,
    seg25_3008,seg26_3008,
    NULL,
};
static int char_p3008[] = {
    XtNumber(seg0_3008), XtNumber(seg1_3008), XtNumber(seg2_3008),
    XtNumber(seg3_3008), XtNumber(seg4_3008), XtNumber(seg5_3008),
    XtNumber(seg6_3008), XtNumber(seg7_3008), XtNumber(seg8_3008),
    XtNumber(seg9_3008), XtNumber(seg10_3008),XtNumber(seg11_3008),
    XtNumber(seg12_3008),XtNumber(seg13_3008),XtNumber(seg14_3008),
    XtNumber(seg15_3008),XtNumber(seg16_3008),XtNumber(seg17_3008),
    XtNumber(seg18_3008),XtNumber(seg19_3008),XtNumber(seg20_3008),
    XtNumber(seg21_3008),XtNumber(seg22_3008),XtNumber(seg23_3008),
    XtNumber(seg24_3008),XtNumber(seg25_3008),XtNumber(seg26_3008),
};
static XPoint seg0_3009[] = {
    {-1,-12},{-1,9},
};
static XPoint seg1_3009[] = {
    {0,-11},{0,8},
};
static XPoint seg2_3009[] = {
    {1,-12},{1,9},
};
static XPoint seg3_3009[] = {
    {-4,-12},{4,-12},
};
static XPoint seg4_3009[] = {
    {-4,9},{4,9},
};
static XPoint seg5_3009[] = {
    {-3,-12},{-1,-11},
};
static XPoint seg6_3009[] = {
    {-2,-12},{-1,-10},
};
static XPoint seg7_3009[] = {
    {2,-12},{1,-10},
};
static XPoint seg8_3009[] = {
    {3,-12},{1,-11},
};
static XPoint seg9_3009[] = {
    {-1,8},{-3,9},
};
static XPoint seg10_3009[] = {
    {-1,7},{-2,9},
};
static XPoint seg11_3009[] = {
    {1,7},{2,9},
};
static XPoint seg12_3009[] = {
    {1,8},{3,9},
};
static XPoint *char3009[] = {
    seg0_3009,seg1_3009,seg2_3009,seg3_3009,seg4_3009,
    seg5_3009,seg6_3009,seg7_3009,seg8_3009,seg9_3009,
    seg10_3009,seg11_3009,seg12_3009,
    NULL,
};
static int char_p3009[] = {
    XtNumber(seg0_3009),XtNumber(seg1_3009), XtNumber(seg2_3009),
    XtNumber(seg3_3009),XtNumber(seg4_3009), XtNumber(seg5_3009),
```

continues

Listing C.4 Continued

```
    XtNumber(seg6_3009),XtNumber(seg7_3009), XtNumber(seg8_3009),
    XtNumber(seg9_3009),XtNumber(seg10_3009),XtNumber(seg11_3009),
    XtNumber(seg12_3009),
};
static XPoint seg0_3010[] = {
    {1,-12},{1,5},{0,8},{-1,9},
};
static XPoint seg1_3010[] = {
    {2,-11},{2,5},{1,8},
};
static XPoint seg2_3010[] = {
    {3,-12},{3,5},{2,8},{-1,9},{-3,9},{-5,8},{-6,6},{-6,4},
    {-5,3},{-4,3},{-3,4},{-3,5},{-4,6},{-5,6},
};
static XPoint seg3_3010[] = {
    {-5,4},{-5,5},{-4,5},{-4,4},{-5,4},
};
static XPoint seg4_3010[] = {
    {-2,-12},{6,-12},
};
static XPoint seg5_3010[] = {
    {-1,-12},{1,-11},
};
static XPoint seg6_3010[] = {
    {0,-12},{1,-10},
};
static XPoint seg7_3010[] = {
    {4,-12},{3,-10},
};
static XPoint seg8_3010[] = {
    {5,-12},{3,-11},
};
static XPoint *char3010[] = {
    seg0_3010,seg1_3010,seg2_3010,seg3_3010,seg4_3010,seg5_3010,
    seg6_3010,seg7_3010,seg8_3010,
    NULL,
};
static int char_p3010[] = {
    XtNumber(seg0_3010),XtNumber(seg1_3010),XtNumber(seg2_3010),
    XtNumber(seg3_3010),XtNumber(seg4_3010),XtNumber(seg5_3010),
    XtNumber(seg6_3010),XtNumber(seg7_3010),XtNumber(seg8_3010),
};
static XPoint seg0_3011[] = {
    {-7,-12},{-7,9},
};
static XPoint seg1_3011[] = {
    {-6,-11},{-6,8},
};
static XPoint seg2_3011[] = {
    {-5,-12},{-5,9},
};
```

```
static XPoint seg3_3011[] = {
    {6,-11},{-5,0},
};
static XPoint seg4_3011[] = {
    {-2,-2},{5,9},
};
static XPoint seg5_3011[] = {
    {-1,-2},{6,9},
};
static XPoint seg6_3011[] = {
    {-1,-4},{7,9},
};
static XPoint seg7_3011[] = {
    {-10,-12},{-2,-12},
};
static XPoint seg8_3011[] = {
    {3,-12},{9,-12},
};
static XPoint seg9_3011[] = {
    {-10,9},{-2,9},
};
static XPoint seg10_3011[] = {
    {2,9},{9,9},
};
static XPoint seg11_3011[] = {
    {-9,-12},{-7,-11},
};
static XPoint seg12_3011[] = {
    {-8,-12},{-7,-10},
};
static XPoint seg13_3011[] = {
    {-4,-12},{-5,-10},
};
static XPoint seg14_3011[] = {
    {-3,-12},{-5,-11},
};
static XPoint seg15_3011[] = {
    {5,-12},{6,-11},
};
static XPoint seg16_3011[] = {
    {8,-12},{6,-11},
};
static XPoint seg17_3011[] = {
    {-7,8},{-9,9},
};
static XPoint seg18_3011[] = {
    {-7,7},{-8,9},
};
static XPoint seg19_3011[] = {
    {-5,7},{-4,9},
};
```

continues

Listing C.4 Continued

```
static XPoint seg20_3011[] = {
    {-5,8},{-3,9},
};
static XPoint seg21_3011[] = {
    {5,7},{3,9},
};
static XPoint seg22_3011[] = {
    {5,7},{8,9},
};
static XPoint *char3011[] = {
    seg0_3011,seg1_3011,seg2_3011,seg3_3011,seg4_3011,
    seg5_3011,seg6_3011,seg7_3011,seg8_3011,seg9_3011,
    seg10_3011,seg11_3011,seg12_3011,seg13_3011,seg14_3011,
    seg15_3011,seg16_3011,seg17_3011,seg18_3011,seg19_3011,
    seg20_3011,seg21_3011,seg22_3011,
    NULL,
};
static int char_p3011[] = {
    XtNumber(seg0_3011), XtNumber(seg1_3011), XtNumber(seg2_3011),
    XtNumber(seg3_3011), XtNumber(seg4_3011), XtNumber(seg5_3011),
    XtNumber(seg6_3011), XtNumber(seg7_3011), XtNumber(seg8_3011),
    XtNumber(seg9_3011), XtNumber(seg10_3011),XtNumber(seg11_3011),
    XtNumber(seg12_3011),XtNumber(seg13_3011),XtNumber(seg14_3011),
    XtNumber(seg15_3011),XtNumber(seg16_3011),XtNumber(seg17_3011),
    XtNumber(seg18_3011),XtNumber(seg19_3011),XtNumber(seg20_3011),
    XtNumber(seg21_3011),XtNumber(seg22_3011),
};
static XPoint seg0_3012[] = {
    {-4,-12},{-4,9},
};
static XPoint seg1_3012[] = {
    {-3,-11},{-3,8},
};
static XPoint seg2_3012[] = {
    {-2,-12},{-2,9},
};
static XPoint seg3_3012[] = {
    {-7,-12},{1,-12},
};
static XPoint seg4_3012[] = {
    {-7,9},{8,9},{8,3},
};
static XPoint seg5_3012[] = {
    {-6,-12},{-4,-11},
};
static XPoint seg6_3012[] = {
    {-5,-12},{-4,-10},
};
static XPoint seg7_3012[] = {
    {-1,-12},{-2,-10},
};
```

```
static XPoint seg8_3012[] = {
    {0,-12},{-2,-11},
};
static XPoint seg9_3012[] = {
    {-4,8},{-6,9},
};
static XPoint seg10_3012[] = {
    {-4,7},{-5,9},
};
static XPoint seg11_3012[] = {
    {-2,7},{-1,9},
};
static XPoint seg12_3012[] = {
    {-2,8},{0,9},
};
static XPoint seg13_3012[] = {
    {3,9},{8,8},
};
static XPoint seg14_3012[] = {
    {5,9},{8,7},
};
static XPoint seg15_3012[] = {
    {6,9},{8,6},
};
static XPoint seg16_3012[] = {
    {7,9},{8,3},
};
static XPoint *char3012[] = {
    seg0_3012,seg1_3012,seg2_3012,seg3_3012,seg4_3012,
    seg5_3012,seg6_3012,seg7_3012,seg8_3012,seg9_3012,
    seg10_3012,seg11_3012,seg12_3012,seg13_3012,seg14_3012,
    seg15_3012,seg16_3012,
    NULL,
};
static int char_p3012[] = {
    XtNumber(seg0_3012), XtNumber(seg1_3012), XtNumber(seg2_3012),
    XtNumber(seg3_3012), XtNumber(seg4_3012), XtNumber(seg5_3012),
    XtNumber(seg6_3012), XtNumber(seg7_3012), XtNumber(seg8_3012),
    XtNumber(seg9_3012), XtNumber(seg10_3012),XtNumber(seg11_3012),
    XtNumber(seg12_3012),XtNumber(seg13_3012),XtNumber(seg14_3012),
    XtNumber(seg15_3012),XtNumber(seg16_3012),
};
static XPoint seg0_3013[] = {
    {-8,-12},{-8,8},
};
static XPoint seg1_3013[] = {
    {-8,-12},{-1,9},
};
static XPoint seg2_3013[] = {
    {-7,-12},{-1,6},
};
static XPoint seg3_3013[] = {
    {-6,-12},{0,6},
};
```

continues

Listing C.4 Continued

```
static XPoint seg4_3013[] = {
    {6,-12},{-1,9},
};
static XPoint seg5_3013[] = {
    {6,-12},{6,9},
};
static XPoint seg6_3013[] = {
    {7,-11},{7,8},
};
static XPoint seg7_3013[] = {
    {8,-12},{8,9},
};
static XPoint seg8_3013[] = {
    {-11,-12},{-6,-12},
};
static XPoint seg9_3013[] = {
    {6,-12},{11,-12},
};
static XPoint seg10_3013[] = {
    {-11,9},{-5,9},
};
static XPoint seg11_3013[] = {
    {3,9},{11,9},
};
static XPoint seg12_3013[] = {
    {-10,-12},{-8,-11},
};
static XPoint seg13_3013[] = {
    {9,-12},{8,-10},
};
static XPoint seg14_3013[] = {
    {10,-12},{8,-11},
};
static XPoint seg15_3013[] = {
    {-8,8},{-10,9},
};
static XPoint seg16_3013[] = {
    {-8,8},{-6,9},
};
static XPoint seg17_3013[] = {
    {6,8},{4,9},
};
static XPoint seg18_3013[] = {
    {6,7},{5,9},
};
static XPoint seg19_3013[] = {
    {8,7},{9,9},
};
```

```
static XPoint seg20_3013[] = {
    {8,8},{10,9},
};
static XPoint *char3013[] = {
    seg0_3013,seg1_3013,seg2_3013,seg3_3013,seg4_3013,
    seg5_3013,seg6_3013,seg7_3013,seg8_3013,seg9_3013,
    seg10_3013,seg11_3013,seg12_3013,seg13_3013,seg14_3013,
    seg15_3013,seg16_3013,seg17_3013,seg18_3013,seg19_3013,
    seg20_3013,
    NULL,
};
static int char_p3013[] = {
    XtNumber(seg0_3013), XtNumber(seg1_3013), XtNumber(seg2_3013),
    XtNumber(seg3_3013), XtNumber(seg4_3013), XtNumber(seg5_3013),
    XtNumber(seg6_3013), XtNumber(seg7_3013), XtNumber(seg8_3013),
    XtNumber(seg9_3013), XtNumber(seg10_3013),XtNumber(seg11_3013),
    XtNumber(seg12_3013),XtNumber(seg13_3013),XtNumber(seg14_3013),
    XtNumber(seg15_3013),XtNumber(seg16_3013),XtNumber(seg17_3013),
    XtNumber(seg18_3013),XtNumber(seg19_3013),XtNumber(seg20_3013),
};
static XPoint seg0_3014[] = {
    {-7,-12},{-7,8},
};
static XPoint seg1_3014[] = {
    {-7,-12},{7,9},
};
static XPoint seg2_3014[] = {
    {-6,-12},{6,6},
};
static XPoint seg3_3014[] = {
    {-5,-12},{7,6},
};
static XPoint seg4_3014[] = {
    {7,-11},{7,9},
};
static XPoint seg5_3014[] = {
    {-10,-12},{-5,-12},
};
static XPoint seg6_3014[] = {
    {4,-12},{10,-12},
};
static XPoint seg7_3014[] = {
    {-10,9},{-4,9},
};
static XPoint seg8_3014[] = {
    {-9,-12},{-7,-11},
};
static XPoint seg9_3014[] = {
    {5,-12},{7,-11},
};
static XPoint seg10_3014[] = {
    {9,-12},{7,-11},
};
```

continues

Listing C.4 Continued

```
static XPoint seg11_3014[] = {
    {-7,8},{-9,9},
};
static XPoint seg12_3014[] = {
    {-7,8},{-5,9},
};
static XPoint *char3014[] = {
    seg0_3014,seg1_3014,seg2_3014,seg3_3014,seg4_3014,
    seg5_3014,seg6_3014,seg7_3014,seg8_3014,seg9_3014,
    seg10_3014,seg11_3014,seg12_3014,
    NULL,
};
static int char_p3014[] = {
    XtNumber(seg0_3014),XtNumber(seg1_3014), XtNumber(seg2_3014),
    XtNumber(seg3_3014),XtNumber(seg4_3014), XtNumber(seg5_3014),
    XtNumber(seg6_3014),XtNumber(seg7_3014), XtNumber(seg8_3014),
    XtNumber(seg9_3014),XtNumber(seg10_3014),XtNumber(seg11_3014),
    XtNumber(seg12_3014),
};
static XPoint seg0_3015[] = {
    {-1,-12},{-4,-11},{-6,-9},{-7,-7},{-8,-3},{-8,0},
    {-7,4},{-6,6},{-4,8},{-1,9},{1,9},{4,8},{6,6},{7,4},{8,0},
    {8,-3},{7,-7},{6,-9},{4,-11},{1,-12},{-1,-12},
};
static XPoint seg1_3015[] = {
    {-5,-9},{-6,-7},{-7,-4},{-7,1},{-6,4},{-5,6},
};
static XPoint seg2_3015[] = {
    {5,6},{6,4},{7,1},{7,-4},{6,-7},{5,-9},
};
static XPoint seg3_3015[] = {
    {-1,-12},{-3,-11},{-5,-8},{-6,-4},{-6,1},{-5,5},{-3,8},{-1,9},
};
static XPoint seg4_3015[] = {
    {1,9},{3,8},{5,5},{6,1},{6,-4},{5,-8},{3,-11},{1,-12},
};
static XPoint *char3015[] = {
    seg0_3015,seg1_3015,seg2_3015,seg3_3015,seg4_3015,
    NULL,
};
static int char_p3015[] = {
    XtNumber(seg0_3015),XtNumber(seg1_3015),XtNumber(seg2_3015),
    XtNumber(seg3_3015),XtNumber(seg4_3015),
};
static XPoint seg0_3016[] = {
    {-6,-12},{-6,9},
};
static XPoint seg1_3016[] = {
    {-5,-11},{-5,8},
};
```

```
static XPoint seg2_3016[] = {
    {-4,-12},{-4,9},
};
static XPoint seg3_3016[] = {
    {-9,-12},{3,-12},{6,-11},{7,-10},{8,-8},{8,-5},{7,-3},{6,-2},
    {3,-1},{-4,-1},
};
static XPoint seg4_3016[] = {
    {6,-10},{7,-8},{7,-5},{6,-3},
};
static XPoint seg5_3016[] = {
    {3,-12},{5,-11},{6,-9},{6,-4},{5,-2},{3,-1},
};
static XPoint seg6_3016[] = {
    {-9,9},{-1,9},
};
static XPoint seg7_3016[] = {
    {-8,-12},{-6,-11},
};
static XPoint seg8_3016[] = {
    {-7,-12},{-6,-10},
};
static XPoint seg9_3016[] = {
    {-3,-12},{-4,-10},
};
static XPoint seg10_3016[] = {
    {-2,-12},{-4,-11},
};
static XPoint seg11_3016[] = {
    {-6,8},{-8,9},
};
static XPoint seg12_3016[] = {
    {-6,7},{-7,9},
};
static XPoint seg13_3016[] = {
    {-4,7},{-3,9},
};
static XPoint seg14_3016[] = {
    {-4,8},{-2,9},
};
static XPoint *char3016[] = {
    seg0_3016,seg1_3016,seg2_3016,seg3_3016,seg4_3016,
    seg5_3016,seg6_3016,seg7_3016,seg8_3016,seg9_3016,
    seg10_3016,seg11_3016,seg12_3016,seg13_3016,seg14_3016,
    NULL,
};
static int char_p3016[] = {
    XtNumber(seg0_3016), XtNumber(seg1_3016), XtNumber(seg2_3016),
    XtNumber(seg3_3016), XtNumber(seg4_3016), XtNumber(seg5_3016),
    XtNumber(seg6_3016), XtNumber(seg7_3016), XtNumber(seg8_3016),
    XtNumber(seg9_3016), XtNumber(seg10_3016),XtNumber(seg11_3016),
    XtNumber(seg12_3016),XtNumber(seg13_3016),XtNumber(seg14_3016),
};
```

continues

Listing C.4 Continued

```
static XPoint seg0_3017[] = {
    {-1,-12},{-4,-11},{-6,-9},{-7,-7},{-8,-3},{-8,0},
    {-7,4},{-6,6},{-4,8},{-1,9},{1,9},{4,8},{6,6},{7,4},{8,0},
    {8,-3},{7,-7},{6,-9},{4,-11},{1,-12},{-1,-12},
};
static XPoint seg1_3017[] = {
    {-5,-9},{-6,-7},{-7,-4},{-7,1},{-6,4},{-5,6},
};
static XPoint seg2_3017[] = {
    {5,6},{6,4},{7,1},{7,-4},{6,-7},{5,-9},
};
static XPoint seg3_3017[] = {
    {-1,-12},{-3,-11},{-5,-8},{-6,-4},{-6,1},{-5,5},{-3,8},{-1,9},
};
static XPoint seg4_3017[] = {
    {1,9},{3,8},{5,5},{6,1},{6,-4},{5,-8},{3,-11},{1,-12},
};
static XPoint seg5_3017[] = {
    {-4,6},{-3,4},{-1,3},{0,3},{2,4},{3,6},{4,12},{5,14},
    {7,14},{8,12},{8,10},
};
static XPoint seg6_3017[] = {
    {4,10},{5,12},{6,13},{7,13},
};
static XPoint seg7_3017[] = {
    {3,6},{5,11},{6,12},{7,12},{8,11},
};
static XPoint *char3017[] = {
    seg0_3017,seg1_3017,seg2_3017,seg3_3017,seg4_3017,seg5_3017,
    seg6_3017,seg7_3017,
    NULL,
};
static int char_p3017[] = {
    XtNumber(seg0_3017),XtNumber(seg1_3017),XtNumber(seg2_3017),
    XtNumber(seg3_3017),XtNumber(seg4_3017),XtNumber(seg5_3017),
    XtNumber(seg6_3017),XtNumber(seg7_3017),
};
static XPoint seg0_3018[] = {
    {-6,-12},{-6,9},
};
static XPoint seg1_3018[] = {
    {-5,-11},{-5,8},
};
static XPoint seg2_3018[] = {
    {-4,-12},{-4,9},
};
static XPoint seg3_3018[] = {
    {-9,-12},{3,-12},{6,-11},{7,-10},{8,-8},{8,-6},{7,-4},{6,-3},
    {3,-2},{-4,-2},
};
```

```
static XPoint seg4_3018[] = {
    {6,-10},{7,-8},{7,-6},{6,-4},
};
static XPoint seg5_3018[] = {
    {3,-12},{5,-11},{6,-9},{6,-5},{5,-3},{3,-2},
};
static XPoint seg6_3018[] = {
    {0,-2},{2,-1},{3,1},{5,7},{6,9},{8,9},{9,7},{9,5},
};
static XPoint seg7_3018[] = {
    {5,5},{6,7},{7,8},{8,8},
};
static XPoint seg8_3018[] = {
    {2,-1},{3,0},{6,6},{7,7},{8,7},{9,6},
};
static XPoint seg9_3018[] = {
    {-9,9},{-1,9},
};
static XPoint seg10_3018[] = {
    {-8,-12},{-6,-11},
};
static XPoint seg11_3018[] = {
    {-7,-12},{-6,-10},
};
static XPoint seg12_3018[] = {
    {-3,-12},{-4,-10},
};
static XPoint seg13_3018[] = {
    {-2,-12},{-4,-11},
};
static XPoint seg14_3018[] = {
    {-6,8},{-8,9},
};
static XPoint seg15_3018[] = {
    {-6,7},{-7,9},
};
static XPoint seg16_3018[] = {
    {-4,7},{-3,9},
};
static XPoint seg17_3018[] = {
    {-4,8},{-2,9},
};
static XPoint *char3018[] = {
    seg0_3018,seg1_3018,seg2_3018,seg3_3018,seg4_3018,
    seg5_3018,seg6_3018,seg7_3018,seg8_3018,seg9_3018,
    seg10_3018,seg11_3018,seg12_3018,seg13_3018,
    seg14_3018,seg15_3018,seg16_3018,seg17_3018,
    NULL,
};
static int char_p3018[] = {
    XtNumber(seg0_3018), XtNumber(seg1_3018), XtNumber(seg2_3018),
    XtNumber(seg3_3018), XtNumber(seg4_3018), XtNumber(seg5_3018),
    XtNumber(seg6_3018), XtNumber(seg7_3018), XtNumber(seg8_3018),
    XtNumber(seg9_3018), XtNumber(seg10_3018),XtNumber(seg11_3018),
```

continues

Listing C.4 Continued

```
    XtNumber(seg12_3018),XtNumber(seg13_3018),XtNumber(seg14_3018),
    XtNumber(seg15_3018),XtNumber(seg16_3018),XtNumber(seg17_3018),
};
static XPoint seg0_3019[] = {
    {6,-9},{7,-12},{7,-6},{6,-9},{4,-11},{1,-12},
    {-2,-12},{-5,-11},{-7,-9},{-7,-6},{-6,-4},{-3,-2},{3,0},{5,1},
    {6,3},{6,6},{5,8},
};
static XPoint seg1_3019[] = {
    {-6,-6},{-5,-4},{-3,-3},{3,-1},{5,0},{6,2},
};
static XPoint seg2_3019[] = {
    {-5,-11},{-6,-9},{-6,-7},{-5,-5},{-3,-4},{3,-2},{6,0},{7,2},
    {7,5},{6,7},{5,8},{2,9},{-1,9},{-4,8},{-6,6},{-7,3},{-7,9},
    {-6,6},
};
static XPoint *char3019[] = {
    seg0_3019,seg1_3019,seg2_3019,
    NULL,
};
static int char_p3019[] = {
    XtNumber(seg0_3019),XtNumber(seg1_3019),XtNumber(seg2_3019),
};
static XPoint seg0_3020[] = {
    {-8,-12},{-8,-6},
};
static XPoint seg1_3020[] = {
    {-1,-12},{-1,9},
};
static XPoint seg2_3020[] = {
    {0,-11},{0,8},
};
static XPoint seg3_3020[] = {
    {1,-12},{1,9},
};
static XPoint seg4_3020[] = {
    {8,-12},{8,-6},
};
static XPoint seg5_3020[] = {
    {-8,-12},{8,-12},
};
static XPoint seg6_3020[] = {
    {-4,9},{4,9},
};
static XPoint seg7_3020[] = {
    {-7,-12},{-8,-6},
};
static XPoint seg8_3020[] = {
    {-6,-12},{-8,-9},
};
```

```
static XPoint seg9_3020[] = {
    {-5,-12},{-8,-10},
};
static XPoint seg10_3020[] = {
    {-3,-12},{-8,-11},
};
static XPoint seg11_3020[] = {
    {3,-12},{8,-11},
};
static XPoint seg12_3020[] = {
    {5,-12},{8,-10},
};
static XPoint seg13_3020[] = {
    {6,-12},{8,-9},
};
static XPoint seg14_3020[] = {
    {7,-12},{8,-6},
};
static XPoint seg15_3020[] = {
    {-1,8},{-3,9},
};
static XPoint seg16_3020[] = {
    {-1,7},{-2,9},
};
static XPoint seg17_3020[] = {
    {1,7},{2,9},
};
static XPoint seg18_3020[] = {
    {1,8},{3,9},
};
static XPoint *char3020[] = {
    seg0_3020,seg1_3020,seg2_3020,seg3_3020,seg4_3020,
    seg5_3020,seg6_3020,seg7_3020,seg8_3020,seg9_3020,
    seg10_3020,seg11_3020,seg12_3020,seg13_3020,seg14_3020,
    seg15_3020,seg16_3020,seg17_3020,seg18_3020,
    NULL,
};
static int char_p3020[] = {
    XtNumber(seg0_3020), XtNumber(seg1_3020), XtNumber(seg2_3020),
    XtNumber(seg3_3020), XtNumber(seg4_3020), XtNumber(seg5_3020),
    XtNumber(seg6_3020), XtNumber(seg7_3020), XtNumber(seg8_3020),
    XtNumber(seg9_3020), XtNumber(seg10_3020),XtNumber(seg11_3020),
    XtNumber(seg12_3020),XtNumber(seg13_3020),XtNumber(seg14_3020),
    XtNumber(seg15_3020),XtNumber(seg16_3020),XtNumber(seg17_3020),
    XtNumber(seg18_3020),
};
static XPoint seg0_3021[] = {
    {-7,-12},{-7,3},{-6,6},{-4,8},{-1,9},{1,9},
    {4,8},{6,6},{7,3},{7,-11},
};
static XPoint seg1_3021[] = {
    {-6,-11},{-6,4},{-5,6},
};
```

continues

Listing C.4 Continued

```
static XPoint seg2_3021[] = {
    {-5,-12},{-5,4},{-4,7},{-3,8},{-1,9},
};
static XPoint seg3_3021[] = {
    {-10,-12},{-2,-12},
};
static XPoint seg4_3021[] = {
    {4,-12},{10,-12},
};
static XPoint seg5_3021[] = {
    {-9,-12},{-7,-11},
};
static XPoint seg6_3021[] = {
    {-8,-12},{-7,-10},
};
static XPoint seg7_3021[] = {
    {-4,-12},{-5,-10},
};
static XPoint seg8_3021[] = {
    {-3,-12},{-5,-11},
};
static XPoint seg9_3021[] = {
    {5,-12},{7,-11},
};
static XPoint seg10_3021[] = {
    {9,-12},{7,-11},
};
static XPoint *char3021[] = {
    seg0_3021,seg1_3021,seg2_3021,seg3_3021,seg4_3021,seg5_3021,
    seg6_3021,seg7_3021,seg8_3021,seg9_3021,seg10_3021,
    NULL,
};
static int char_p3021[] = {
    XtNumber(seg0_3021),XtNumber(seg1_3021),XtNumber(seg2_3021),
    XtNumber(seg3_3021),XtNumber(seg4_3021),XtNumber(seg5_3021),
    XtNumber(seg6_3021),XtNumber(seg7_3021),XtNumber(seg8_3021),
    XtNumber(seg9_3021),XtNumber(seg10_3021),
};
static XPoint seg0_3022[] = {
    {-7,-12},{0,9},
};
static XPoint seg1_3022[] = {
    {-6,-12},{0,6},{0,9},
};
static XPoint seg2_3022[] = {
    {-5,-12},{1,6},
};
static XPoint seg3_3022[] = {
    {7,-11},{0,9},
};
```

```
static XPoint seg4_3022[] = {
    {-9,-12},{-2,-12},
};
static XPoint seg5_3022[] = {
    {3,-12},{9,-12},
};
static XPoint seg6_3022[] = {
    {-8,-12},{-6,-10},
};
static XPoint seg7_3022[] = {
    {-4,-12},{-5,-10},
};
static XPoint seg8_3022[] = {
    {-3,-12},{-5,-11},
};
static XPoint seg9_3022[] = {
    {5,-12},{7,-11},
};
static XPoint seg10_3022[] = {
    {8,-12},{7,-11},
};
static XPoint *char3022[] = {
    seg0_3022,seg1_3022,seg2_3022,seg3_3022,seg4_3022,seg5_3022,
    seg6_3022,seg7_3022,seg8_3022,seg9_3022,seg10_3022,
    NULL,
};
static int char_p3022[] = {
    XtNumber(seg0_3022),XtNumber(seg1_3022),XtNumber(seg2_3022),
    XtNumber(seg3_3022),XtNumber(seg4_3022),XtNumber(seg5_3022),
    XtNumber(seg6_3022),XtNumber(seg7_3022),XtNumber(seg8_3022),
    XtNumber(seg9_3022),XtNumber(seg10_3022),
};
static XPoint seg0_3023[] = {
    {-8,-12},{-4,9},
};
static XPoint seg1_3023[] = {
    {-7,-12},{-4,4},{-4,9},
};
static XPoint seg2_3023[] = {
    {-6,-12},{-3,4},
};
static XPoint seg3_3023[] = {
    {0,-12},{-3,4},{-4,9},
};
static XPoint seg4_3023[] = {
    {0,-12},{4,9},
};
static XPoint seg5_3023[] = {
    {1,-12},{4,4},{4,9},
};
static XPoint seg6_3023[] = {
    {2,-12},{5,4},
};
```

continues

Listing C.4 Continued

```
static XPoint seg7_3023[] = {
    {8,-11},{5,4},{4,9},
};
static XPoint seg8_3023[] = {
    {-11,-12},{-3,-12},
};
static XPoint seg9_3023[] = {
    {0,-12},{2,-12},
};
static XPoint seg10_3023[] = {
    {5,-12},{11,-12},
};
static XPoint seg11_3023[] = {
    {-10,-12},{-7,-11},
};
static XPoint seg12_3023[] = {
    {-9,-12},{-7,-10},
};
static XPoint seg13_3023[] = {
    {-5,-12},{-6,-10},
};
static XPoint seg14_3023[] = {
    {-4,-12},{-6,-11},
};
static XPoint seg15_3023[] = {
    {6,-12},{8,-11},
};
static XPoint seg16_3023[] = {
    {10,-12},{8,-11},
};
static XPoint *char3023[] = {
    seg0_3023,seg1_3023,seg2_3023,seg3_3023,seg4_3023,
    seg5_3023,seg6_3023,seg7_3023,seg8_3023,seg9_3023,
    seg10_3023,seg11_3023,seg12_3023,seg13_3023,seg14_3023,
    seg15_3023,seg16_3023,
    NULL,
};
static int char_p3023[] = {
    XtNumber(seg0_3023), XtNumber(seg1_3023), XtNumber(seg2_3023),
    XtNumber(seg3_3023), XtNumber(seg4_3023), XtNumber(seg5_3023),
    XtNumber(seg6_3023), XtNumber(seg7_3023), XtNumber(seg8_3023),
    XtNumber(seg9_3023), XtNumber(seg10_3023),XtNumber(seg11_3023),
    XtNumber(seg12_3023),XtNumber(seg13_3023),XtNumber(seg14_3023),
    XtNumber(seg15_3023),XtNumber(seg16_3023),
};
static XPoint seg0_3024[] = {
    {-7,-12},{5,9},
};
static XPoint seg1_3024[] = {
    {-6,-12},{6,9},
};
```

```
static XPoint seg2_3024[] = {
    {-5,-12},{7,9},
};
static XPoint seg3_3024[] = {
    {6,-11},{-6,8},
};
static XPoint seg4_3024[] = {
    {-9,-12},{-2,-12},
};
static XPoint seg5_3024[] = {
    {3,-12},{9,-12},
};
static XPoint seg6_3024[] = {
    {-9,9},{-3,9},
};
static XPoint seg7_3024[] = {
    {2,9},{9,9},
};
static XPoint seg8_3024[] = {
    {-8,-12},{-5,-10},
};
static XPoint seg9_3024[] = {
    {-4,-12},{-5,-10},
};
static XPoint seg10_3024[] = {
    {-3,-12},{-5,-11},
};
static XPoint seg11_3024[] = {
    {4,-12},{6,-11},
};
static XPoint seg12_3024[] = {
    {8,-12},{6,-11},
};
static XPoint seg13_3024[] = {
    {-6,8},{-8,9},
};
static XPoint seg14_3024[] = {
    {-6,8},{-4,9},
};
static XPoint seg15_3024[] = {
    {5,8},{3,9},
};
static XPoint seg16_3024[] = {
    {5,7},{4,9},
};
static XPoint seg17_3024[] = {
    {5,7},{8,9},
};
static XPoint *char3024[] = {
    seg0_3024,seg1_3024,seg2_3024,seg3_3024,seg4_3024,
    seg5_3024,seg6_3024,seg7_3024,seg8_3024,seg9_3024,
    seg10_3024,seg11_3024,seg12_3024,seg13_3024,seg14_3024,
```

continues

Listing C.4 Continued

```
    seg15_3024,seg16_3024,seg17_3024,
    NULL,
};
static int char_p3024[] = {
    XtNumber(seg0_3024), XtNumber(seg1_3024), XtNumber(seg2_3024),
    XtNumber(seg3_3024), XtNumber(seg4_3024), XtNumber(seg5_3024),
    XtNumber(seg6_3024), XtNumber(seg7_3024), XtNumber(seg8_3024),
    XtNumber(seg9_3024), XtNumber(seg10_3024),XtNumber(seg11_3024),
    XtNumber(seg12_3024),XtNumber(seg13_3024),XtNumber(seg14_3024),
    XtNumber(seg15_3024),XtNumber(seg16_3024),XtNumber(seg17_3024),
};
static XPoint seg0_3025[] = {
    {-8,-12},{-1,-1},{-1,9},
};
static XPoint seg1_3025[] = {
    {-7,-12},{0,-1},{0,8},
};
static XPoint seg2_3025[] = {
    {-6,-12},{1,-1},{1,9},
};
static XPoint seg3_3025[] = {
    {7,-11},{1,-1},
};
static XPoint seg4_3025[] = {
    {-10,-12},{-3,-12},
};
static XPoint seg5_3025[] = {
    {4,-12},{10,-12},
};
static XPoint seg6_3025[] = {
    {-4,9},{4,9},
};
static XPoint seg7_3025[] = {
    {-9,-12},{-7,-11},
};
static XPoint seg8_3025[] = {
    {-4,-12},{-6,-11},
};
static XPoint seg9_3025[] = {
    {5,-12},{7,-11},
};
static XPoint seg10_3025[] = {
    {9,-12},{7,-11},
};
static XPoint seg11_3025[] = {
    {-1,8},{-3,9},
};
static XPoint seg12_3025[] = {
    {-1,7},{-2,9},
};
```

```
static XPoint seg13_3025[] = {
    {1,7},{2,9},
};
static XPoint seg14_3025[] = {
    {1,8},{3,9},
};
static XPoint *char3025[] = {
    seg0_3025,seg1_3025,seg2_3025,seg3_3025,seg4_3025,
    seg5_3025,seg6_3025,seg7_3025,seg8_3025,seg9_3025,
    seg10_3025,seg11_3025,seg12_3025,seg13_3025,seg14_3025,
    NULL,
};
static int char_p3025[] = {
    XtNumber(seg0_3025), XtNumber(seg1_3025), XtNumber(seg2_3025),
    XtNumber(seg3_3025), XtNumber(seg4_3025), XtNumber(seg5_3025),
    XtNumber(seg6_3025), XtNumber(seg7_3025), XtNumber(seg8_3025),
    XtNumber(seg9_3025), XtNumber(seg10_3025),XtNumber(seg11_3025),
    XtNumber(seg12_3025),XtNumber(seg13_3025),XtNumber(seg14_3025),
};
static XPoint seg0_3026[] = {
    {7,-12},{-7,-12},{-7,-6},
};
static XPoint seg1_3026[] = {
    {5,-12},{-7,9},
};
static XPoint seg2_3026[] = {
    {6,-12},{-6,9},
};
static XPoint seg3_3026[] = {
    {7,-12},{-5,9},
};
static XPoint seg4_3026[] = {
    {-7,9},{7,9},{7,3},
};
static XPoint seg5_3026[] = {
    {-6,-12},{-7,-6},
};
static XPoint seg6_3026[] = {
    {-5,-12},{-7,-9},
};
static XPoint seg7_3026[] = {
    {-4,-12},{-7,-10},
};
static XPoint seg8_3026[] = {
    {-2,-12},{-7,-11},
};
static XPoint seg9_3026[] = {
    {2,9},{7,8},
};
static XPoint seg10_3026[] = {
    {4,9},{7,7},
};
```

continues

Listing C.4 Continued

```
static XPoint seg11_3026[] = {
    {5,9},{7,6},
};
static XPoint seg12_3026[] = {
    {6,9},{7,3},
};
static XPoint *char3026[] = {
    seg0_3026,seg1_3026,seg2_3026,seg3_3026,seg4_3026,
    seg5_3026,seg6_3026,seg7_3026,seg8_3026,seg9_3026,
    seg10_3026,seg11_3026,seg12_3026,
    NULL,
};
static int char_p3026[] = {
    XtNumber(seg0_3026),XtNumber(seg1_3026), XtNumber(seg2_3026),
    XtNumber(seg3_3026),XtNumber(seg4_3026), XtNumber(seg5_3026),
    XtNumber(seg6_3026),XtNumber(seg7_3026), XtNumber(seg8_3026),
    XtNumber(seg9_3026),XtNumber(seg10_3026),XtNumber(seg11_3026),
    XtNumber(seg12_3026),
};
static XPoint seg0_3051[] = {
    {3,-12},{-9,8},
};
static XPoint seg1_3051[] = {
    {1,-8},{2,9},
};
static XPoint seg2_3051[] = {
    {2,-10},{3,8},
};
static XPoint seg3_3051[] = {
    {3,-12},{3,-10},{4,7},{4,9},
};
static XPoint seg4_3051[] = {
    {-6,3},{2,3},
};
static XPoint seg5_3051[] = {
    {-12,9},{-6,9},
};
static XPoint seg6_3051[] = {
    {-1,9},{6,9},
};
static XPoint seg7_3051[] = {
    {-9,8},{-11,9},
};
static XPoint seg8_3051[] = {
    {-9,8},{-7,9},
};
static XPoint seg9_3051[] = {
    {2,8},{0,9},
};
static XPoint seg10_3051[] = {
    {2,7},{1,9},
};
```

```
static XPoint seg11_3051[] = {
    {4,7},{5,9},
};
static XPoint *char3051[] = {
    seg0_3051,seg1_3051,seg2_3051,seg3_3051,seg4_3051,
    seg5_3051,seg6_3051,seg7_3051,seg8_3051,seg9_3051,
    seg10_3051,seg11_3051,
    NULL,
};
static int char_p3051[] = {
    XtNumber(seg0_3051),XtNumber(seg1_3051), XtNumber(seg2_3051),
    XtNumber(seg3_3051),XtNumber(seg4_3051), XtNumber(seg5_3051),
    XtNumber(seg6_3051),XtNumber(seg7_3051), XtNumber(seg8_3051),
    XtNumber(seg9_3051),XtNumber(seg10_3051),XtNumber(seg11_3051),
};
static XPoint seg0_3052[] = {
    {-3,-12},{-9,9},
};
static XPoint seg1_3052[] = {
    {-2,-12},{-8,9},
};
static XPoint seg2_3052[] = {
    {-1,-12},{-7,9},
};
static XPoint seg3_3052[] = {
    {-6,-12},{5,-12},{8,-11},{9,-9},{9,-7},{8,-4},{7,-3},{4,-2},
};
static XPoint seg4_3052[] = {
    {7,-11},{8,-9},{8,-7},{7,-4},{6,-3},
};
static XPoint seg5_3052[] = {
    {5,-12},{6,-11},{7,-9},{7,-7},{6,-4},{4,-2},
};
static XPoint seg6_3052[] = {
    {-4,-2},{4,-2},{6,-1},{7,1},{7,3},{6,6},{4,8},{0,9},
    {-12,9},
};
static XPoint seg7_3052[] = {
    {5,-1},{6,1},{6,3},{5,6},{3,8},
};
static XPoint seg8_3052[] = {
    {4,-2},{5,0},{5,3},{4,6},{2,8},{0,9},
};
static XPoint seg9_3052[] = {
    {-5,-12},{-2,-11},
};
static XPoint seg10_3052[] = {
    {-4,-12},{-3,-10},
};
static XPoint seg11_3052[] = {
    {0,-12},{-2,-10},
};
```

continues

Listing C.4 Continued

```
static XPoint seg12_3052[] = {
    {1,-12},{-2,-11},
};
static XPoint seg13_3052[] = {
    {-8,8},{-11,9},
};
static XPoint seg14_3052[] = {
    {-8,7},{-10,9},
};
static XPoint seg15_3052[] = {
    {-7,7},{-6,9},
};
static XPoint seg16_3052[] = {
    {-8,8},{-5,9},
};
static XPoint *char3052[] = {
    seg0_3052,seg1_3052,seg2_3052,seg3_3052,seg4_3052,
    seg5_3052,seg6_3052,seg7_3052,seg8_3052,seg9_3052,
    seg10_3052,seg11_3052,seg12_3052,seg13_3052,seg14_3052,
    seg15_3052,seg16_3052,
    NULL,
};
static int char_p3052[] = {
    XtNumber(seg0_3052), XtNumber(seg1_3052), XtNumber(seg2_3052),
    XtNumber(seg3_3052), XtNumber(seg4_3052), XtNumber(seg5_3052),
    XtNumber(seg6_3052), XtNumber(seg7_3052), XtNumber(seg8_3052),
    XtNumber(seg9_3052), XtNumber(seg10_3052),XtNumber(seg11_3052),
    XtNumber(seg12_3052),XtNumber(seg13_3052),XtNumber(seg14_3052),
    XtNumber(seg15_3052),XtNumber(seg16_3052),
};
static XPoint seg0_3053[] = {
    {8,-10},{9,-10},{10,-12},{9,-6},{9,-8},{8,-10},
    {7,-11},{5,-12},{2,-12},{-1,-11},{-3,-9},{-5,-6},{-6,-3},
    {-7,1},{-7,4},{-6,7},{-5,8},{-2,9},{1,9},{3,8},{5,6},{6,4},
};
static XPoint seg1_3053[] = {
    {-1,-10},{-3,-8},{-4,-6},{-5,-3},{-6,1},{-6,5},{-5,7},
};
static XPoint seg2_3053[] = {
    {2,-12},{0,-11},{-2,-8},{-3,-6},{-4,-3},{-5,1},{-5,6},{-4,8},
    {-2,9},
};
static XPoint *char3053[] = {
    seg0_3053,seg1_3053,seg2_3053,
    NULL,
};
static int char_p3053[] = {
    XtNumber(seg0_3053),XtNumber(seg1_3053),XtNumber(seg2_3053),
};
```

```
static XPoint seg0_3054[] = {
    {-3,-12},{-9,9},
};
static XPoint seg1_3054[] = {
    {-2,-12},{-8,9},
};
static XPoint seg2_3054[] = {
    {-1,-12},{-7,9},
};
static XPoint seg3_3054[] = {
    {-6,-12},{3,-12},{6,-11},{7,-10},{8,-7},{8,-3},{7,1},{5,5},
    {3,7},{1,8},{-3,9},{-12,9},
};
static XPoint seg4_3054[] = {
    {5,-11},{6,-10},{7,-7},{7,-3},{6,1},{4,5},{2,7},
};
static XPoint seg5_3054[] = {
    {3,-12},{5,-10},{6,-7},{6,-3},{5,1},{3,5},{0,8},{-3,9},
};
static XPoint seg6_3054[] = {
    {-5,-12},{-2,-11},
};
static XPoint seg7_3054[] = {
    {-4,-12},{-3,-10},
};
static XPoint seg8_3054[] = {
    {0,-12},{-2,-10},
};
static XPoint seg9_3054[] = {
    {1,-12},{-2,-11},
};
static XPoint seg10_3054[] = {
    {-8,8},{-11,9},
};
static XPoint seg11_3054[] = {
    {-8,7},{-10,9},
};
static XPoint seg12_3054[] = {
    {-7,7},{-6,9},
};
static XPoint seg13_3054[] = {
    {-8,8},{-5,9},
};
static XPoint *char3054[] = {
    seg0_3054,seg1_3054,seg2_3054,seg3_3054,seg4_3054,
    seg5_3054,seg6_3054,seg7_3054,seg8_3054,seg9_3054,
    seg10_3054,seg11_3054,seg12_3054,seg13_3054,
    NULL,
};
static int char_p3054[] = {
    XtNumber(seg0_3054), XtNumber(seg1_3054), XtNumber(seg2_3054),
    XtNumber(seg3_3054), XtNumber(seg4_3054), XtNumber(seg5_3054),
    XtNumber(seg6_3054), XtNumber(seg7_3054), XtNumber(seg8_3054),
    XtNumber(seg9_3054), XtNumber(seg10_3054),XtNumber(seg11_3054),
```

C

continues

Listing C.4 Continued

```
    XtNumber(seg12_3054),XtNumber(seg13_3054),
};
static XPoint seg0_3055[] = {
    {-3,-12},{-9,9},
};
static XPoint seg1_3055[] = {
    {-2,-12},{-8,9},
};
static XPoint seg2_3055[] = {
    {-1,-12},{-7,9},
};
static XPoint seg3_3055[] = {
    {3,-6},{1,2},
};
static XPoint seg4_3055[] = {
    {-6,-12},{9,-12},{8,-6},
};
static XPoint seg5_3055[] = {
    {-4,-2},{2,-2},
};
static XPoint seg6_3055[] = {
    {-12,9},{3,9},{5,4},
};
static XPoint seg7_3055[] = {
    {-5,-12},{-2,-11},
};
static XPoint seg8_3055[] = {
    {-4,-12},{-3,-10},
};
static XPoint seg9_3055[] = {
    {0,-12},{-2,-10},
};
static XPoint seg10_3055[] = {
    {1,-12},{-2,-11},
};
static XPoint seg11_3055[] = {
    {5,-12},{8,-11},
};
static XPoint seg12_3055[] = {
    {6,-12},{8,-10},
};
static XPoint seg13_3055[] = {
    {7,-12},{8,-9},
};
static XPoint seg14_3055[] = {
    {8,-12},{8,-6},
};
static XPoint seg15_3055[] = {
    {3,-6},{1,-2},{1,2},
};
```

```
static XPoint seg16_3055[] = {
    {2,-4},{0,-2},{1,0},
};
static XPoint seg17_3055[] = {
    {2,-3},{-1,-2},{1,-1},
};
static XPoint seg18_3055[] = {
    {-8,8},{-11,9},
};
static XPoint seg19_3055[] = {
    {-8,7},{-10,9},
};
static XPoint seg20_3055[] = {
    {-7,7},{-6,9},
};
static XPoint seg21_3055[] = {
    {-8,8},{-5,9},
};
static XPoint seg22_3055[] = {
    {-2,9},{3,8},
};
static XPoint seg23_3055[] = {
    {0,9},{3,7},
};
static XPoint seg24_3055[] = {
    {3,7},{5,4},
};
static XPoint *char3055[] = {
    seg0_3055,seg1_3055,seg2_3055,seg3_3055,seg4_3055,
    seg5_3055,seg6_3055,seg7_3055,seg8_3055,seg9_3055,
    seg10_3055,seg11_3055,seg12_3055,seg13_3055,seg14_3055,
    seg15_3055,seg16_3055,seg17_3055,seg18_3055,seg19_3055,
    seg20_3055,seg21_3055,seg22_3055,seg23_3055,seg24_3055,
    NULL,
};
static int char_p3055[] = {
    XtNumber(seg0_3055), XtNumber(seg1_3055), XtNumber(seg2_3055),
    XtNumber(seg3_3055), XtNumber(seg4_3055), XtNumber(seg5_3055),
    XtNumber(seg6_3055), XtNumber(seg7_3055), XtNumber(seg8_3055),
    XtNumber(seg9_3055), XtNumber(seg10_3055),XtNumber(seg11_3055),
    XtNumber(seg12_3055),XtNumber(seg13_3055),XtNumber(seg14_3055),
    XtNumber(seg15_3055),XtNumber(seg16_3055),XtNumber(seg17_3055),
    XtNumber(seg18_3055),XtNumber(seg19_3055),XtNumber(seg20_3055),
    XtNumber(seg21_3055),XtNumber(seg22_3055),XtNumber(seg23_3055),
    XtNumber(seg24_3055),
};
static XPoint seg0_3056[] = {
    {-3,-12},{-9,9},
};
static XPoint seg1_3056[] = {
    {-2,-12},{-8,9},
};
```

continues

Listing C.4 Continued

```
static XPoint seg2_3056[] = {
    {-1,-12},{-7,9},
};
static XPoint seg3_3056[] = {
    {3,-6},{1,2},
};
static XPoint seg4_3056[] = {
    {-6,-12},{9,-12},{8,-6},
};
static XPoint seg5_3056[] = {
    {-4,-2},{2,-2},
};
static XPoint seg6_3056[] = {
    {-12,9},{-4,9},
};
static XPoint seg7_3056[] = {
    {-5,-12},{-2,-11},
};
static XPoint seg8_3056[] = {
    {-4,-12},{-3,-10},
};
static XPoint seg9_3056[] = {
    {0,-12},{-2,-10},
};
static XPoint seg10_3056[] = {
    {1,-12},{-2,-11},
};
static XPoint seg11_3056[] = {
    {5,-12},{8,-11},
};
static XPoint seg12_3056[] = {
    {6,-12},{8,-10},
};
static XPoint seg13_3056[] = {
    {7,-12},{8,-9},
};
static XPoint seg14_3056[] = {
    {8,-12},{8,-6},
};
static XPoint seg15_3056[] = {
    {3,-6},{1,-2},{1,2},
};
static XPoint seg16_3056[] = {
    {2,-4},{0,-2},{1,0},
};
static XPoint seg17_3056[] = {
    {2,-3},{-1,-2},{1,-1},
};
static XPoint seg18_3056[] = {
    {-8,8},{-11,9},
};
```

```
static XPoint seg19_3056[] = {
    {-8,7},{-10,9},
};
static XPoint seg20_3056[] = {
    {-7,7},{-6,9},
};
static XPoint seg21_3056[] = {
    {-8,8},{-5,9},
};
static XPoint *char3056[] = {
    seg0_3056,seg1_3056,seg2_3056,seg3_3056,seg4_3056,
    seg5_3056,seg6_3056,seg7_3056,seg8_3056,seg9_3056,
    seg10_3056,seg11_3056,seg12_3056,seg13_3056,seg14_3056,
    seg15_3056,seg16_3056,seg17_3056,seg18_3056,seg19_3056,
    seg20_3056,seg21_3056,
    NULL,
};
static int char_p3056[] = {
    XtNumber(seg0_3056), XtNumber(seg1_3056), XtNumber(seg2_3056),
    XtNumber(seg3_3056), XtNumber(seg4_3056), XtNumber(seg5_3056),
    XtNumber(seg6_3056), XtNumber(seg7_3056), XtNumber(seg8_3056),
    XtNumber(seg9_3056), XtNumber(seg10_3056),XtNumber(seg11_3056),
    XtNumber(seg12_3056),XtNumber(seg13_3056),XtNumber(seg14_3056),
    XtNumber(seg15_3056),XtNumber(seg16_3056),XtNumber(seg17_3056),
    XtNumber(seg18_3056),XtNumber(seg19_3056),XtNumber(seg20_3056),
    XtNumber(seg21_3056),
};
static XPoint seg0_3057[] = {
    {8,-10},{9,-10},{10,-12},{9,-6},{9,-8},{8,-10},
    {7,-11},{5,-12},{2,-12},{-1,-11},{-3,-9},{-5,-6},{-6,-3},
    {-7,1},{-7,4},{-6,7},{-5,8},{-2,9},{0,9},{3,8},{5,6},{7,2},
};
static XPoint seg1_3057[] = {
    {-1,-10},{-3,-8},{-4,-6},{-5,-3},{-6,1},{-6,5},{-5,7},
};
static XPoint seg2_3057[] = {
    {4,6},{5,5},{6,2},
};
static XPoint seg3_3057[] = {
    {2,-12},{0,-11},{-2,-8},{-3,-6},{-4,-3},{-5,1},{-5,6},{-4,8},
    {-2,9},
};
static XPoint seg4_3057[] = {
    {0,9},{2,8},{4,5},{5,2},
};
static XPoint seg5_3057[] = {
    {2,2},{10,2},
};
static XPoint seg6_3057[] = {
    {3,2},{5,3},
};
static XPoint seg7_3057[] = {
    {4,2},{5,5},
};
```

continues

Listing C.4 Continued

```
static XPoint seg8_3057[] = {
    {8,2},{6,4},
};
static XPoint seg9_3057[] = {
    {9,2},{6,3},
};
static XPoint *char3057[] = {
    seg0_3057,seg1_3057,seg2_3057,seg3_3057,seg4_3057,seg5_3057,
    seg6_3057,seg7_3057,seg8_3057,seg9_3057,
    NULL,
};
static int char_p3057[] = {
    XtNumber(seg0_3057),XtNumber(seg1_3057),XtNumber(seg2_3057),
    XtNumber(seg3_3057),XtNumber(seg4_3057),XtNumber(seg5_3057),
    XtNumber(seg6_3057),XtNumber(seg7_3057),XtNumber(seg8_3057),
    XtNumber(seg9_3057),
};
static XPoint seg0_3058[] = {
    {-4,-12},{-10,9},
};
static XPoint seg1_3058[] = {
    {-3,-12},{-9,9},
};
static XPoint seg2_3058[] = {
    {-2,-12},{-8,9},
};
static XPoint seg3_3058[] = {
    {8,-12},{2,9},
};
static XPoint seg4_3058[] = {
    {9,-12},{3,9},
};
static XPoint seg5_3058[] = {
    {10,-12},{4,9},
};
static XPoint seg6_3058[] = {
    {-7,-12},{1,-12},
};
static XPoint seg7_3058[] = {
    {5,-12},{13,-12},
};
static XPoint seg8_3058[] = {
    {-6,-2},{6,-2},
};
static XPoint seg9_3058[] = {
    {-13,9},{-5,9},
};
static XPoint seg10_3058[] = {
    {-1,9},{7,9},
};
```

```
static XPoint seg11_3058[] = {
    {-6,-12},{-3,-11},
};
static XPoint seg12_3058[] = {
    {-5,-12},{-4,-10},
};
static XPoint seg13_3058[] = {
    {-1,-12},{-3,-10},
};
static XPoint seg14_3058[] = {
    {0,-12},{-3,-11},
};
static XPoint seg15_3058[] = {
    {6,-12},{9,-11},
};
static XPoint seg16_3058[] = {
    {7,-12},{8,-10},
};
static XPoint seg17_3058[] = {
    {11,-12},{9,-10},
};
static XPoint seg18_3058[] = {
    {12,-12},{9,-11},
};
static XPoint seg19_3058[] = {
    {-9,8},{-12,9},
};
static XPoint seg20_3058[] = {
    {-9,7},{-11,9},
};
static XPoint seg21_3058[] = {
    {-8,7},{-7,9},
};
static XPoint seg22_3058[] = {
    {-9,8},{-6,9},
};
static XPoint seg23_3058[] = {
    {3,8},{0,9},
};
static XPoint seg24_3058[] = {
    {3,7},{1,9},
};
static XPoint seg25_3058[] = {
    {4,7},{5,9},
};
static XPoint seg26_3058[] = {
    {3,8},{6,9},
};
static XPoint *char3058[] = {
    seg0_3058,seg1_3058,seg2_3058,seg3_3058,seg4_3058,
    seg5_3058,seg6_3058,seg7_3058,seg8_3058,seg9_3058,
    seg10_3058,seg11_3058,seg12_3058,seg13_3058,seg14_3058,
```

continues

Listing C.4 Continued

```
    seg15_3058,seg16_3058,seg17_3058,seg18_3058,seg19_3058,
    seg20_3058,seg21_3058,seg22_3058,seg23_3058,seg24_3058,
    seg25_3058,seg26_3058,
    NULL,
};
static int char_p3058[] = {
    XtNumber(seg0_3058), XtNumber(seg1_3058), XtNumber(seg2_3058),
    XtNumber(seg3_3058), XtNumber(seg4_3058), XtNumber(seg5_3058),
    XtNumber(seg6_3058), XtNumber(seg7_3058), XtNumber(seg8_3058),
    XtNumber(seg9_3058), XtNumber(seg10_3058),XtNumber(seg11_3058),
    XtNumber(seg12_3058),XtNumber(seg13_3058),XtNumber(seg14_3058),
    XtNumber(seg15_3058),XtNumber(seg16_3058),XtNumber(seg17_3058),
    XtNumber(seg18_3058),XtNumber(seg19_3058),XtNumber(seg20_3058),
    XtNumber(seg21_3058),XtNumber(seg22_3058),XtNumber(seg23_3058),
    XtNumber(seg24_3058),XtNumber(seg25_3058),XtNumber(seg26_3058),
};
static XPoint seg0_3059[] = {
    {2,-12},{-4,9},
};
static XPoint seg1_3059[] = {
    {3,-12},{-3,9},
};
static XPoint seg2_3059[] = {
    {4,-12},{-2,9},
};
static XPoint seg3_3059[] = {
    {-1,-12},{7,-12},
};
static XPoint seg4_3059[] = {
    {-7,9},{1,9},
};
static XPoint seg5_3059[] = {
    {0,-12},{3,-11},
};
static XPoint seg6_3059[] = {
    {1,-12},{2,-10},
};
static XPoint seg7_3059[] = {
    {5,-12},{3,-10},
};
static XPoint seg8_3059[] = {
    {6,-12},{3,-11},
};
static XPoint seg9_3059[] = {
    {-3,8},{-6,9},
};
static XPoint seg10_3059[] = {
    {-3,7},{-5,9},
};
```

```
static XPoint seg11_3059[] = {
    {-2,7},{-1,9},
};
static XPoint seg12_3059[] = {
    {-3,8},{0,9},
};
static XPoint *char3059[] = {
    seg0_3059,seg1_3059,seg2_3059,seg3_3059,seg4_3059,
    seg5_3059,seg6_3059,seg7_3059,seg8_3059,seg9_3059,
    seg10_3059,seg11_3059,seg12_3059,
    NULL,
};
static int char_p3059[] = {
    XtNumber(seg0_3059),XtNumber(seg1_3059), XtNumber(seg2_3059),
    XtNumber(seg3_3059),XtNumber(seg4_3059), XtNumber(seg5_3059),
    XtNumber(seg6_3059),XtNumber(seg7_3059), XtNumber(seg8_3059),
    XtNumber(seg9_3059),XtNumber(seg10_3059),XtNumber(seg11_3059),
    XtNumber(seg12_3059),
};
static XPoint seg0_3060[] = {
    {5,-12},{0,5},{-1,7},{-3,9},
};
static XPoint seg1_3060[] = {
    {6,-12},{2,1},{1,4},{0,6},
};
static XPoint seg2_3060[] = {
    {7,-12},{3,1},{1,6},{-1,8},{-3,9},{-5,9},{-7,8},{-8,6},
    {-8,4},{-7,3},{-6,3},{-5,4},{-5,5},{-6,6},{-7,6},
};
static XPoint seg3_3060[] = {
    {-7,4},{-7,5},{-6,5},{-6,4},{-7,4},
};
static XPoint seg4_3060[] = {
    {2,-12},{10,-12},
};
static XPoint seg5_3060[] = {
    {3,-12},{6,-11},
};
static XPoint seg6_3060[] = {
    {4,-12},{5,-10},
};
static XPoint seg7_3060[] = {
    {8,-12},{6,-10},
};
static XPoint seg8_3060[] = {
    {9,-12},{6,-11},
};
static XPoint *char3060[] = {
    seg0_3060,seg1_3060,seg2_3060,seg3_3060,seg4_3060,seg5_3060,
    seg6_3060,seg7_3060,seg8_3060,
    NULL,
};
static int char_p3060[] = {
    XtNumber(seg0_3060),XtNumber(seg1_3060),XtNumber(seg2_3060),
```

continues

Listing C.4 Continued

```
    XtNumber(seg3_3060),XtNumber(seg4_3060),XtNumber(seg5_3060),
    XtNumber(seg6_3060),XtNumber(seg7_3060),XtNumber(seg8_3060),
};
static XPoint seg0_3061[] = {
    {-3,-12},{-9,9},
};
static XPoint seg1_3061[] = {
    {-2,-12},{-8,9},
};
static XPoint seg2_3061[] = {
    {-1,-12},{-7,9},
};
static XPoint seg3_3061[] = {
    {10,-11},{-5,0},
};
static XPoint seg4_3061[] = {
    {-1,-3},{3,9},
};
static XPoint seg5_3061[] = {
    {0,-3},{4,9},
};
static XPoint seg6_3061[] = {
    {1,-4},{5,8},
};
static XPoint seg7_3061[] = {
    {-6,-12},{2,-12},
};
static XPoint seg8_3061[] = {
    {7,-12},{13,-12},
};
static XPoint seg9_3061[] = {
    {-12,9},{-4,9},
};
static XPoint seg10_3061[] = {
    {0,9},{7,9},
};
static XPoint seg11_3061[] = {
    {-5,-12},{-2,-11},
};
static XPoint seg12_3061[] = {
    {-4,-12},{-3,-10},
};
static XPoint seg13_3061[] = {
    {0,-12},{-2,-10},
};
static XPoint seg14_3061[] = {
    {1,-12},{-2,-11},
};
static XPoint seg15_3061[] = {
    {8,-12},{10,-11},
};
```

```
static XPoint seg16_3061[] = {
    {12,-12},{10,-11},
};
static XPoint seg17_3061[] = {
    {-8,8},{-11,9},
};
static XPoint seg18_3061[] = {
    {-8,7},{-10,9},
};
static XPoint seg19_3061[] = {
    {-7,7},{-6,9},
};
static XPoint seg20_3061[] = {
    {-8,8},{-5,9},
};
static XPoint seg21_3061[] = {
    {3,8},{1,9},
};
static XPoint seg22_3061[] = {
    {3,7},{2,9},
};
static XPoint seg23_3061[] = {
    {4,7},{6,9},
};
static XPoint *char3061[] = {
    seg0_3061,seg1_3061,seg2_3061,seg3_3061,seg4_3061,
    seg5_3061,seg6_3061,seg7_3061,seg8_3061,seg9_3061,
    seg10_3061,seg11_3061,seg12_3061,seg13_3061,seg14_3061,
    seg15_3061,seg16_3061,seg17_3061,seg18_3061,seg19_3061,
    seg20_3061,seg21_3061,seg22_3061,seg23_3061,
    NULL,
};
static int char_p3061[] = {
    XtNumber(seg0_3061), XtNumber(seg1_3061), XtNumber(seg2_3061),
    XtNumber(seg3_3061), XtNumber(seg4_3061), XtNumber(seg5_3061),
    XtNumber(seg6_3061), XtNumber(seg7_3061), XtNumber(seg8_3061),
    XtNumber(seg9_3061), XtNumber(seg10_3061),XtNumber(seg11_3061),
    XtNumber(seg12_3061),XtNumber(seg13_3061),XtNumber(seg14_3061),
    XtNumber(seg15_3061),XtNumber(seg16_3061),XtNumber(seg17_3061),
    XtNumber(seg18_3061),XtNumber(seg19_3061),XtNumber(seg20_3061),
    XtNumber(seg21_3061),XtNumber(seg22_3061),XtNumber(seg23_3061),
};
static XPoint seg0_3062[] = {
    {-1,-12},{-7,9},
};
static XPoint seg1_3062[] = {
    {0,-12},{-6,9},
};
static XPoint seg2_3062[] = {
    {1,-12},{-5,9},
};
static XPoint seg3_3062[] = {
    {-4,-12},{4,-12},
};
```

continues

Listing C.4 Continued

```
static XPoint seg4_3062[] = {
    {-10,9},{5,9},{7,3},
};
static XPoint seg5_3062[] = {
    {-3,-12},{0,-11},
};
static XPoint seg6_3062[] = {
    {-2,-12},{-1,-10},
};
static XPoint seg7_3062[] = {
    {2,-12},{0,-10},
};
static XPoint seg8_3062[] = {
    {3,-12},{0,-11},
};
static XPoint seg9_3062[] = {
    {-6,8},{-9,9},
};
static XPoint seg10_3062[] = {
    {-6,7},{-8,9},
};
static XPoint seg11_3062[] = {
    {-5,7},{-4,9},
};
static XPoint seg12_3062[] = {
    {-6,8},{-3,9},
};
static XPoint seg13_3062[] = {
    {0,9},{5,8},
};
static XPoint seg14_3062[] = {
    {2,9},{6,6},
};
static XPoint seg15_3062[] = {
    {4,9},{7,3},
};
static XPoint *char3062[] = {
    seg0_3062,seg1_3062,seg2_3062,seg3_3062,seg4_3062,
    seg5_3062,seg6_3062,seg7_3062,seg8_3062,seg9_3062,
    seg10_3062,seg11_3062,seg12_3062,seg13_3062,seg14_3062,
    seg15_3062,
    NULL,
};
static int char_p3062[] = {
    XtNumber(seg0_3062), XtNumber(seg1_3062), XtNumber(seg2_3062),
    XtNumber(seg3_3062), XtNumber(seg4_3062), XtNumber(seg5_3062),
    XtNumber(seg6_3062), XtNumber(seg7_3062), XtNumber(seg8_3062),
    XtNumber(seg9_3062), XtNumber(seg10_3062),XtNumber(seg11_3062),
    XtNumber(seg12_3062),XtNumber(seg13_3062),XtNumber(seg14_3062),
    XtNumber(seg15_3062),
};
static XPoint seg0_3063[] = {
    {-5,-12},{-11,8},
};
static XPoint seg1_3063[] = {
    {-5,-11},{-4,7},{-4,9},
};
static XPoint seg2_3063[] = {
    {-4,-12},{-3,7},
};
static XPoint seg3_3063[] = {
    {-3,-12},{-2,6},
};
static XPoint seg4_3063[] = {
    {9,-12},{-2,6},{-4,9},
};
static XPoint seg5_3063[] = {
    {9,-12},{3,9},
};
static XPoint seg6_3063[] = {
    {10,-12},{4,9},
};
static XPoint seg7_3063[] = {
    {11,-12},{5,9},
};
static XPoint seg8_3063[] = {
    {-8,-12},{-3,-12},
};
static XPoint seg9_3063[] = {
    {9,-12},{14,-12},
};
static XPoint seg10_3063[] = {
    {-14,9},{-8,9},
};
static XPoint seg11_3063[] = {
    {0,9},{8,9},
};
static XPoint seg12_3063[] = {
    {-7,-12},{-5,-11},
};
static XPoint seg13_3063[] = {
    {-6,-12},{-5,-10},
};
static XPoint seg14_3063[] = {
    {12,-12},{10,-10},
};
static XPoint seg15_3063[] = {
    {13,-12},{10,-11},
};
static XPoint seg16_3063[] = {
    {-11,8},{-13,9},
};
```

continues

Listing C.4 Continued

```
static XPoint seg17_3063[] = {
    {-11,8},{-9,9},
};
static XPoint seg18_3063[] = {
    {4,8},{1,9},
};
static XPoint seg19_3063[] = {
    {4,7},{2,9},
};
static XPoint seg20_3063[] = {
    {5,7},{6,9},
};
static XPoint seg21_3063[] = {
    {4,8},{7,9},
};
static XPoint *char3063[] = {
    seg0_3063,seg1_3063,seg2_3063,seg3_3063,seg4_3063,
    seg5_3063,seg6_3063,seg7_3063,seg8_3063,seg9_3063,
    seg10_3063,seg11_3063,seg12_3063,seg13_3063,seg14_3063,
    seg15_3063,seg16_3063,seg17_3063,seg18_3063,seg19_3063,
    seg20_3063,seg21_3063,
    NULL,
};
static int char_p3063[] = {
    XtNumber(seg0_3063), XtNumber(seg1_3063), XtNumber(seg2_3063),
    XtNumber(seg3_3063), XtNumber(seg4_3063), XtNumber(seg5_3063),
    XtNumber(seg6_3063), XtNumber(seg7_3063), XtNumber(seg8_3063),
    XtNumber(seg9_3063), XtNumber(seg10_3063),XtNumber(seg11_3063),
    XtNumber(seg12_3063),XtNumber(seg13_3063),XtNumber(seg14_3063),
    XtNumber(seg15_3063),XtNumber(seg16_3063),XtNumber(seg17_3063),
    XtNumber(seg18_3063),XtNumber(seg19_3063),XtNumber(seg20_3063),
    XtNumber(seg21_3063),
};
static XPoint seg0_3064[] = {
    {-3,-12},{-9,8},
};
static XPoint seg1_3064[] = {
    {-3,-12},{4,9},
};
static XPoint seg2_3064[] = {
    {-2,-12},{4,6},
};
static XPoint seg3_3064[] = {
    {-1,-12},{5,6},
};
static XPoint seg4_3064[] = {
    {10,-11},{5,6},{4,9},
};
static XPoint seg5_3064[] = {
    {-6,-12},{-1,-12},
};
```

```
static XPoint seg6_3064[] = {
    {7,-12},{13,-12},
};
static XPoint seg7_3064[] = {
    {-12,9},{-6,9},
};
static XPoint seg8_3064[] = {
    {-5,-12},{-2,-11},
};
static XPoint seg9_3064[] = {
    {-4,-12},{-2,-10},
};
static XPoint seg10_3064[] = {
    {8,-12},{10,-11},
};
static XPoint seg11_3064[] = {
    {12,-12},{10,-11},
};
static XPoint seg12_3064[] = {
    {-9,8},{-11,9},
};
static XPoint seg13_3064[] = {
    {-9,8},{-7,9},
};
static XPoint *char3064[] = {
    seg0_3064,seg1_3064,seg2_3064,seg3_3064,seg4_3064,
    seg5_3064,seg6_3064,seg7_3064,seg8_3064,seg9_3064,
    seg10_3064,seg11_3064,seg12_3064,seg13_3064,
    NULL,
};
static int char_p3064[] = {
    XtNumber(seg0_3064), XtNumber(seg1_3064), XtNumber(seg2_3064),
    XtNumber(seg3_3064), XtNumber(seg4_3064), XtNumber(seg5_3064),
    XtNumber(seg6_3064), XtNumber(seg7_3064), XtNumber(seg8_3064),
    XtNumber(seg9_3064), XtNumber(seg10_3064),XtNumber(seg11_3064),
    XtNumber(seg12_3064),XtNumber(seg13_3064),
};
static XPoint seg0_3065[] = {
    {1,-12},{-2,-11},{-4,-9},{-6,-6},{-7,-3},{-8,1},
    {-8,4},{-7,7},{-6,8},{-4,9},{-1,9},{2,8},{4,6},{6,3},{7,0},
    {8,-4},{8,-7},{7,-10},{6,-11},{4,-12},{1,-12},
};
static XPoint seg1_3065[] = {
    {-3,-9},{-5,-6},{-6,-3},{-7,1},{-7,5},{-6,7},
};
static XPoint seg2_3065[] = {
    {3,6},{5,3},{6,0},{7,-4},{7,-8},{6,-10},
};
static XPoint seg3_3065[] = {
    {1,-12},{-1,-11},{-3,-8},{-4,-6},{-5,-3},{-6,1},{-6,6},{-5,8},
    {-4,9},
};
```

continues

Listing C.4 Continued

```
static XPoint seg4_3065[] = {
    {-1,9},{1,8},{3,5},{4,3},{5,0},{6,-4},{6,-9},{5,-11},
    {4,-12},
};
static XPoint *char3065[] = {
    seg0_3065,seg1_3065,seg2_3065,seg3_3065,seg4_3065,
    NULL,
};
static int char_p3065[] = {
    XtNumber(seg0_3065),XtNumber(seg1_3065),XtNumber(seg2_3065),
    XtNumber(seg3_3065),XtNumber(seg4_3065),
};
static XPoint seg0_3066[] = {
    {-3,-12},{-9,9},
};
static XPoint seg1_3066[] = {
    {-2,-12},{-8,9},
};
static XPoint seg2_3066[] = {
    {-1,-12},{-7,9},
};
static XPoint seg3_3066[] = {
    {-6,-12},{6,-12},{9,-11},{10,-9},{10,-7},{9,-4},{7,-2},{3,-1},
    {-5,-1},
};
static XPoint seg4_3066[] = {
    {8,-11},{9,-9},{9,-7},{8,-4},{6,-2},
};
static XPoint seg5_3066[] = {
    {6,-12},{7,-11},{8,-9},{8,-7},{7,-4},{5,-2},{3,-1},
};
static XPoint seg6_3066[] = {
    {-12,9},{-4,9},
};
static XPoint seg7_3066[] = {
    {-5,-12},{-2,-11},
};
static XPoint seg8_3066[] = {
    {-4,-12},{-3,-10},
};
static XPoint seg9_3066[] = {
    {0,-12},{-2,-10},
};
static XPoint seg10_3066[] = {
    {1,-12},{-2,-11},
};
static XPoint seg11_3066[] = {
    {-8,8},{-11,9},
};
static XPoint seg12_3066[] = {
    {-8,7},{-10,9},
};
```

```
static XPoint seg13_3066[] = {
    {-7,7},{-6,9},
};
static XPoint seg14_3066[] = {
    {-8,8},{-5,9},
};
static XPoint *char3066[] = {
    seg0_3066,seg1_3066,seg2_3066,seg3_3066,seg4_3066,
    seg5_3066,seg6_3066,seg7_3066,seg8_3066,seg9_3066,
    seg10_3066,seg11_3066,seg12_3066,seg13_3066,seg14_3066,
    NULL,
};
static int char_p3066[] = {
    XtNumber(seg0_3066),XtNumber(seg1_3066),XtNumber(seg2_3066),
    XtNumber(seg3_3066),XtNumber(seg4_3066),XtNumber(seg5_3066),
    XtNumber(seg6_3066),XtNumber(seg7_3066),XtNumber(seg8_3066),
    XtNumber(seg9_3066),XtNumber(seg10_3066),XtNumber(seg11_3066),
    XtNumber(seg12_3066),XtNumber(seg13_3066),XtNumber(seg14_3066),
};
static XPoint seg0_3067[] = {
    {1,-12},{-2,-11},{-4,-9},{-6,-6},{-7,-3},{-8,1},
    {-8,4},{-7,7},{-6,8},{-4,9},{-1,9},{2,8},{4,6},{6,3},{7,0},
    {8,-4},{8,-7},{7,-10},{6,-11},{4,-12},{1,-12},
};
static XPoint seg1_3067[] = {
    {-3,-9},{-5,-6},{-6,-3},{-7,1},{-7,5},{-6,7},
};
static XPoint seg2_3067[] = {
    {3,6},{5,3},{6,0},{7,-4},{7,-8},{6,-10},
};
static XPoint seg3_3067[] = {
    {1,-12},{-1,-11},{-3,-8},{-4,-6},{-5,-3},{-6,1},{-6,6},{-5,8},
    {-4,9},
};
static XPoint seg4_3067[] = {
    {-1,9},{1,8},{3,5},{4,3},{5,0},{6,-4},{6,-9},{5,-11},
    {4,-12},
};
static XPoint seg5_3067[] = {
    {-6,6},{-5,4},{-3,3},{-2,3},{0,4},{1,6},{2,11},{3,12},
    {4,12},{5,11},
};
static XPoint seg6_3067[] = {
    {2,12},{3,13},{4,13},
};
static XPoint seg7_3067[] = {
    {1,6},{1,13},{2,14},{4,14},{5,11},{5,10},
};
static XPoint *char3067[] = {
    seg0_3067,seg1_3067,seg2_3067,seg3_3067,seg4_3067,seg5_3067,
    seg6_3067,seg7_3067,
    NULL,
};
```

continues

Listing C.4 Continued

```
static int char_p3067[] = {
    XtNumber(seg0_3067),XtNumber(seg1_3067),XtNumber(seg2_3067),
    XtNumber(seg3_3067),XtNumber(seg4_3067),XtNumber(seg5_3067),
    XtNumber(seg6_3067),XtNumber(seg7_3067),
};
static XPoint seg0_3068[] = {
    {-3,-12},{-9,9},
};
static XPoint seg1_3068[] = {
    {-2,-12},{-8,9},
};
static XPoint seg2_3068[] = {
    {-1,-12},{-7,9},
};
static XPoint seg3_3068[] = {
    {-6,-12},{5,-12},{8,-11},{9,-9},{9,-7},{8,-4},{7,-3},{4,-2},
    {-4,-2},
};
static XPoint seg4_3068[] = {
    {7,-11},{8,-9},{8,-7},{7,-4},{6,-3},
};
static XPoint seg5_3068[] = {
    {5,-12},{6,-11},{7,-9},{7,-7},{6,-4},{4,-2},
};
static XPoint seg6_3068[] = {
    {0,-2},{2,-1},{3,0},{5,6},{6,7},{7,7},{8,6},
};
static XPoint seg7_3068[] = {
    {5,7},{6,8},{7,8},
};
static XPoint seg8_3068[] = {
    {3,0},{4,8},{5,9},{7,9},{8,6},{8,5},
};
static XPoint seg9_3068[] = {
    {-12,9},{-4,9},
};
static XPoint seg10_3068[] = {
    {-5,-12},{-2,-11},
};
static XPoint seg11_3068[] = {
    {-4,-12},{-3,-10},
};
static XPoint seg12_3068[] = {
    {0,-12},{-2,-10},
};
static XPoint seg13_3068[] = {
    {1,-12},{-2,-11},
};
```

```
static XPoint seg14_3068[] = {
    {-8,8},{-11,9},
};
static XPoint seg15_3068[] = {
    {-8,7},{-10,9},
};
static XPoint seg16_3068[] = {
    {-7,7},{-6,9},
};
static XPoint seg17_3068[] = {
    {-8,8},{-5,9},
};
static XPoint *char3068[] = {
    seg0_3068,seg1_3068,seg2_3068,seg3_3068,seg4_3068,
    seg5_3068,seg6_3068,seg7_3068,seg8_3068,seg9_3068,
    seg10_3068,seg11_3068,seg12_3068,seg13_3068,seg14_3068,
    seg15_3068,seg16_3068,seg17_3068,
    NULL,
};
static int char_p3068[] = {
    XtNumber(seg0_3068),XtNumber(seg1_3068),XtNumber(seg2_3068),
    XtNumber(seg3_3068),XtNumber(seg4_3068),XtNumber(seg5_3068),
    XtNumber(seg6_3068),XtNumber(seg7_3068),XtNumber(seg8_3068),
    XtNumber(seg9_3068),XtNumber(seg10_3068),XtNumber(seg11_3068),
    XtNumber(seg12_3068),XtNumber(seg13_3068),XtNumber(seg14_3068),
    XtNumber(seg15_3068),XtNumber(seg16_3068),XtNumber(seg17_3068),
};
static XPoint seg0_3069[] = {
    {8,-10},{9,-10},{10,-12},{9,-6},{9,-8},{8,-10},{7,-11},{4,-12},
    {0,-12},{-3,-11},{-5,-9},{-5,-6},{-4,-4},{-2,-2},{4,1},
    {5,3},{5,6},{4,8},
};
static XPoint seg1_3069[] = {
    {-4,-6},{-3,-4},{4,0},{5,2},
};
static XPoint seg2_3069[] = {
    {-3,-11},{-4,-9},{-4,-7},{-3,-5},{3,-2},{5,0},{6,2},{6,5},
    {5,7},{4,8},{1,9},{-3,9},{-6,8},{-7,7},{-8,5},{-8,3},{-9,9},
    {-8,7},{-7,7},
};
static XPoint *char3069[] = {
    seg0_3069,seg1_3069,seg2_3069,
    NULL,
};
static int char_p3069[] = {
    XtNumber(seg0_3069),XtNumber(seg1_3069),XtNumber(seg2_3069),
};
static XPoint seg0_3070[] = {
    {2,-12},{-4,9},
};
static XPoint seg1_3070[] = {
    {3,-12},{-3,9},
};
```

continues

Listing C.4 Continued

```
static XPoint seg2_3070[] = {
    {4,-12},{-2,9},
};
static XPoint seg3_3070[] = {
    {-5,-12},{-7,-6},
};
static XPoint seg4_3070[] = {
    {11,-12},{10,-6},
};
static XPoint seg5_3070[] = {
    {-5,-12},{11,-12},
};
static XPoint seg6_3070[] = {
    {-7,9},{1,9},
};
static XPoint seg7_3070[] = {
    {-4,-12},{-7,-6},
};
static XPoint seg8_3070[] = {
    {-2,-12},{-6,-9},
};
static XPoint seg9_3070[] = {
    {0,-12},{-5,-11},
};
static XPoint seg10_3070[] = {
    {7,-12},{10,-11},
};
static XPoint seg11_3070[] = {
    {8,-12},{10,-10},
};
static XPoint seg12_3070[] = {
    {9,-12},{10,-9},
};
static XPoint seg13_3070[] = {
    {10,-12},{10,-6},
};
static XPoint seg14_3070[] = {
    {-3,8},{-6,9},
};
static XPoint seg15_3070[] = {
    {-3,7},{-5,9},
};
static XPoint seg16_3070[] = {
    {-2,7},{-1,9},
};
static XPoint seg17_3070[] = {
    {-3,8},{0,9},
};
static XPoint *char3070[] = {
    seg0_3070,seg1_3070,seg2_3070,seg3_3070,seg4_3070,
    seg5_3070,seg6_3070,seg7_3070,seg8_3070,seg9_3070,
    seg10_3070,seg11_3070,seg12_3070,seg13_3070,seg14_3070,
    seg15_3070,seg16_3070,seg17_3070,
    NULL,
};
static int char_p3070[] = {
    XtNumber(seg0_3070), XtNumber(seg1_3070), XtNumber(seg2_3070),
    XtNumber(seg3_3070), XtNumber(seg4_3070), XtNumber(seg5_3070),
    XtNumber(seg6_3070), XtNumber(seg7_3070), XtNumber(seg8_3070),
    XtNumber(seg9_3070), XtNumber(seg10_3070),XtNumber(seg11_3070),
    XtNumber(seg12_3070),XtNumber(seg13_3070),XtNumber(seg14_3070),
    XtNumber(seg15_3070),XtNumber(seg16_3070),XtNumber(seg17_3070),
};
static XPoint seg0_3071[] = {
    {-4,-12},{-7,-1},{-8,3},{-8,6},{-7,8},{-4,9},
    {0,9},{3,8},{5,6},{6,3},{10,-11},
};
static XPoint seg1_3071[] = {
    {-3,-12},{-6,-1},{-7,3},{-7,7},{-6,8},
};
static XPoint seg2_3071[] = {
    {-2,-12},{-5,-1},{-6,3},{-6,7},{-4,9},
};
static XPoint seg3_3071[] = {
    {-7,-12},{1,-12},
};
static XPoint seg4_3071[] = {
    {7,-12},{13,-12},
};
static XPoint seg5_3071[] = {
    {-6,-12},{-3,-11},
};
static XPoint seg6_3071[] = {
    {-5,-12},{-4,-10},
};
static XPoint seg7_3071[] = {
    {-1,-12},{-3,-10},
};
static XPoint seg8_3071[] = {
    {0,-12},{-3,-11},
};
static XPoint seg9_3071[] = {
    {8,-12},{10,-11},
};
static XPoint seg10_3071[] = {
    {12,-12},{10,-11},
};
static XPoint *char3071[] = {
    seg0_3071,seg1_3071,seg2_3071,seg3_3071,seg4_3071,seg5_3071,
    seg6_3071,seg7_3071,seg8_3071,seg9_3071,seg10_3071,
    NULL,
};
static int char_p3071[] = {
    XtNumber(seg0_3071),XtNumber(seg1_3071),XtNumber(seg2_3071),
```

continues

Listing C.4 Continued

```
    XtNumber(seg3_3071),XtNumber(seg4_3071),XtNumber(seg5_3071),
    XtNumber(seg6_3071),XtNumber(seg7_3071),XtNumber(seg8_3071),
    XtNumber(seg9_3071),XtNumber(seg10_3071),
};
static XPoint seg0_3072[] = {
    {-4,-12},{-4,-10},{-3,7},{-3,9},
};
static XPoint seg1_3072[] = {
    {-3,-11},{-2,6},
};
static XPoint seg2_3072[] = {
    {-2,-12},{-1,5},
};
static XPoint seg3_3072[] = {
    {9,-11},{-3,9},
};
static XPoint seg4_3072[] = {
    {-6,-12},{1,-12},
};
static XPoint seg5_3072[] = {
    {6,-12},{12,-12},
};
static XPoint seg6_3072[] = {
    {-5,-12},{-4,-10},
};
static XPoint seg7_3072[] = {
    {-1,-12},{-2,-10},
};
static XPoint seg8_3072[] = {
    {0,-12},{-3,-11},
};
static XPoint seg9_3072[] = {
    {7,-12},{9,-11},
};
static XPoint seg10_3072[] = {
    {11,-12},{9,-11},
};
static XPoint *char3072[] = {
    seg0_3072,seg1_3072,seg2_3072,seg3_3072,seg4_3072,seg5_3072,
    seg6_3072,seg7_3072,seg8_3072,seg9_3072,seg10_3072,
    NULL,
};
static int char_p3072[] = {
    XtNumber(seg0_3072),XtNumber(seg1_3072),XtNumber(seg2_3072),
    XtNumber(seg3_3072),XtNumber(seg4_3072),XtNumber(seg5_3072),
    XtNumber(seg6_3072),XtNumber(seg7_3072),XtNumber(seg8_3072),
    XtNumber(seg9_3072),XtNumber(seg10_3072),
};
static XPoint seg0_3073[] = {
    {-5,-12},{-5,-10},{-7,7},{-7,9},
};
```

```
static XPoint seg1_3073[] = {
    {-4,-11},{-6,6},
};
static XPoint seg2_3073[] = {
    {-3,-12},{-5,5},
};
static XPoint seg3_3073[] = {
    {3,-12},{-5,5},{-7,9},
};
static XPoint seg4_3073[] = {
    {3,-12},{3,-10},{1,7},{1,9},
};
static XPoint seg5_3073[] = {
    {4,-11},{2,6},
};
static XPoint seg6_3073[] = {
    {5,-12},{3,5},
};
static XPoint seg7_3073[] = {
    {11,-11},{3,5},{1,9},
};
static XPoint seg8_3073[] = {
    {-8,-12},{0,-12},
};
static XPoint seg9_3073[] = {
    {3,-12},{5,-12},
};
static XPoint seg10_3073[] = {
    {8,-12},{14,-12},
};
static XPoint seg11_3073[] = {
    {-7,-12},{-4,-11},
};
static XPoint seg12_3073[] = {
    {-6,-12},{-5,-10},
};
static XPoint seg13_3073[] = {
    {-2,-12},{-4,-9},
};
static XPoint seg14_3073[] = {
    {-1,-12},{-4,-11},
};
static XPoint seg15_3073[] = {
    {9,-12},{11,-11},
};
static XPoint seg16_3073[] = {
    {13,-12},{11,-11},
};
static XPoint *char3073[] = {
    seg0_3073,seg1_3073,seg2_3073,seg3_3073,seg4_3073,
    seg5_3073,seg6_3073,seg7_3073,seg8_3073,seg9_3073,
    seg10_3073,seg11_3073,seg12_3073,seg13_3073,seg14_3073,
```

continues

Listing C.4 Continued

```
    seg15_3073,seg16_3073,
    NULL,
};
static int char_p3073[] = {
    XtNumber(seg0_3073),XtNumber(seg1_3073),XtNumber(seg2_3073),
    XtNumber(seg3_3073),XtNumber(seg4_3073),XtNumber(seg5_3073),
    XtNumber(seg6_3073),XtNumber(seg7_3073),XtNumber(seg8_3073),
    XtNumber(seg9_3073),XtNumber(seg10_3073),XtNumber(seg11_3073),
    XtNumber(seg12_3073),XtNumber(seg13_3073),XtNumber(seg14_3073),
    XtNumber(seg15_3073),XtNumber(seg16_3073),
};
static XPoint seg0_3074[] = {
    {-4,-12},{2,9},
};
static XPoint seg1_3074[] = {
    {-3,-12},{3,9},
};
static XPoint seg2_3074[] = {
    {-2,-12},{4,9},
};
static XPoint seg3_3074[] = {
    {9,-11},{-9,8},
};
static XPoint seg4_3074[] = {
    {-6,-12},{1,-12},
};
static XPoint seg5_3074[] = {
    {6,-12},{12,-12},
};
static XPoint seg6_3074[] = {
    {-12,9},{-6,9},
};
static XPoint seg7_3074[] = {
    {-1,9},{6,9},
};
static XPoint seg8_3074[] = {
    {-5,-12},{-3,-10},
};
static XPoint seg9_3074[] = {
    {-1,-12},{-2,-10},
};
static XPoint seg10_3074[] = {
    {0,-12},{-2,-11},
};
static XPoint seg11_3074[] = {
    {7,-12},{9,-11},
};
static XPoint seg12_3074[] = {
    {11,-12},{9,-11},
};
```

```
static XPoint seg13_3074[] = {
    {-9,8},{-11,9},
};
static XPoint seg14_3074[] = {
    {-9,8},{-7,9},
};
static XPoint seg15_3074[] = {
    {2,8},{0,9},
};
static XPoint seg16_3074[] = {
    {2,7},{1,9},
};
static XPoint seg17_3074[] = {
    {3,7},{5,9},
};
static XPoint *char3074[] = {
    seg0_3074,seg1_3074,seg2_3074,seg3_3074,seg4_3074,
    seg5_3074,seg6_3074,seg7_3074,seg8_3074,seg9_3074,
    seg10_3074,seg11_3074,seg12_3074,seg13_3074,seg14_3074,
    seg15_3074,seg16_3074,seg17_3074,
    NULL,
};
static int char_p3074[] = {
    XtNumber(seg0_3074),XtNumber(seg1_3074),XtNumber(seg2_3074),
    XtNumber(seg3_3074),XtNumber(seg4_3074),XtNumber(seg5_3074),
    XtNumber(seg6_3074),XtNumber(seg7_3074),XtNumber(seg8_3074),
    XtNumber(seg9_3074),XtNumber(seg10_3074),XtNumber(seg11_3074),
    XtNumber(seg12_3074),XtNumber(seg13_3074),XtNumber(seg14_3074),
    XtNumber(seg15_3074),XtNumber(seg16_3074),XtNumber(seg17_3074),
};
static XPoint seg0_3075[] = {
    {-5,-12},{-1,-2},{-4,9},
};
static XPoint seg1_3075[] = {
    {-4,-12},{0,-2},{-3,9},
};
static XPoint seg2_3075[] = {
    {-3,-12},{1,-2},{-2,9},
};
static XPoint seg3_3075[] = {
    {10,-11},{1,-2},
};
static XPoint seg4_3075[] = {
    {-7,-12},{0,-12},
};
static XPoint seg5_3075[] = {
    {7,-12},{13,-12},
};
static XPoint seg6_3075[] = {
    {-7,9},{1,9},
};
static XPoint seg7_3075[] = {
    {-6,-12},{-4,-11},
};
```

continues

Listing C.4 Continued

```
static XPoint seg8_3075[] = {
    {-2,-12},{-3,-10},
};
static XPoint seg9_3075[] = {
    {-1,-12},{-4,-11},
};
static XPoint seg10_3075[] = {
    {8,-12},{10,-11},
};
static XPoint seg11_3075[] = {
    {12,-12},{10,-11},
};
static XPoint seg12_3075[] = {
    {-3,8},{-6,9},
};
static XPoint seg13_3075[] = {
    {-3,7},{-5,9},
};
static XPoint seg14_3075[] = {
    {-2,7},{-1,9},
};
static XPoint seg15_3075[] = {
    {-3,8},{0,9},
};
static XPoint *char3075[] = {
    seg0_3075,seg1_3075,seg2_3075,seg3_3075,seg4_3075,
    seg5_3075,seg6_3075,seg7_3075,seg8_3075,seg9_3075,
    seg10_3075,seg11_3075,seg12_3075,seg13_3075,seg14_3075,
    seg15_3075,
    NULL,
};
static int char_p3075[] = {
    XtNumber(seg0_3075),XtNumber(seg1_3075),XtNumber(seg2_3075),
    XtNumber(seg3_3075),XtNumber(seg4_3075),XtNumber(seg5_3075),
    XtNumber(seg6_3075),XtNumber(seg7_3075),XtNumber(seg8_3075),
    XtNumber(seg9_3075),XtNumber(seg10_3075),XtNumber(seg11_3075),
    XtNumber(seg12_3075),XtNumber(seg13_3075),XtNumber(seg14_3075),
    XtNumber(seg15_3075),
};
static XPoint seg0_3076[] = {
    {8,-12},{-10,9},
};
static XPoint seg1_3076[] = {
    {9,-12},{-9,9},
};
static XPoint seg2_3076[] = {
    {10,-12},{-8,9},
};
```

```
static XPoint seg3_3076[] = {
    {10,-12},{-4,-12},{-6,-6},
};
static XPoint seg4_3076[] = {
    {-10,9},{4,9},{6,3},
};
static XPoint seg5_3076[] = {
    {-3,-12},{-6,-6},
};
static XPoint seg6_3076[] = {
    {-2,-12},{-5,-9},
};
static XPoint seg7_3076[] = {
    {0,-12},{-4,-11},
};
static XPoint seg8_3076[] = {
    {0,9},{4,8},
};
static XPoint seg9_3076[] = {
    {2,9},{5,6},
};
static XPoint seg10_3076[] = {
    {3,9},{6,3},
};
static XPoint *char3076[] = {
    seg0_3076,seg1_3076,seg2_3076,seg3_3076,seg4_3076,seg5_3076,
    seg6_3076,seg7_3076,seg8_3076,seg9_3076,seg10_3076,
    NULL,
};
static int char_p3076[] = {
    XtNumber(seg0_3076),XtNumber(seg1_3076),XtNumber(seg2_3076),
    XtNumber(seg3_3076),XtNumber(seg4_3076),XtNumber(seg5_3076),
    XtNumber(seg6_3076),XtNumber(seg7_3076),XtNumber(seg8_3076),
    XtNumber(seg9_3076),XtNumber(seg10_3076),
};
static XPoint seg0_3101[] = {
    {-4,-2},{-4,-3},{-3,-3},{-3,-1},{-5,-1},{-5,-3},
    {-4,-4},{-2,-5},{2,-5},{4,-4},{5,-3},{6,-1},{6,6},{7,8},{8,9},
};
static XPoint seg1_3101[] = {
    {4,-3},{5,-1},{5,6},{6,8},
};
static XPoint seg2_3101[] = {
    {2,-5},{3,-4},{4,-2},{4,6},{5,8},{8,9},{9,9},
};
static XPoint seg3_3101[] = {
    {4,0},{3,1},{-2,2},{-5,3},{-6,5},{-6,6},{-5,8},{-2,9},
    {1,9},{3,8},{4,6},
};
static XPoint seg4_3101[] = {
    {-4,3},{-5,5},{-5,6},{-4,8},
};
static XPoint seg5_3101[] = {
    {3,1},{-1,2},{-3,3},{-4,5},{-4,6},{-3,8},{-2,9},
};
```

continues

Listing C.4 Continued

```
static XPoint *char3101[] = {
    seg0_3101,seg1_3101,seg2_3101,seg3_3101,seg4_3101,seg5_3101,
    NULL,
};
static int char_p3101[] = {
    XtNumber(seg0_3101),XtNumber(seg1_3101),XtNumber(seg2_3101),
    XtNumber(seg3_3101),XtNumber(seg4_3101),XtNumber(seg5_3101),
};
static XPoint seg0_3102[] = {
    {-6,-12},
    {-6,9},{-5,8},{-3,8},
};
static XPoint seg1_3102[] = {
    {-5,-11},{-5,7},
};
static XPoint seg2_3102[] = {
    {-9,-12},{-4,-12},{-4,8},
};
static XPoint seg3_3102[] = {
    {-4,-2},{-3,-4},{-1,-5},{1,-5},{4,-4},{6,-2},{7,1},{7,3},
    {6,6},{4,8},{1,9},{-1,9},{-3,8},{-4,6},
};
static XPoint seg4_3102[] = {
    {5,-2},{6,0},{6,4},{5,6},
};
static XPoint seg5_3102[] = {
    {1,-5},{3,-4},{4,-3},{5,0},{5,4},{4,7},{3,8},{1,9},
};
static XPoint seg6_3102[] = {
    {-8,-12},{-6,-11},
};
static XPoint seg7_3102[] = {
    {-7,-12},{-6,-10},
};
static XPoint *char3102[] = {
    seg0_3102,seg1_3102,seg2_3102,seg3_3102,seg4_3102,seg5_3102,
    seg6_3102,seg7_3102,
    NULL,
};
static int char_p3102[] = {
    XtNumber(seg0_3102),XtNumber(seg1_3102),XtNumber(seg2_3102),
    XtNumber(seg3_3102),XtNumber(seg4_3102),XtNumber(seg5_3102),
    XtNumber(seg6_3102),XtNumber(seg7_3102),
};
static XPoint seg0_3103[] = {
    {5,-1},{5,-2},{4,-2},{4,0},{6,0},{6,-2},
    {4,-4},{2,-5},{-1,-5},{-4,-4},{-6,-2},{-7,1},{-7,3},{-6,6},
    {-4,8},{-1,9},{1,9},{4,8},{6,6},
};
```

```
static XPoint seg1_3103[] = {
    {-5,-2},{-6,0},{-6,4},{-5,6},
};
static XPoint seg2_3103[] = {
    {-1,-5},{-3,-4},{-4,-3},{-5,0},{-5,4},{-4,7},{-3,8},{-1,9},
};
static XPoint *char3103[] = {
    seg0_3103,seg1_3103,seg2_3103,
    NULL,
};
static int char_p3103[] = {
    XtNumber(seg0_3103),XtNumber(seg1_3103),XtNumber(seg2_3103),
};
static XPoint seg0_3104[] = {
    {4,-12},{4,9},{9,9},
};
static XPoint seg1_3104[] = {
    {5,-11},{5,8},
};
static XPoint seg2_3104[] = {
    {1,-12},{6,-12},{6,9},
};
static XPoint seg3_3104[] = {
    {4,-2},{3,-4},{1,-5},{-1,-5},{-4,-4},{-6,-2},{-7,1},{-7,3},
    {-6,6},{-4,8},{-1,9},{1,9},{3,8},{4,6},
};
static XPoint seg4_3104[] = {
    {-5,-2},{-6,0},{-6,4},{-5,6},
};
static XPoint seg5_3104[] = {
    {-1,-5},{-3,-4},{-4,-3},{-5,0},{-5,4},{-4,7},{-3,8},{-1,9},
};
static XPoint seg6_3104[] = {
    {2,-12},{4,-11},
};
static XPoint seg7_3104[] = {
    {3,-12},{4,-10},
};
static XPoint seg8_3104[] = {
    {6,7},{7,9},
};
static XPoint seg9_3104[] = {
    {6,8},{8,9},
};
static XPoint *char3104[] = {
    seg0_3104,seg1_3104,seg2_3104,seg3_3104,seg4_3104,seg5_3104,
    seg6_3104,seg7_3104,seg8_3104,seg9_3104,
    NULL,
};
static int char_p3104[] = {
    XtNumber(seg0_3104),XtNumber(seg1_3104),XtNumber(seg2_3104),
    XtNumber(seg3_3104),XtNumber(seg4_3104),XtNumber(seg5_3104),
    XtNumber(seg6_3104),XtNumber(seg7_3104),XtNumber(seg8_3104),
```

continues

Listing C.4 Continued

```
    XtNumber(seg9_3104),
};
static XPoint seg0_3105[] = {
    {-5,1},{6,1},{6,-1},{5,-3},{4,-4},{1,-5},
    {-1,-5},{-4,-4},{-6,-2},{-7,1},{-7,3},{-6,6},{-4,8},{-1,9},
    {1,9},{4,8},{6,6},
};
static XPoint seg1_3105[] = {
    {5,0},{5,-1},{4,-3},
};
static XPoint seg2_3105[] = {
    {-5,-2},{-6,0},{-6,4},{-5,6},
};
static XPoint seg3_3105[] = {
    {4,1},{4,-2},{3,-4},{1,-5},
};
static XPoint seg4_3105[] = {
    {-1,-5},{-3,-4},{-4,-3},{-5,0},{-5,4},{-4,7},{-3,8},{-1,9},
};
static XPoint *char3105[] = {
    seg0_3105,seg1_3105,seg2_3105,seg3_3105,seg4_3105,
    NULL,
};
static int char_p3105[] = {
    XtNumber(seg0_3105),XtNumber(seg1_3105),XtNumber(seg2_3105),
    XtNumber(seg3_3105),XtNumber(seg4_3105),
};
static XPoint seg0_3106[] = {
    {5,-10},{5,-11},{4,-11},{4,-9},{6,-9},{6,-11},{5,-12},
    {2,-12},{0,-11},{-1,-10},{-2,-7},{-2,9},
};
static XPoint seg1_3106[] = {
    {0,-10},{-1,-7},{-1,8},
};
static XPoint seg2_3106[] = {
    {2,-12},{1,-11},{0,-9},{0,9},
};
static XPoint seg3_3106[] = {
    {-5,-5},{4,-5},
};
static XPoint seg4_3106[] = {
    {-5,9},{3,9},
};
static XPoint seg5_3106[] = {
    {-2,8},{-4,9},
};
static XPoint seg6_3106[] = {
    {-2,7},{-3,9},
};
static XPoint seg7_3106[] = {
    {0,7},{1,9},
};
```

```
static XPoint seg8_3106[] = {
    {0,8},{2,9},
};
static XPoint *char3106[] = {
    seg0_3106,seg1_3106,seg2_3106,seg3_3106,seg4_3106,seg5_3106,
    seg6_3106,seg7_3106,seg8_3106,
    NULL,
};
static int char_p3106[] = {
    XtNumber(seg0_3106),XtNumber(seg1_3106),XtNumber(seg2_3106),
    XtNumber(seg3_3106),XtNumber(seg4_3106),XtNumber(seg5_3106),
    XtNumber(seg6_3106),XtNumber(seg7_3106),XtNumber(seg8_3106),
};
static XPoint seg0_3107[] = {
    {6,-4},{7,-3},{8,-4},{7,-5},{6,-5},{4,-4},
    {3,-3},
};
static XPoint seg1_3107[] = {
    {-1,-5},{-3,-4},{-4,-3},{-5,-1},{-5,1},{-4,3},{-3,4},{-1,5},
    {1,5},{3,4},{4,3},{5,1},{5,-1},{4,-3},{3,-4},{1,-5},{-1,-5},
};
static XPoint seg2_3107[] = {
    {-3,-3},{-4,-1},{-4,1},{-3,3},
};
static XPoint seg3_3107[] = {
    {3,3},{4,1},{4,-1},{3,-3},
};
static XPoint seg4_3107[] = {
    {-1,-5},{-2,-4},{-3,-2},{-3,2},{-2,4},{-1,5},
};
static XPoint seg5_3107[] = {
    {1,5},{2,4},{3,2},{3,-2},{2,-4},{1,-5},
};
static XPoint seg6_3107[] = {
    {-4,3},{-5,4},{-6,6},{-6,7},{-5,9},{-4,10},{-1,11},{3,11},
    {6,12},{7,13},
};
static XPoint seg7_3107[] = {
    {-4,9},{-1,10},{3,10},{6,11},
};
static XPoint seg8_3107[] = {
    {-6,7},{-5,8},{-2,9},{3,9},{6,10},{7,12},{7,13},{6,15},
    {3,16},{-3,16},{-6,15},{-7,13},{-7,12},{-6,10},{-3,9},
};
static XPoint seg9_3107[] = {
    {-3,16},{-5,15},{-6,13},{-6,12},{-5,10},{-3,9},
};
static XPoint *char3107[] = {
    seg0_3107,seg1_3107,seg2_3107,seg3_3107,seg4_3107,seg5_3107,
    seg6_3107,seg7_3107,seg8_3107,seg9_3107,
    NULL,
};
```

continues

Listing C.4 Continued

```
static int char_p3107[] = {
    XtNumber(seg0_3107),XtNumber(seg1_3107),XtNumber(seg2_3107),
    XtNumber(seg3_3107),XtNumber(seg4_3107),XtNumber(seg5_3107),
    XtNumber(seg6_3107),XtNumber(seg7_3107),XtNumber(seg8_3107),
    XtNumber(seg9_3107),
};
static XPoint seg0_3108[] = {
    {-6,-12},{-6,9},
};
static XPoint seg1_3108[] = {
    {-5,-11},{-5,8},
};
static XPoint seg2_3108[] = {
    {-9,-12},{-4,-12},{-4,9},
};
static XPoint seg3_3108[] = {
    {-4,-1},{-3,-3},{-2,-4},{0,-5},{3,-5},{5,-4},{6,-3},{7,0},
    {7,9},
};
static XPoint seg4_3108[] = {
    {5,-3},{6,0},{6,8},
};
static XPoint seg5_3108[] = {
    {3,-5},{4,-4},{5,-1},{5,9},
};
static XPoint seg6_3108[] = {
    {-9,9},{-1,9},
};
static XPoint seg7_3108[] = {
    {2,9},{10,9},
};
static XPoint seg8_3108[] = {
    {-8,-12},{-6,-11},
};
static XPoint seg9_3108[] = {
    {-7,-12},{-6,-10},
};
static XPoint seg10_3108[] = {
    {-6,8},{-8,9},
};
static XPoint seg11_3108[] = {
    {-6,7},{-7,9},
};
static XPoint seg12_3108[] = {
    {-4,7},{-3,9},
};
static XPoint seg13_3108[] = {
    {-4,8},{-2,9},
};
```

```
static XPoint seg14_3108[] = {
    {5,8},{3,9},
};
static XPoint seg15_3108[] = {
    {5,7},{4,9},
};
static XPoint seg16_3108[] = {
    {7,7},{8,9},
};
static XPoint seg17_3108[] = {
    {7,8},{9,9},
};
static XPoint *char3108[] = {
    seg0_3108,seg1_3108,seg2_3108,seg3_3108,seg4_3108,
    seg5_3108,seg6_3108,seg7_3108,seg8_3108,seg9_3108,
    seg10_3108,seg11_3108,seg12_3108,seg13_3108,seg14_3108,
    seg15_3108,seg16_3108,seg17_3108,
    NULL,
};
static int char_p3108[] = {
    XtNumber(seg0_3108),XtNumber(seg1_3108),XtNumber(seg2_3108),
    XtNumber(seg3_3108),XtNumber(seg4_3108),XtNumber(seg5_3108),
    XtNumber(seg6_3108),XtNumber(seg7_3108),XtNumber(seg8_3108),
    XtNumber(seg9_3108),XtNumber(seg10_3108),XtNumber(seg11_3108),
    XtNumber(seg12_3108),XtNumber(seg13_3108),XtNumber(seg14_3108),
    XtNumber(seg15_3108),XtNumber(seg16_3108),XtNumber(seg17_3108),
};
static XPoint seg0_3109[] = {
    {-1,-12},{-1,-10},{1,-10},{1,-12},{-1,-12},
};
static XPoint seg1_3109[] = {
    {0,-12},{0,-10},
};
static XPoint seg2_3109[] = {
    {-1,-11},{1,-11},
};
static XPoint seg3_3109[] = {
    {-1,-5},{-1,9},
};
static XPoint seg4_3109[] = {
    {0,-4},{0,8},
};
static XPoint seg5_3109[] = {
    {-4,-5},{1,-5},{1,9},
};
static XPoint seg6_3109[] = {
    {-4,9},{4,9},
};
static XPoint seg7_3109[] = {
    {-3,-5},{-1,-4},
};
static XPoint seg8_3109[] = {
    {-2,-5},{-1,-3},
};
```

continues

Listing C.4 Continued

```
static XPoint seg9_3109[] = {
    {-1,8},{-3,9},
};
static XPoint seg10_3109[] = {
    {-1,7},{-2,9},
};
static XPoint seg11_3109[] = {
    {1,7},{2,9},
};
static XPoint seg12_3109[] = {
    {1,8},{3,9},
};
static XPoint *char3109[] = {
    seg0_3109,seg1_3109,seg2_3109,seg3_3109,seg4_3109,
    seg5_3109,seg6_3109,seg7_3109,seg8_3109,seg9_3109,
    seg10_3109,seg11_3109,seg12_3109,
    NULL,
};
static int char_p3109[] = {
    XtNumber(seg0_3109),XtNumber(seg1_3109),XtNumber(seg2_3109),
    XtNumber(seg3_3109),XtNumber(seg4_3109),XtNumber(seg5_3109),
    XtNumber(seg6_3109),XtNumber(seg7_3109),XtNumber(seg8_3109),
    XtNumber(seg9_3109),XtNumber(seg10_3109),XtNumber(seg11_3109),
    XtNumber(seg12_3109),
};
static XPoint seg0_3110[] = {
    {0,-12},{0,-10},{2,-10},{2,-12},{0,-12},
};
static XPoint seg1_3110[] = {
    {1,-12},{1,-10},
};
static XPoint seg2_3110[] = {
    {0,-11},{2,-11},
};
static XPoint seg3_3110[] = {
    {0,-5},{0,12},{-1,15},{-2,16},
};
static XPoint seg4_3110[] = {
    {1,-4},{1,11},{0,14},
};
static XPoint seg5_3110[] = {
    {-3,-5},{2,-5},{2,11},{1,14},{0,15},{-2,16},{-5,16},{-6,15},
    {-6,13},{-4,13},{-4,15},{-5,15},{-5,14},
};
static XPoint seg6_3110[] = {
    {-2,-5},{0,-4},
};
static XPoint seg7_3110[] = {
    {-1,-5},{0,-3},
};
```

```
static XPoint *char3110[] = {
    seg0_3110,seg1_3110,seg2_3110,seg3_3110,seg4_3110,seg5_3110,
    seg6_3110,seg7_3110,
    NULL,
};
static int char_p3110[] = {
    XtNumber(seg0_3110),XtNumber(seg1_3110),XtNumber(seg2_3110),
    XtNumber(seg3_3110),XtNumber(seg4_3110),XtNumber(seg5_3110),
    XtNumber(seg6_3110),XtNumber(seg7_3110),
};
static XPoint seg0_3111[] = {
    {-6,-12},{-6,9},
};
static XPoint seg1_3111[] = {
    {-5,-11},{-5,8},
};
static XPoint seg2_3111[] = {
    {-9,-12},{-4,-12},{-4,9},
};
static XPoint seg3_3111[] = {
    {5,-4},{-4,5},
};
static XPoint seg4_3111[] = {
    {0,1},{7,9},
};
static XPoint seg5_3111[] = {
    {0,2},{6,9},
};
static XPoint seg6_3111[] = {
    {-1,2},{5,9},
};
static XPoint seg7_3111[] = {
    {2,-5},{9,-5},
};
static XPoint seg8_3111[] = {
    {-9,9},{-1,9},
};
static XPoint seg9_3111[] = {
    {2,9},{9,9},
};
static XPoint seg10_3111[] = {
    {-8,-12},{-6,-11},
};
static XPoint seg11_3111[] = {
    {-7,-12},{-6,-10},
};
static XPoint seg12_3111[] = {
    {3,-5},{5,-4},
};
static XPoint seg13_3111[] = {
    {8,-5},{5,-4},
};
```

continues

Listing C.4 Continued

```
static XPoint seg14_3111[] = {
    {-6,8},{-8,9},
};
static XPoint seg15_3111[] = {
    {-6,7},{-7,9},
};
static XPoint seg16_3111[] = {
    {-4,7},{-3,9},
};
static XPoint seg17_3111[] = {
    {-4,8},{-2,9},
};
static XPoint seg18_3111[] = {
    {5,7},{3,9},
};
static XPoint seg19_3111[] = {
    {4,7},{8,9},
};
static XPoint *char3111[] = {
    seg0_3111,seg1_3111,seg2_3111,seg3_3111,seg4_3111,
    seg5_3111,seg6_3111,seg7_3111,seg8_3111,seg9_3111,
    seg10_3111,seg11_3111,seg12_3111,seg13_3111,seg14_3111,
    seg15_3111,seg16_3111,seg17_3111,seg18_3111,seg19_3111,
    NULL,
};
static int char_p3111[] = {
    XtNumber(seg0_3111),XtNumber(seg1_3111),XtNumber(seg2_3111),
    XtNumber(seg3_3111),XtNumber(seg4_3111),XtNumber(seg5_3111),
    XtNumber(seg6_3111),XtNumber(seg7_3111),XtNumber(seg8_3111),
    XtNumber(seg9_3111),XtNumber(seg10_3111),XtNumber(seg11_3111),
    XtNumber(seg12_3111),XtNumber(seg13_3111),XtNumber(seg14_3111),
    XtNumber(seg15_3111),XtNumber(seg16_3111),XtNumber(seg17_3111),
    XtNumber(seg18_3111),XtNumber(seg19_3111),
};
static XPoint seg0_3112[] = {
    {-1,-12},{-1,9},
};
static XPoint seg1_3112[] = {
    {0,-11},{0,8},
};
static XPoint seg2_3112[] = {
    {-4,-12},{1,-12},{1,9},
};
static XPoint seg3_3112[] = {
    {-4,9},{4,9},
};
static XPoint seg4_3112[] = {
    {-3,-12},{-1,-11},
};
static XPoint seg5_3112[] = {
    {-2,-12},{-1,-10},
};
```

```
static XPoint seg6_3112[] = {
    {-1,8},{-3,9},
};
static XPoint seg7_3112[] = {
    {-1,7},{-2,9},
};
static XPoint seg8_3112[] = {
    {1,7},{2,9},
};
static XPoint seg9_3112[] = {
    {1,8},{3,9},
};
static XPoint *char3112[] = {
    seg0_3112,seg1_3112,seg2_3112,seg3_3112,seg4_3112,seg5_3112,
    seg6_3112,seg7_3112,seg8_3112,seg9_3112,
    NULL,
};
static int char_p3112[] = {
    XtNumber(seg0_3112),XtNumber(seg1_3112),XtNumber(seg2_3112),
    XtNumber(seg3_3112),XtNumber(seg4_3112),XtNumber(seg5_3112),
    XtNumber(seg6_3112),XtNumber(seg7_3112),XtNumber(seg8_3112),
    XtNumber(seg9_3112),
};
static XPoint seg0_3113[] = {
    {-12,-5},{-12,9},
};
static XPoint seg1_3113[] = {
    {-11,-4},{-11,8},
};
static XPoint seg2_3113[] = {
    {-15,-5},{-10,-5},{-10,9},
};
static XPoint seg3_3113[] = {
    {-10,-1},{-9,-3},{-8,-4},{-6,-5},{-3,-5},{-1,-4},{0,-3},
    {1,0},{1,9},
};
static XPoint seg4_3113[] = {
    {-1,-3},{0,0},{0,8},
};
static XPoint seg5_3113[] = {
    {-3,-5},{-2,-4},{-1,-1},{-1,9},
};
static XPoint seg6_3113[] = {
    {1,-1},{2,-3},{3,-4},{5,-5},{8,-5},{10,-4},{11,-3},{12,0},
    {12,9},
};
static XPoint seg7_3113[] = {
    {10,-3},{11,0},{11,8},
};
static XPoint seg8_3113[] = {
    {8,-5},{9,-4},{10,-1},{10,9},
};
```

continues

Listing C.4 Continued

```
static XPoint seg9_3113[] = {
    {-15,9},{-7,9},
};
static XPoint seg10_3113[] = {
    {-4,9},{4,9},
};
static XPoint seg11_3113[] = {
    {7,9},{15,9},
};
static XPoint seg12_3113[] = {
    {-14,-5},{-12,-4},
};
static XPoint seg13_3113[] = {
    {-13,-5},{-12,-3},
};
static XPoint seg14_3113[] = {
    {-12,8},{-14,9},
};
static XPoint seg15_3113[] = {
    {-12,7},{-13,9},
};
static XPoint seg16_3113[] = {
    {-10,7},{-9,9},
};
static XPoint seg17_3113[] = {
    {-10,8},{-8,9},
};
static XPoint seg18_3113[] = {
    {-1,8},{-3,9},
};
static XPoint seg19_3113[] = {
    {-1,7},{-2,9},
};
static XPoint seg20_3113[] = {
    {1,7},{2,9},
};
static XPoint seg21_3113[] = {
    {1,8},{3,9},
};
static XPoint seg22_3113[] = {
    {10,8},{8,9},
};
static XPoint seg23_3113[] = {
    {10,7},{9,9},
};
static XPoint seg24_3113[] = {
    {12,7},{13,9},
};
static XPoint seg25_3113[] = {
    {12,8},{14,9},
};
```

```
static XPoint *char3113[] = {
    seg0_3113,seg1_3113,seg2_3113,seg3_3113,seg4_3113,
    seg5_3113,seg6_3113,seg7_3113,seg8_3113,seg9_3113,
    seg10_3113,seg11_3113,seg12_3113,seg13_3113,seg14_3113,
    seg15_3113,seg16_3113,seg17_3113,seg18_3113,seg19_3113,
    seg20_3113,seg21_3113,seg22_3113,seg23_3113,seg24_3113,
    seg25_3113,
    NULL,
};
static int char_p3113[] = {
    XtNumber(seg0_3113),XtNumber(seg1_3113),XtNumber(seg2_3113),
    XtNumber(seg3_3113),XtNumber(seg4_3113),XtNumber(seg5_3113),
    XtNumber(seg6_3113),XtNumber(seg7_3113),XtNumber(seg8_3113),
    XtNumber(seg9_3113),XtNumber(seg10_3113),XtNumber(seg11_3113),
    XtNumber(seg12_3113),XtNumber(seg13_3113),XtNumber(seg14_3113),
    XtNumber(seg15_3113),XtNumber(seg16_3113),XtNumber(seg17_3113),
    XtNumber(seg18_3113),XtNumber(seg19_3113),XtNumber(seg20_3113),
    XtNumber(seg21_3113),XtNumber(seg22_3113),XtNumber(seg23_3113),
    XtNumber(seg24_3113),XtNumber(seg25_3113),
};
static XPoint seg0_3114[] = {
    {-6,-5},{-6,9},
};
static XPoint seg1_3114[] = {
    {-5,-4},{-5,8},
};
static XPoint seg2_3114[] = {
    {-9,-5},{-4,-5},{-4,9},
};
static XPoint seg3_3114[] = {
    {-4,-1},{-3,-3},{-2,-4},{0,-5},{3,-5},{5,-4},{6,-3},{7,0},
    {7,9},
};
static XPoint seg4_3114[] = {
    {5,-3},{6,0},{6,8},
};
static XPoint seg5_3114[] = {
    {3,-5},{4,-4},{5,-1},{5,9},
};
static XPoint seg6_3114[] = {
    {-9,9},{-1,9},
};
static XPoint seg7_3114[] = {
    {2,9},{10,9},
};
static XPoint seg8_3114[] = {
    {-8,-5},{-6,-4},
};
static XPoint seg9_3114[] = {
    {-7,-5},{-6,-3},
};
static XPoint seg10_3114[] = {
    {-6,8},{-8,9},
};
```

continues

Listing C.4 Continued

```
static XPoint seg11_3114[] = {
    {-6,7},{-7,9},
};
static XPoint seg12_3114[] = {
    {-4,7},{-3,9},
};
static XPoint seg13_3114[] = {
    {-4,8},{-2,9},
};
static XPoint seg14_3114[] = {
    {5,8},{3,9},
};
static XPoint seg15_3114[] = {
    {5,7},{4,9},
};
static XPoint seg16_3114[] = {
    {7,7},{8,9},
};
static XPoint seg17_3114[] = {
    {7,8},{9,9},
};
static XPoint *char3114[] = {
    seg0_3114,seg1_3114,seg2_3114,seg3_3114,seg4_3114,
    seg5_3114,seg6_3114,seg7_3114,seg8_3114,seg9_3114,
    seg10_3114,seg11_3114,seg12_3114,seg13_3114,
    seg14_3114,seg15_3114,seg16_3114,seg17_3114,
    NULL,
};
static int char_p3114[] = {
    XtNumber(seg0_3114),XtNumber(seg1_3114),XtNumber(seg2_3114),
    XtNumber(seg3_3114),XtNumber(seg4_3114),XtNumber(seg5_3114),
    XtNumber(seg6_3114),XtNumber(seg7_3114),XtNumber(seg8_3114),
    XtNumber(seg9_3114),XtNumber(seg10_3114),XtNumber(seg11_3114),
    XtNumber(seg12_3114),XtNumber(seg13_3114),XtNumber(seg14_3114),
    XtNumber(seg15_3114),XtNumber(seg16_3114),XtNumber(seg17_3114),
};
static XPoint seg0_3115[] = {
    {-1,-5},{-4,-4},{-6,-2},{-7,1},{-7,3},{-6,6},
    {-4,8},{-1,9},{1,9},{4,8},{6,6},{7,3},{7,1},{6,-2},{4,-4},
    {1,-5},{-1,-5},
};
static XPoint seg1_3115[] = {
    {-5,-2},{-6,0},{-6,4},{-5,6},
};
static XPoint seg2_3115[] = {
    {5,6},{6,4},{6,0},{5,-2},
};
static XPoint seg3_3115[] = {
    {-1,-5},{-3,-4},{-4,-3},{-5,0},{-5,4},{-4,7},{-3,8},{-1,9},
};
```

```
static XPoint seg4_3115[] = {
    {1,9},{3,8},{4,7},{5,4},{5,0},{4,-3},{3,-4},{1,-5},
};
static XPoint *char3115[] = {
    seg0_3115,seg1_3115,seg2_3115,seg3_3115,seg4_3115,
    NULL,
};
static int char_p3115[] = {
    XtNumber(seg0_3115),XtNumber(seg1_3115),XtNumber(seg2_3115),
    XtNumber(seg3_3115),XtNumber(seg4_3115),
};
static XPoint seg0_3116[] = {
    {-6,-5},{-6,16},
};
static XPoint seg1_3116[] = {
    {-5,-4},{-5,15},
};
static XPoint seg2_3116[] = {
    {-9,-5},{-4,-5},{-4,16},
};
static XPoint seg3_3116[] = {
    {-4,-2},{-3,-4},{-1,-5},{1,-5},{4,-4},{6,-2},{7,1},{7,3},
    {6,6},{4,8},{1,9},{-1,9},{-3,8},{-4,6},
};
static XPoint seg4_3116[] = {
    {5,-2},{6,0},{6,4},{5,6},
};
static XPoint seg5_3116[] = {
    {1,-5},{3,-4},{4,-3},{5,0},{5,4},{4,7},{3,8},{1,9},
};
static XPoint seg6_3116[] = {
    {-9,16},{-1,16},
};
static XPoint seg7_3116[] = {
    {-8,-5},{-6,-4},
};
static XPoint seg8_3116[] = {
    {-7,-5},{-6,-3},
};
static XPoint seg9_3116[] = {
    {-6,15},{-8,16},
};
static XPoint seg10_3116[] = {
    {-6,14},{-7,16},
};
static XPoint seg11_3116[] = {
    {-4,14},{-3,16},
};
static XPoint seg12_3116[] = {
    {-4,15},{-2,16},
};
```

continues

Listing C.4 Continued

```
static XPoint *char3116[] = {
    seg0_3116,seg1_3116,seg2_3116,seg3_3116,seg4_3116,
    seg5_3116,seg6_3116,seg7_3116,seg8_3116,seg9_3116,
    seg10_3116,seg11_3116,
    seg12_3116,
    NULL,
};
static int char_p3116[] = {
    XtNumber(seg0_3116),XtNumber(seg1_3116),XtNumber(seg2_3116),
    XtNumber(seg3_3116),XtNumber(seg4_3116),XtNumber(seg5_3116),
    XtNumber(seg6_3116),XtNumber(seg7_3116),XtNumber(seg8_3116),
    XtNumber(seg9_3116),XtNumber(seg10_3116),XtNumber(seg11_3116),
    XtNumber(seg12_3116),
};
static XPoint seg0_3117[] = {
    {4,-4},{4,16},
};
static XPoint seg1_3117[] = {
    {5,-3},{5,15},
};
static XPoint seg2_3117[] = {
    {3,-4},{5,-4},{6,-5},{6,16},
};
static XPoint seg3_3117[] = {
    {4,-2},{3,-4},{1,-5},{-1,-5},{-4,-4},{-6,-2},{-7,1},{-7,3},
    {-6,6},{-4,8},{-1,9},{1,9},{3,8},{4,6},
};
static XPoint seg4_3117[] = {
    {-5,-2},{-6,0},{-6,4},{-5,6},
};
static XPoint seg5_3117[] = {
    {-1,-5},{-3,-4},{-4,-3},{-5,0},{-5,4},{-4,7},{-3,8},{-1,9},
};
static XPoint seg6_3117[] = {
    {1,16},{9,16},
};
static XPoint seg7_3117[] = {
    {4,15},{2,16},
};
static XPoint seg8_3117[] = {
    {4,14},{3,16},
};
static XPoint seg9_3117[] = {
    {6,14},{7,16},
};
static XPoint seg10_3117[] = {
    {6,15},{8,16},
};
static XPoint *char3117[] = {
    seg0_3117,seg1_3117,seg2_3117,seg3_3117,seg4_3117,seg5_3117,
    seg6_3117,seg7_3117,seg8_3117,seg9_3117,seg10_3117,
    NULL,
};
static int char_p3117[] = {
    XtNumber(seg0_3117),XtNumber(seg1_3117),XtNumber(seg2_3117),
    XtNumber(seg3_3117),XtNumber(seg4_3117),XtNumber(seg5_3117),
    XtNumber(seg6_3117),XtNumber(seg7_3117),XtNumber(seg8_3117),
    XtNumber(seg9_3117),XtNumber(seg10_3117),
};
static XPoint seg0_3118[] = {
    {-4,-5},{-4,9},
};
static XPoint seg1_3118[] = {
    {-3,-4},{-3,8},
};
static XPoint seg2_3118[] = {
    {-7,-5},{-2,-5},{-2,9},
};
static XPoint seg3_3118[] = {
    {5,-3},{5,-4},{4,-4},{4,-2},{6,-2},{6,-4},{5,-5},{3,-5},
    {1,-4},{-1,-2},{-2,1},
};
static XPoint seg4_3118[] = {
    {-7,9},{1,9},
};
static XPoint seg5_3118[] = {
    {-6,-5},{-4,-4},
};
static XPoint seg6_3118[] = {
    {-5,-5},{-4,-3},
};
static XPoint seg7_3118[] = {
    {-4,8},{-6,9},
};
static XPoint seg8_3118[] = {
    {-4,7},{-5,9},
};
static XPoint seg9_3118[] = {
    {-2,7},{-1,9},
};
static XPoint seg10_3118[] = {
    {-2,8},{0,9},
};
static XPoint *char3118[] = {
    seg0_3118,seg1_3118,seg2_3118,seg3_3118,seg4_3118,seg5_3118,
    seg6_3118,seg7_3118,seg8_3118,seg9_3118,seg10_3118,
    NULL,
};
static int char_p3118[] = {
    XtNumber(seg0_3118),XtNumber(seg1_3118),XtNumber(seg2_3118),
    XtNumber(seg3_3118),XtNumber(seg4_3118),XtNumber(seg5_3118),
    XtNumber(seg6_3118),XtNumber(seg7_3118),XtNumber(seg8_3118),
    XtNumber(seg9_3118),XtNumber(seg10_3118),
};
```

continues

Listing C.4 Continued

```
static XPoint seg0_3119[] = {
    {5,-3},{6,-5},{6,-1},{5,-3},{4,-4},{2,-5},{-2,-5},
    {-4,-4},{-5,-3},{-5,-1},{-4,1},{-2,2},{3,3},
    {5,4},{6,7},
};
static XPoint seg1_3119[] = {
    {-4,-4},{-5,-1},
};
static XPoint seg2_3119[] = {
    {-4,0},{-2,1},{3,2},{5,3},
};
static XPoint seg3_3119[] = {
    {6,4},{5,8},
};
static XPoint seg4_3119[] = {
    {-5,-3},{-4,-1},{-2,0},{3,1},{5,2},{6,4},{6,7},{5,8},
    {3,9},{-1,9},{-3,8},{-4,7},{-5,5},{-5,9},{-4,7},
};
static XPoint *char3119[] = {
    seg0_3119,seg1_3119,seg2_3119,seg3_3119,seg4_3119,
    NULL,
};
static int char_p3119[] = {
    XtNumber(seg0_3119),XtNumber(seg1_3119),XtNumber(seg2_3119),
    XtNumber(seg3_3119),XtNumber(seg4_3119),
};
static XPoint seg0_3120[] = {
    {-2,-10},{-2,4},
    {-1,7},{0,8},{2,9},{4,9},{6,8},{7,6},
};
static XPoint seg1_3120[] = {
    {-1,-10},{-1,5},{0,7},
};
static XPoint seg2_3120[] = {
    {-2,-10},{0,-12},{0,5},{1,8},{2,9},
};
static XPoint seg3_3120[] = {
    {-5,-5},{4,-5},
};
static XPoint *char3120[] = {
    seg0_3120,seg1_3120,seg2_3120,seg3_3120,
    NULL,
};
static int char_p3120[] = {
    XtNumber(seg0_3120),XtNumber(seg1_3120),XtNumber(seg2_3120),
    XtNumber(seg3_3120),
};
static XPoint seg0_3121[] = {
    {-6,-5},{-6,4},{-5,7},{-4,8},{-2,9},{1,9},
    {3,8},{4,7},{5,5},
};
```

```
static XPoint seg1_3121[] = {
    {-5,-4},{-5,5},{-4,7},
};
static XPoint seg2_3121[] = {
    {-9,-5},{-4,-5},{-4,5},{-3,8},{-2,9},
};
static XPoint seg3_3121[] = {
    {5,-5},{5,9},{10,9},
};
static XPoint seg4_3121[] = {
    {6,-4},{6,8},
};
static XPoint seg5_3121[] = {
    {2,-5},{7,-5},{7,9},
};
static XPoint seg6_3121[] = {
    {-8,-5},{-6,-4},
};
static XPoint seg7_3121[] = {
    {-7,-5},{-6,-3},
};
static XPoint seg8_3121[] = {
    {7,7},{8,9},
};
static XPoint seg9_3121[] = {
    {7,8},{9,9},
};
static XPoint *char3121[] = {
    seg0_3121,seg1_3121,seg2_3121,seg3_3121,seg4_3121,seg5_3121,
    seg6_3121,seg7_3121,seg8_3121,seg9_3121,
    NULL,
};
static int char_p3121[] = {
    XtNumber(seg0_3121),XtNumber(seg1_3121),XtNumber(seg2_3121),
    XtNumber(seg3_3121),XtNumber(seg4_3121),XtNumber(seg5_3121),
    XtNumber(seg6_3121),XtNumber(seg7_3121),XtNumber(seg8_3121),
    XtNumber(seg9_3121),
};
static XPoint seg0_3122[] = {
    {-6,-5},{0,9},
};
static XPoint seg1_3122[] = {
    {-5,-5},{0,7},
};
static XPoint seg2_3122[] = {
    {-4,-5},{1,7},
};
static XPoint seg3_3122[] = {
    {6,-4},{1,7},{0,9},
};
static XPoint seg4_3122[] = {
    {-8,-5},{-1,-5},
};
```

continues

Listing C.4 Continued

```
static XPoint seg5_3122[] = {
    {2,-5},{8,-5},
};
static XPoint seg6_3122[] = {
    {-7,-5},{-4,-3},
};
static XPoint seg7_3122[] = {
    {-2,-5},{-4,-4},
};
static XPoint seg8_3122[] = {
    {4,-5},{6,-4},
};
static XPoint seg9_3122[] = {
    {7,-5},{6,-4},
};
static XPoint *char3122[] = {
    seg0_3122,seg1_3122,seg2_3122,seg3_3122,seg4_3122,seg5_3122,
    seg6_3122,seg7_3122,seg8_3122,seg9_3122,
    NULL,
};
static int char_p3122[] = {
    XtNumber(seg0_3122),XtNumber(seg1_3122),XtNumber(seg2_3122),
    XtNumber(seg3_3122),XtNumber(seg4_3122),XtNumber(seg5_3122),
    XtNumber(seg6_3122),XtNumber(seg7_3122),XtNumber(seg8_3122),
    XtNumber(seg9_3122),
};
static XPoint seg0_3123[] = {
    {-8,-5},{-4,9},
};
static XPoint seg1_3123[] = {
    {-7,-5},{-4,6},
};
static XPoint seg2_3123[] = {
    {-6,-5},{-3,6},
};
static XPoint seg3_3123[] = {
    {0,-5},{-3,6},{-4,9},
};
static XPoint seg4_3123[] = {
    {0,-5},{4,9},
};
static XPoint seg5_3123[] = {
    {1,-5},{4,6},
};
static XPoint seg6_3123[] = {
    {0,-5},{2,-5},{5,6},
};
static XPoint seg7_3123[] = {
    {8,-4},{5,6},{4,9},
};
```

```
static XPoint seg8_3123[] = {
    {-11,-5},{-3,-5},
};
static XPoint seg9_3123[] = {
    {5,-5},{11,-5},
};
static XPoint seg10_3123[] = {
    {-10,-5},{-7,-4},
};
static XPoint seg11_3123[] = {
    {-4,-5},{-6,-4},
};
static XPoint seg12_3123[] = {
    {6,-5},{8,-4},
};
static XPoint seg13_3123[] = {
    {10,-5},{8,-4},
};
static XPoint *char3123[] = {
    seg0_3123,seg1_3123,seg2_3123,seg3_3123,seg4_3123,
    seg5_3123,seg6_3123,seg7_3123,seg8_3123,seg9_3123,
    seg10_3123,seg11_3123,seg12_3123,seg13_3123,
    NULL,
};
static int char_p3123[] = {
    XtNumber(seg0_3123),XtNumber(seg1_3123),XtNumber(seg2_3123),
    XtNumber(seg3_3123),XtNumber(seg4_3123),XtNumber(seg5_3123),
    XtNumber(seg6_3123),XtNumber(seg7_3123),XtNumber(seg8_3123),
    XtNumber(seg9_3123),XtNumber(seg10_3123),XtNumber(seg11_3123),
    XtNumber(seg12_3123),XtNumber(seg13_3123),
};
static XPoint seg0_3124[] = {
    {-6,-5},{4,9},
};
static XPoint seg1_3124[] = {
    {-5,-5},{5,9},
};
static XPoint seg2_3124[] = {
    {-4,-5},{6,9},
};
static XPoint seg3_3124[] = {
    {5,-4},{-5,8},
};
static XPoint seg4_3124[] = {
    {-8,-5},{-1,-5},
};
static XPoint seg5_3124[] = {
    {2,-5},{8,-5},
};
static XPoint seg6_3124[] = {
    {-8,9},{-2,9},
};
```

continues

Listing C.4 Continued

```
static XPoint seg7_3124[] = {
    {1,9},{8,9},
};
static XPoint seg8_3124[] = {
    {-7,-5},{-5,-4},
};
static XPoint seg9_3124[] = {
    {-2,-5},{-4,-4},
};
static XPoint seg10_3124[] = {
    {3,-5},{5,-4},
};
static XPoint seg11_3124[] = {
    {7,-5},{5,-4},
};
static XPoint seg12_3124[] = {
    {-5,8},{-7,9},
};
static XPoint seg13_3124[] = {
    {-5,8},{-3,9},
};
static XPoint seg14_3124[] = {
    {4,8},{2,9},
};
static XPoint seg15_3124[] = {
    {5,8},{7,9},
};
static XPoint *char3124[] = {
    seg0_3124,seg1_3124,seg2_3124,seg3_3124,seg4_3124,
    seg5_3124,seg6_3124,seg7_3124,seg8_3124,seg9_3124,
    seg10_3124,seg11_3124,seg12_3124,seg13_3124,seg14_3124,
    seg15_3124,
    NULL,
};
static int char_p3124[] = {
    XtNumber(seg0_3124),XtNumber(seg1_3124),XtNumber(seg2_3124),
    XtNumber(seg3_3124),XtNumber(seg4_3124),XtNumber(seg5_3124),
    XtNumber(seg6_3124),XtNumber(seg7_3124),XtNumber(seg8_3124),
    XtNumber(seg9_3124),XtNumber(seg10_3124),XtNumber(seg11_3124),
    XtNumber(seg12_3124),XtNumber(seg13_3124),XtNumber(seg14_3124),
    XtNumber(seg15_3124),
};
static XPoint seg0_3125[] = {
    {-6,-5},{0,9},
};
static XPoint seg1_3125[] = {
    {-5,-5},{0,7},
};
static XPoint seg2_3125[] = {
    {-4,-5},{1,7},
};
```

```
static XPoint seg3_3125[] = {
    {6,-4},{1,7},{-2,13},{-4,15},{-6,16},{-8,16},{-9,15},
    {-9,13},{-7,13},{-7,15},{-8,15},{-8,14},
};
static XPoint seg4_3125[] = {
    {-8,-5},{-1,-5},
};
static XPoint seg5_3125[] = {
    {2,-5},{8,-5},
};
static XPoint seg6_3125[] = {
    {-7,-5},{-4,-3},
};
static XPoint seg7_3125[] = {
    {-2,-5},{-4,-4},
};
static XPoint seg8_3125[] = {
    {4,-5},{6,-4},
};
static XPoint seg9_3125[] = {
    {7,-5},{6,-4},
};
static XPoint *char3125[] = {
    seg0_3125,seg1_3125,seg2_3125,seg3_3125,seg4_3125,seg5_3125,
    seg6_3125,seg7_3125,seg8_3125,seg9_3125,
    NULL,
};
static int char_p3125[] = {
    XtNumber(seg0_3125),XtNumber(seg1_3125),XtNumber(seg2_3125),
    XtNumber(seg3_3125),XtNumber(seg4_3125),XtNumber(seg5_3125),
    XtNumber(seg6_3125),XtNumber(seg7_3125),XtNumber(seg8_3125),
    XtNumber(seg9_3125),
};
static XPoint seg0_3126[] = {
    {4,-5},{-6,9},
};
static XPoint seg1_3126[] = {
    {5,-5},{-5,9},
};
static XPoint seg2_3126[] = {
    {6,-5},{-4,9},
};
static XPoint seg3_3126[] = {
    {6,-5},{-6,-5},{-6,-1},
};
static XPoint seg4_3126[] = {
    {-6,9},{6,9},{6,5},
};
static XPoint seg5_3126[] = {
    {-5,-5},{-6,-1},
};
```

continues

Listing C.4 Continued

```
static XPoint seg6_3126[] = {
    {-4,-5},{-6,-2},
};
static XPoint seg7_3126[] = {
    {-3,-5},{-6,-3},
};
static XPoint seg8_3126[] = {
    {-1,-5},{-6,-4},
};
static XPoint seg9_3126[] = {
    {1,9},{6,8},
};
static XPoint seg10_3126[] = {
    {3,9},{6,7},
};
static XPoint seg11_3126[] = {
    {4,9},{6,6},
};
static XPoint seg12_3126[] = {
    {5,9},{6,5},
};
static XPoint *char3126[] = {
    seg0_3126,seg1_3126,seg2_3126,seg3_3126,seg4_3126,
    seg5_3126,seg6_3126,seg7_3126,seg8_3126,seg9_3126,
    seg10_3126,seg11_3126,seg12_3126,
    NULL,
};
static int char_p3126[] = {
    XtNumber(seg0_3126),XtNumber(seg1_3126),XtNumber(seg2_3126),
    XtNumber(seg3_3126),XtNumber(seg4_3126),XtNumber(seg5_3126),
    XtNumber(seg6_3126),XtNumber(seg7_3126),XtNumber(seg8_3126),
    XtNumber(seg9_3126),XtNumber(seg10_3126),XtNumber(seg11_3126),
    XtNumber(seg12_3126),
};
static XPoint seg0_3151[] = {
    {5,-5},{3,2},{3,6},{4,8},{5,9},{7,9},
    {9,7},{10,5},
};
static XPoint seg1_3151[] = {
    {6,-5},{4,2},{4,8},
};
static XPoint seg2_3151[] = {
    {5,-5},{7,-5},{5,2},{4,6},
};
static XPoint seg3_3151[] = {
    {3,2},{3,-1},{2,-4},{0,-5},{-2,-5},{-5,-4},{-7,-1},{-8,2},
    {-8,4},{-7,7},{-6,8},{-4,9},{-2,9},{0,8},{1,7},{2,5},{3,2},
};
static XPoint seg4_3151[] = {
    {-4,-4},{-6,-1},{-7,2},{-7,5},{-6,7},
};
```

```
static XPoint seg5_3151[] = {
    {-2,-5},{-4,-3},{-5,-1},{-6,2},{-6,5},{-5,8},{-4,9},
};
static XPoint *char3151[] = {
    seg0_3151,seg1_3151,seg2_3151,seg3_3151,seg4_3151,seg5_3151,
    NULL,
};
static int char_p3151[] = {
    XtNumber(seg0_3151),XtNumber(seg1_3151),XtNumber(seg2_3151),
    XtNumber(seg3_3151),XtNumber(seg4_3151),XtNumber(seg5_3151),
};
static XPoint seg0_3152[] = {
    {-2,-12},{-4,-5},{-5,1},{-5,5},{-4,7},{-3,8},{-1,9},
    {1,9},{4,8},{6,5},{7,2},{7,0},{6,-3},{5,-4},{3,-5},
    {1,-5},{-1,-4},{-2,-3},{-3,-1},{-4,2},
};
static XPoint seg1_3152[] = {
    {-1,-12},{-3,-5},{-4,-1},{-4,5},{-3,8},
};
static XPoint seg2_3152[] = {
    {4,7},{5,5},{6,2},{6,-1},{5,-3},
};
static XPoint seg3_3152[] = {
    {-5,-12},{0,-12},{-2,-5},{-4,2},
};
static XPoint seg4_3152[] = {
    {1,9},{3,7},{4,5},{5,2},{5,-1},{4,-4},{3,-5},
};
static XPoint seg5_3152[] = {
    {-4,-12},{-1,-11},
};
static XPoint seg6_3152[] = {
    {-3,-12},{-2,-10},
};
static XPoint *char3152[] = {
    seg0_3152,seg1_3152,seg2_3152,seg3_3152,seg4_3152,seg5_3152,
    seg6_3152,
    NULL,
};
static int char_p3152[] = {
    XtNumber(seg0_3152),XtNumber(seg1_3152),XtNumber(seg2_3152),
    XtNumber(seg3_3152),XtNumber(seg4_3152),XtNumber(seg5_3152),
    XtNumber(seg6_3152),
};
static XPoint seg0_3153[] = {
    {5,-1},{5,-2},{4,-2},{4,0},{6,0},{6,-2},
    {5,-4},{3,-5},{0,-5},{-3,-4},{-5,-1},{-6,2},{-6,4},{-5,7},
    {-4,8},{-2,9},{0,9},{3,8},{5,5},
};
static XPoint seg1_3153[] = {
    {-3,-3},{-4,-1},{-5,2},{-5,5},{-4,7},
};
```

continues

Listing C.4 Continued

```
static XPoint seg2_3153[] = {
    {0,-5},{-2,-3},{-3,-1},{-4,2},{-4,5},{-3,8},{-2,9},
};
static XPoint *char3153[] = {
    seg0_3153,seg1_3153,seg2_3153,
    NULL,
};
static int char_p3153[] = {
    XtNumber(seg0_3153),XtNumber(seg1_3153),XtNumber(seg2_3153),
};
static XPoint seg0_3154[] = {
    {7,-12},
    {4,-1},{3,3},{3,6},{4,8},{5,9},{7,9},{9,7},{10,5},
};
static XPoint seg1_3154[] = {
    {8,-12},{5,-1},{4,3},{4,8},
};
static XPoint seg2_3154[] = {
    {4,-12},{9,-12},{5,2},{4,6},
};
static XPoint seg3_3154[] = {
    {3,2},{3,-1},{2,-4},{0,-5},{-2,-5},{-5,-4},{-7,-1},{-8,2},
    {-8,4},{-7,7},{-6,8},{-4,9},{-2,9},{0,8},{1,7},{2,5},{3,2},
};
static XPoint seg4_3154[] = {
    {-5,-3},{-6,-1},{-7,2},{-7,5},{-6,7},
};
static XPoint seg5_3154[] = {
    {-2,-5},{-4,-3},{-5,-1},{-6,2},{-6,5},{-5,8},{-4,9},
};
static XPoint seg6_3154[] = {
    {5,-12},{8,-11},
};
static XPoint seg7_3154[] = {
    {6,-12},{7,-10},
};
static XPoint *char3154[] = {
    seg0_3154,seg1_3154,seg2_3154,seg3_3154,seg4_3154,seg5_3154,
    seg6_3154,seg7_3154,
    NULL,
};
static int char_p3154[] = {
    XtNumber(seg0_3154),XtNumber(seg1_3154),XtNumber(seg2_3154),
    XtNumber(seg3_3154),XtNumber(seg4_3154),XtNumber(seg5_3154),
    XtNumber(seg6_3154),XtNumber(seg7_3154),
};
static XPoint seg0_3155[] = {
    {-5,4},{-1,3},{2,2},{5,0},{6,-2},{5,-4},
    {3,-5},{0,-5},{-3,-4},{-5,-1},{-6,2},{-6,4},{-5,7},{-4,8},
    {-2,9},{0,9},{3,8},{5,6},
};
```

```
static XPoint seg1_3155[] = {
    {-3,-3},{-4,-1},{-5,2},{-5,5},{-4,7},
};
static XPoint seg2_3155[] = {
    {0,-5},{-2,-3},{-3,-1},{-4,2},{-4,5},{-3,8},{-2,9},
};
static XPoint *char3155[] = {
    seg0_3155,seg1_3155,seg2_3155,
    NULL,
};
static int char_p3155[] = {
    XtNumber(seg0_3155),XtNumber(seg1_3155),XtNumber(seg2_3155),
};
static XPoint seg0_3156[] = {
    {8,-10},{8,-11},{7,-11},{7,-9},{9,-9},{9,-11},{8,-12},
    {6,-12},{4,-11},{2,-9},{1,-7},{0,-4},{-1,0},{-3,9},{-4,12},
    {-5,14},{-7,16},
};
static XPoint seg1_3156[] = {
    {2,-8},{1,-5},{0,0},{-2,9},{-3,12},
};
static XPoint seg2_3156[] = {
    {6,-12},{4,-10},{3,-8},{2,-5},{1,0},{-1,8},{-2,11},{-3,13},
    {-5,15},{-7,16},{-9,16},{-10,15},{-10,13},{-8,13},{-8,15},
    {-9,15},{-9,14},
};
static XPoint seg3_3156[] = {
    {-4,-5},{7,-5},
};
static XPoint *char3156[] = {
    seg0_3156,seg1_3156,seg2_3156,seg3_3156,
    NULL,
};
static int char_p3156[] = {
    XtNumber(seg0_3156),XtNumber(seg1_3156),XtNumber(seg2_3156),
    XtNumber(seg3_3156),
};
static XPoint seg0_3157[] = {
    {6,-5},{2,9},{1,12},{-1,15},{-3,16},
};
static XPoint seg1_3157[] = {
    {7,-5},{3,9},{1,13},
};
static XPoint seg2_3157[] = {
    {6,-5},{8,-5},{4,9},{2,13},{0,15},{-3,16},{-6,16},{-8,15},
    {-9,14},{-9,12},{-7,12},{-7,14},{-8,14},{-8,13},
};
static XPoint seg3_3157[] = {
    {4,2},{4,-1},{3,-4},{1,-5},{-1,-5},{-4,-4},{-6,-1},{-7,2},
    {-7,4},{-6,7},{-5,8},{-3,9},{-1,9},{1,8},{2,7},{3,5},{4,2},
};
static XPoint seg4_3157[] = {
    {-4,-3},{-5,-1},{-6,2},{-6,5},{-5,7},
};
```

continues

Listing C.4 Continued

```
static XPoint seg5_3157[] = {
    {-1,-5},{-3,-3},{-4,-1},{-5,2},{-5,5},{-4,8},{-3,9},
};
static XPoint *char3157[] = {
    seg0_3157,seg1_3157,seg2_3157,seg3_3157,seg4_3157,seg5_3157,
    NULL,
};
static int char_p3157[] = {
    XtNumber(seg0_3157),XtNumber(seg1_3157),XtNumber(seg2_3157),
    XtNumber(seg3_3157),XtNumber(seg4_3157),XtNumber(seg5_3157),
};
static XPoint seg0_3158[] = {
    {-3,-12},
    {-9,9},{-7,9},
};
static XPoint seg1_3158[] = {
    {-2,-12},{-8,9},
};
static XPoint seg2_3158[] = {
    {-6,-12},{-1,-12},{-7,9},
};
static XPoint seg3_3158[] = {
    {-5,2},{-3,-2},{-1,-4},{1,-5},{3,-5},{5,-4},{6,-2},{6,1},
    {4,6},
};
static XPoint seg4_3158[] = {
    {5,-4},{5,0},{4,4},{4,8},
};
static XPoint seg5_3158[] = {
    {5,-2},{3,3},{3,6},{4,8},{5,9},{7,9},{9,7},{10,5},
};
static XPoint seg6_3158[] = {
    {-5,-12},{-2,-11},
};
static XPoint seg7_3158[] = {
    {-4,-12},{-3,-10},
};
static XPoint *char3158[] = {
    seg0_3158,seg1_3158,seg2_3158,seg3_3158,seg4_3158,seg5_3158,
    seg6_3158,seg7_3158,
    NULL,
};
static int char_p3158[] = {
    XtNumber(seg0_3158),XtNumber(seg1_3158),XtNumber(seg2_3158),
    XtNumber(seg3_3158),XtNumber(seg4_3158),XtNumber(seg5_3158),
    XtNumber(seg6_3158),XtNumber(seg7_3158),
};
static XPoint seg0_3159[] = {
    {1,-12},{1,-10},{3,-10},{3,-12},{1,-12},
};
```

```
static XPoint seg1_3159[] = {
    {2,-12},{2,-10},
};
static XPoint seg2_3159[] = {
    {1,-11},{3,-11},
};
static XPoint seg3_3159[] = {
    {-6,-1},{-5,-3},{-3,-5},{-1,-5},{0,-4},{1,-2},{1,1},{-1,6},
};
static XPoint seg4_3159[] = {
    {0,-4},{0,0},{-1,4},{-1,8},
};
static XPoint seg5_3159[] = {
    {0,-2},{-2,3},{-2,6},{-1,8},{0,9},{2,9},{4,7},{5,5},
};
static XPoint *char3159[] = {
    seg0_3159,seg1_3159,seg2_3159,seg3_3159,seg4_3159,seg5_3159,
    NULL,
};
static int char_p3159[] = {
    XtNumber(seg0_3159),XtNumber(seg1_3159),XtNumber(seg2_3159),
    XtNumber(seg3_3159),XtNumber(seg4_3159),XtNumber(seg5_3159),
};
static XPoint seg0_3160[] = {
    {3,-12},{3,-10},{5,-10},{5,-12},{3,-12},
};
static XPoint seg1_3160[] = {
    {4,-12},{4,-10},
};
static XPoint seg2_3160[] = {
    {3,-11},{5,-11},
};
static XPoint seg3_3160[] = {
    {-5,-1},{-4,-3},{-2,-5},{0,-5},{1,-4},{2,-2},{2,1},{0,8},
    {-1,11},{-2,13},{-4,15},{-6,16},{-8,16},{-9,15},{-9,13},
    {-7,13},{-7,15},{-8,15},{-8,14},
};
static XPoint seg4_3160[] = {
    {1,-4},{1,1},{-1,8},{-2,11},{-3,13},
};
static XPoint seg5_3160[] = {
    {1,-2},{0,2},{-2,9},{-3,12},{-4,14},{-6,16},
};
static XPoint *char3160[] = {
    seg0_3160,seg1_3160,seg2_3160,seg3_3160,seg4_3160,seg5_3160,
    NULL,
};
static int char_p3160[] = {
    XtNumber(seg0_3160),XtNumber(seg1_3160),XtNumber(seg2_3160),
    XtNumber(seg3_3160),XtNumber(seg4_3160),XtNumber(seg5_3160),
};
static XPoint seg0_3161[] = {
    {-3,-12},{-9,9},
```

continues

Listing C.4 Continued

```
    {-7,9},
};
static XPoint seg1_3161[] = {
    {-2,-12},{-8,9},
};
static XPoint seg2_3161[] = {
    {-6,-12},{-1,-12},{-7,9},
};
static XPoint seg3_3161[] = {
    {7,-3},{7,-4},{6,-4},{6,-2},{8,-2},{8,-4},{7,-5},{5,-5},
    {3,-4},{-1,0},{-3,1},
};
static XPoint seg4_3161[] = {
    {-5,1},{-3,1},{-1,2},{0,3},{2,7},{3,8},{5,8},
};
static XPoint seg5_3161[] = {
    {-1,3},{1,7},{2,8},
};
static XPoint seg6_3161[] = {
    {-3,1},{-2,2},{0,8},{1,9},{3,9},{5,8},{7,5},
};
static XPoint seg7_3161[] = {
    {-5,-12},{-2,-11},
};
static XPoint seg8_3161[] = {
    {-4,-12},{-3,-10},
};
static XPoint *char3161[] = {
    seg0_3161,seg1_3161,seg2_3161,seg3_3161,seg4_3161,seg5_3161,
    seg6_3161,seg7_3161,seg8_3161,
    NULL,
};
static int char_p3161[] = {
    XtNumber(seg0_3161),XtNumber(seg1_3161),XtNumber(seg2_3161),
    XtNumber(seg3_3161),XtNumber(seg4_3161),XtNumber(seg5_3161),
    XtNumber(seg6_3161),XtNumber(seg7_3161),XtNumber(seg8_3161),
};
static XPoint seg0_3162[] = {
    {2,-12},{-1,-1},{-2,3},{-2,6},{-1,8},{0,9},
    {2,9},{4,7},{5,5},
};
static XPoint seg1_3162[] = {
    {3,-12},{0,-1},{-1,3},{-1,8},
};
static XPoint seg2_3162[] = {
    {-1,-12},{4,-12},{0,2},{-1,6},
};
static XPoint seg3_3162[] = {
    {0,-12},{3,-11},
};
```

```
static XPoint seg4_3162[] = {
    {1,-12},{2,-10},
};
static XPoint *char3162[] = {
    seg0_3162,seg1_3162,seg2_3162,seg3_3162,seg4_3162,
    NULL,
};
static int char_p3162[] = {
    XtNumber(seg0_3162),XtNumber(seg1_3162),XtNumber(seg2_3162),
    XtNumber(seg3_3162),XtNumber(seg4_3162),
};
static XPoint seg0_3163[] = {
    {-17,-1},{-16,-3},{-14,-5},{-12,-5},{-11,-4},{-10,-2},
    {-10,1},{-12,9},
};
static XPoint seg1_3163[] = {
    {-11,-4},{-11,1},{-13,9},
};
static XPoint seg2_3163[] = {
    {-11,-2},{-12,2},{-14,9},{-12,9},
};
static XPoint seg3_3163[] = {
    {-10,1},{-8,-2},{-6,-4},{-4,-5},{-2,-5},{0,-4},{1,-2},{1,1},
    {-1,9},
};
static XPoint seg4_3163[] = {
    {0,-4},{0,1},{-2,9},
};
static XPoint seg5_3163[] = {
    {0,-2},{-1,2},{-3,9},{-1,9},
};
static XPoint seg6_3163[] = {
    {1,1},{3,-2},{5,-4},{7,-5},{9,-5},{11,-4},{12,-2},{12,1},
    {10,6},
};
static XPoint seg7_3163[] = {
    {11,-4},{11,0},{10,4},{10,8},
};
static XPoint seg8_3163[] = {
    {11,-2},{9,3},{9,6},{10,8},{11,9},{13,9},{15,7},{16,5},
};
static XPoint *char3163[] = {
    seg0_3163,seg1_3163,seg2_3163,seg3_3163,seg4_3163,seg5_3163,
    seg6_3163,seg7_3163,seg8_3163,
    NULL,
};
static int char_p3163[] = {
    XtNumber(seg0_3163),XtNumber(seg1_3163),XtNumber(seg2_3163),
    XtNumber(seg3_3163),XtNumber(seg4_3163),XtNumber(seg5_3163),
    XtNumber(seg6_3163),XtNumber(seg7_3163),XtNumber(seg8_3163),
};
static XPoint seg0_3164[] = {
    {-11,-1},{-10,-3},{-8,-5},{-6,-5},{-5,-4},{-4,-2},{-4,1},
```

continues

Listing C.4 Continued

```
    {-6,9},
};
static XPoint seg1_3164[] = {
    {-5,-4},{-5,1},{-7,9},
};
static XPoint seg2_3164[] = {
    {-5,-2},{-6,2},{-8,9},{-6,9},
};
static XPoint seg3_3164[] = {
    {-4,1},{-2,-2},{0,-4},{2,-5},{4,-5},{6,-4},{7,-2},{7,1},
    {5,6},
};
static XPoint seg4_3164[] = {
    {6,-4},{6,0},{5,4},{5,8},
};
static XPoint seg5_3164[] = {
    {6,-2},{4,3},{4,6},{5,8},{6,9},{8,9},{10,7},{11,5},
};
static XPoint *char3164[] = {
    seg0_3164,seg1_3164,seg2_3164,seg3_3164,seg4_3164,seg5_3164,
    NULL,
};
static int char_p3164[] = {
    XtNumber(seg0_3164),XtNumber(seg1_3164),XtNumber(seg2_3164),
    XtNumber(seg3_3164),XtNumber(seg4_3164),XtNumber(seg5_3164),
};
static XPoint seg0_3165[] = {
    {-1,-5},{-4,-4},{-6,-1},{-7,2},{-7,4},{-6,7},{-5,8},{-2,9},
    {1,9},{4,8},{6,5},{7,2},{7,0},{6,-3},{5,-4},{2,-5},{-1,-5},
};
static XPoint seg1_3165[] = {
    {-4,-3},{-5,-1},{-6,2},{-6,5},{-5,7},
};
static XPoint seg2_3165[] = {
    {4,7},{5,5},{6,2},{6,-1},{5,-3},
};
static XPoint seg3_3165[] = {
    {-1,-5},{-3,-3},{-4,-1},{-5,2},{-5,5},{-4,8},{-2,9},
};
static XPoint seg4_3165[] = {
    {1,9},{3,7},{4,5},{5,2},{5,-1},{4,-4},{2,-5},
};
static XPoint *char3165[] = {
    seg0_3165,seg1_3165,seg2_3165,seg3_3165,seg4_3165,
    NULL,
};
static int char_p3165[] = {
    XtNumber(seg0_3165),XtNumber(seg1_3165),XtNumber(seg2_3165),
    XtNumber(seg3_3165),XtNumber(seg4_3165),
};
```

```
static XPoint seg0_3166[] = {
    {-10,-1},{-9,-3},{-7,-5},{-5,-5},{-4,-4},{-3,-2},{-3,1},
    {-4,5},{-7,16},
};
static XPoint seg1_3166[] = {
    {-4,-4},{-4,1},{-5,5},{-8,16},
};
static XPoint seg2_3166[] = {
    {-4,-2},{-5,2},{-9,16},
};
static XPoint seg3_3166[] = {
    {-3,2},{-2,-1},{-1,-3},{0,-4},{2,-5},{4,-5},{6,-4},{7,-3},
    {8,0},{8,2},{7,5},{5,8},{2,9},{0,9},{-2,8},{-3,5},{-3,2},
};
static XPoint seg4_3166[] = {
    {6,-3},{7,-1},{7,2},{6,5},{5,7},
};
static XPoint seg5_3166[] = {
    {4,-5},{5,-4},{6,-1},{6,2},{5,5},{4,7},{2,9},
};
static XPoint seg6_3166[] = {
    {-12,16},{-4,16},
};
static XPoint seg7_3166[] = {
    {-8,15},{-11,16},
};
static XPoint seg8_3166[] = {
    {-8,14},{-10,16},
};
static XPoint seg9_3166[] = {
    {-7,14},{-6,16},
};
static XPoint seg10_3166[] = {
    {-8,15},{-5,16},
};
static XPoint *char3166[] = {
    seg0_3166,seg1_3166,seg2_3166,seg3_3166,seg4_3166,seg5_3166,
    seg6_3166,seg7_3166,seg8_3166,seg9_3166,seg10_3166,
    NULL,
};
static int char_p3166[] = {
    XtNumber(seg0_3166),XtNumber(seg1_3166),XtNumber(seg2_3166),
    XtNumber(seg3_3166),XtNumber(seg4_3166),XtNumber(seg5_3166),
    XtNumber(seg6_3166),XtNumber(seg7_3166),XtNumber(seg8_3166),
    XtNumber(seg9_3166),XtNumber(seg10_3166),
};
static XPoint seg0_3167[] = {
    {5,-5},{-1,16},
};
static XPoint seg1_3167[] = {
    {6,-5},{0,16},
};
```

continues

Listing C.4 Continued

```
static XPoint seg2_3167[] = {
    {5,-5},{7,-5},{1,16},
};
static XPoint seg3_3167[] = {
    {3,2},{3,-1},{2,-4},{0,-5},{-2,-5},{-5,-4},{-7,-1},{-8,2},
    {-8,4},{-7,7},{-6,8},{-4,9},{-2,9},{0,8},{1,7},{2,5},{3,2},
};
static XPoint seg4_3167[] = {
    {-5,-3},{-6,-1},{-7,2},{-7,5},{-6,7},
};
static XPoint seg5_3167[] = {
    {-2,-5},{-4,-3},{-5,-1},{-6,2},{-6,5},{-5,8},{-4,9},
};
static XPoint seg6_3167[] = {
    {-4,16},{4,16},
};
static XPoint seg7_3167[] = {
    {0,15},{-3,16},
};
static XPoint seg8_3167[] = {
    {0,14},{-2,16},
};
static XPoint seg9_3167[] = {
    {1,14},{2,16},
};
static XPoint seg10_3167[] = {
    {0,15},{3,16},
};
static XPoint *char3167[] = {
    seg0_3167,seg1_3167,seg2_3167,seg3_3167,seg4_3167,seg5_3167,
    seg6_3167,seg7_3167,seg8_3167,seg9_3167,seg10_3167,
    NULL,
};
static int char_p3167[] = {
    XtNumber(seg0_3167),XtNumber(seg1_3167),XtNumber(seg2_3167),
    XtNumber(seg3_3167),XtNumber(seg4_3167),XtNumber(seg5_3167),
    XtNumber(seg6_3167),XtNumber(seg7_3167),XtNumber(seg8_3167),
    XtNumber(seg9_3167),XtNumber(seg10_3167),
};
static XPoint seg0_3168[] = {
    {-8,-1},{-7,-3},{-5,-5},{-3,-5},{-2,-4},{-1,-2},
    {-1,2},{-3,9},
};
static XPoint seg1_3168[] = {
    {-2,-4},{-2,2},{-4,9},
};
static XPoint seg2_3168[] = {
    {-2,-2},{-3,2},{-5,9},{-3,9},
};
static XPoint seg3_3168[] = {
    {7,-3},{7,-4},{6,-4},{6,-2},{8,-2},{8,-4},{7,-5},{5,-5},
    {3,-4},{1,-2},{-1,2},
};
static XPoint *char3168[] = {
    seg0_3168,seg1_3168,seg2_3168,seg3_3168,
    NULL,
};
static int char_p3168[] = {
    XtNumber(seg0_3168),XtNumber(seg1_3168),XtNumber(seg2_3168),
    XtNumber(seg3_3168),
};
static XPoint seg0_3169[] = {
    {6,-2},{6,-3},{5,-3},{5,-1},{7,-1},{7,-3},
    {6,-4},{3,-5},{0,-5},{-3,-4},{-4,-3},{-4,-1},{-3,1},{-1,2},
    {2,3},{4,4},{5,6},
};
static XPoint seg1_3169[] = {
    {-3,-4},{-4,-1},
};
static XPoint seg2_3169[] = {
    {-3,0},{-1,1},{2,2},{4,3},
};
static XPoint seg3_3169[] = {
    {5,4},{4,8},
};
static XPoint seg4_3169[] = {
    {-4,-3},{-3,-1},{-1,0},{2,1},{4,2},{5,4},{5,6},{4,8},
    {1,9},{-2,9},{-5,8},{-6,7},{-6,5},{-4,5},{-4,7},{-5,7},
    {-5,6},
};
static XPoint *char3169[] = {
    seg0_3169,seg1_3169,seg2_3169,seg3_3169,seg4_3169,
    NULL,
};
static int char_p3169[] = {
    XtNumber(seg0_3169),XtNumber(seg1_3169),XtNumber(seg2_3169),
    XtNumber(seg3_3169),XtNumber(seg4_3169),
};
static XPoint seg0_3170[] = {
    {2,-12},{-1,-1},{-2,3},{-2,6},{-1,8},{0,9},{2,9},{4,7},
    {5,5},
};
static XPoint seg1_3170[] = {
    {3,-12},{0,-1},{-1,3},{-1,8},
};
static XPoint seg2_3170[] = {
    {2,-12},{4,-12},{0,2},{-1,6},
};
static XPoint seg3_3170[] = {
    {-4,-5},{6,-5},
};
static XPoint *char3170[] = {
    seg0_3170,seg1_3170,seg2_3170,seg3_3170,
    NULL,
};
```

continues

Listing C.4 Continued

```
static int char_p3170[] = {
    XtNumber(seg0_3170),XtNumber(seg1_3170),XtNumber(seg2_3170),
    XtNumber(seg3_3170),
};
static XPoint seg0_3171[] = {
    {-11,-1},{-10,-3},{-8,-5},{-6,-5},{-5,-4},{-4,-2},
    {-4,1},{-6,6},
};
static XPoint seg1_3171[] = {
    {-5,-4},{-5,0},{-6,4},{-6,8},
};
static XPoint seg2_3171[] = {
    {-5,-2},{-7,3},{-7,6},{-6,8},{-4,9},{-2,9},{0,8},{2,6},
    {4,3},
};
static XPoint seg3_3171[] = {
    {6,-5},{4,3},{4,6},{5,8},{6,9},{8,9},{10,7},{11,5},
};
static XPoint seg4_3171[] = {
    {7,-5},{5,3},{5,8},
};
static XPoint seg5_3171[] = {
    {6,-5},{8,-5},{6,2},{5,6},
};
static XPoint *char3171[] = {
    seg0_3171,seg1_3171,seg2_3171,seg3_3171,seg4_3171,seg5_3171,
    NULL,
};
static int char_p3171[] = {
    XtNumber(seg0_3171),XtNumber(seg1_3171),XtNumber(seg2_3171),
    XtNumber(seg3_3171),XtNumber(seg4_3171),XtNumber(seg5_3171),
};
static XPoint seg0_3172[] = {
    {-9,-1},{-8,-3},{-6,-5},{-4,-5},
    {-3,-4},{-2,-2},{-2,1},{-4,6},
};
static XPoint seg1_3172[] = {
    {-3,-4},{-3,0},{-4,4},{-4,8},
};
static XPoint seg2_3172[] = {
    {-3,-2},{-5,3},{-5,6},{-4,8},{-2,9},{0,9},{2,8},{4,6},
    {6,3},{7,-1},{7,-5},{6,-5},{6,-4},{7,-2},
};
static XPoint *char3172[] = {
    seg0_3172,seg1_3172,seg2_3172,
    NULL,
};
```

```
static int char_p3172[] = {
    XtNumber(seg0_3172),XtNumber(seg1_3172),XtNumber(seg2_3172),
};
static XPoint seg0_3173[] = {
    {-14,-1},{-13,-3},{-11,-5},
    {-9,-5},{-8,-4},{-7,-2},{-7,1},{-9,6},
};
static XPoint seg1_3173[] = {
    {-8,-4},{-8,0},{-9,4},{-9,8},
};
static XPoint seg2_3173[] = {
    {-8,-2},{-10,3},{-10,6},{-9,8},{-7,9},{-5,9},{-3,8},{-1,6},
    {0,3},
};
static XPoint seg3_3173[] = {
    {2,-5},{0,3},{0,6},{1,8},{3,9},{5,9},{7,8},{9,6},
    {11,3},{12,-1},{12,-5},{11,-5},{11,-4},{12,-2},
};
static XPoint seg4_3173[] = {
    {3,-5},{1,3},{1,8},
};
static XPoint seg5_3173[] = {
    {2,-5},{4,-5},{2,2},{1,6},
};
static XPoint *char3173[] = {
    seg0_3173,seg1_3173,seg2_3173,seg3_3173,seg4_3173,seg5_3173,
    NULL,
};
static int char_p3173[] = {
    XtNumber(seg0_3173),XtNumber(seg1_3173),XtNumber(seg2_3173),
    XtNumber(seg3_3173),XtNumber(seg4_3173),XtNumber(seg5_3173),
};
static XPoint seg0_3174[] = {
    {-8,-1},{-6,-4},{-4,-5},{-2,-5},
    {0,-4},{1,-2},{1,0},
};
static XPoint seg1_3174[] = {
    {-2,-5},{-1,-4},{-1,0},{-2,4},{-3,6},{-5,8},{-7,9},{-9,9},
    {-10,8},{-10,6},{-8,6},{-8,8},{-9,8},{-9,7},
};
static XPoint seg2_3174[] = {
    {0,-3},{0,0},{-1,4},{-1,7},
};
static XPoint seg3_3174[] = {
    {8,-3},{8,-4},{7,-4},{7,-2},{9,-2},{9,-4},{8,-5},{6,-5},
    {4,-4},{2,-2},{1,0},{0,4},{0,8},{1,9},
};
static XPoint seg4_3174[] = {
    {-2,4},{-2,6},{-1,8},{1,9},{3,9},{5,8},{7,5},
};
static XPoint *char3174[] = {
    seg0_3174,seg1_3174,seg2_3174,seg3_3174,seg4_3174,
    NULL,
};
```

continues

Listing C.4 Continued

```
static int char_p3174[] = {
    XtNumber(seg0_3174),XtNumber(seg1_3174),XtNumber(seg2_3174),
    XtNumber(seg3_3174),XtNumber(seg4_3174),
};
static XPoint seg0_3175[] = {
    {-10,-1},
    {-9,-3},{-7,-5},{-5,-5},{-4,-4},{-3,-2},{-3,1},{-5,6},
};
static XPoint seg1_3175[] = {
    {-4,-4},{-4,0},{-5,4},{-5,8},
};
static XPoint seg2_3175[] = {
    {-4,-2},{-6,3},{-6,6},{-5,8},{-3,9},{-1,9},{1,8},{3,6},
    {5,2},
};
static XPoint seg3_3175[] = {
    {7,-5},{3,9},{2,12},{0,15},{-2,16},
};
static XPoint seg4_3175[] = {
    {8,-5},{4,9},{2,13},
};
static XPoint seg5_3175[] = {
    {7,-5},{9,-5},{5,9},{3,13},{1,15},{-2,16},{-5,16},{-7,15},
    {-8,14},{-8,12},{-6,12},{-6,14},{-7,14},{-7,13},
};
static XPoint *char3175[] = {
    seg0_3175,seg1_3175,seg2_3175,seg3_3175,seg4_3175,seg5_3175,
    NULL,
};
static int char_p3175[] = {
    XtNumber(seg0_3175),XtNumber(seg1_3175),XtNumber(seg2_3175),
    XtNumber(seg3_3175),XtNumber(seg4_3175),XtNumber(seg5_3175),
};
static XPoint seg0_3176[] = {
    {7,-5},{6,-3},{4,-1},
    {-4,5},{-6,7},{-7,9},
};
static XPoint seg1_3176[] = {
    {6,-3},{-3,-3},{-5,-2},{-6,0},
};
static XPoint seg2_3176[] = {
    {4,-3},{0,-4},{-3,-4},{-4,-3},
};
static XPoint seg3_3176[] = {
    {4,-3},{0,-5},{-3,-5},{-5,-3},{-6,0},
};
static XPoint seg4_3176[] = {
    {-6,7},{3,7},{5,6},{6,4},
};
```

```
static XPoint seg5_3176[] = {
    {-4,7},{0,8},{3,8},{4,7},
};
static XPoint seg6_3176[] = {
    {-4,7},{0,9},{3,9},{5,7},{6,4},
};
static XPoint *char3176[] = {
    seg0_3176,seg1_3176,seg2_3176,seg3_3176,seg4_3176,seg5_3176,
    seg6_3176,
    NULL,
};
static int char_p3176[] = {
    XtNumber(seg0_3176),XtNumber(seg1_3176),XtNumber(seg2_3176),
    XtNumber(seg3_3176),XtNumber(seg4_3176),XtNumber(seg5_3176),
    XtNumber(seg6_3176),
};
static XPoint seg0_3199[] = {
    {-8,8},
};
static XPoint *char3199[] = {
    seg0_3199,
    NULL,
};
static int char_p3199[] = {
    XtNumber(seg0_3199),
};
static XPoint seg0_3200[] = {
    {-1,-12},{-4,-11},{-6,-8},{-7,-3},{-7,0},{-6,5},{-4,8},
    {-1,9},{1,9},{4,8},{6,5},{7,0},{7,-3},{6,-8},{4,-11},
    {1,-12},{-1,-12},
};
static XPoint seg1_3200[] = {
    {-4,-10},{-5,-8},{-6,-4},{-6,1},{-5,5},{-4,7},
};
static XPoint seg2_3200[] = {
    {4,7},{5,5},{6,1},{6,-4},{5,-8},{4,-10},
};
static XPoint seg3_3200[] = {
    {-1,-12},{-3,-11},{-4,-9},{-5,-4},{-5,1},{-4,6},{-3,8},
    {-1,9},
};
static XPoint seg4_3200[] = {
    {1,9},{3,8},{4,6},{5,1},{5,-4},{4,-9},{3,-11},{1,-12},
};
static XPoint *char3200[] = {
    seg0_3200,seg1_3200,seg2_3200,seg3_3200,seg4_3200,
    NULL,
};
static int char_p3200[] = {
    XtNumber(seg0_3200),XtNumber(seg1_3200),XtNumber(seg2_3200),
    XtNumber(seg3_3200),XtNumber(seg4_3200),
};
```

continues

Listing C.4 Continued

```
static XPoint seg0_3201[] = {
    {-1,-10},{-1,9},
};
static XPoint seg1_3201[] = {
    {0,-10},{0,8},
};
static XPoint seg2_3201[] = {
    {1,-12},{1,9},
};
static XPoint seg3_3201[] = {
    {1,-12},{-2,-9},{-4,-8},
};
static XPoint seg4_3201[] = {
    {-5,9},{5,9},
};
static XPoint seg5_3201[] = {
    {-1,8},{-3,9},
};
static XPoint seg6_3201[] = {
    {-1,7},{-2,9},
};
static XPoint seg7_3201[] = {
    {1,7},{2,9},
};
static XPoint seg8_3201[] = {
    {1,8},{3,9},
};
static XPoint *char3201[] = {
    seg0_3201,seg1_3201,seg2_3201,seg3_3201,seg4_3201,seg5_3201,
    seg6_3201,seg7_3201,seg8_3201,
    NULL,
};
static int char_p3201[] = {
    XtNumber(seg0_3201),XtNumber(seg1_3201),XtNumber(seg2_3201),
    XtNumber(seg3_3201),XtNumber(seg4_3201),XtNumber(seg5_3201),
    XtNumber(seg6_3201),XtNumber(seg7_3201),XtNumber(seg8_3201),
};
static XPoint seg0_3202[] = {
    {-6,-8},{-6,-7},{-5,-7},{-5,-8},{-6,-8},
};
static XPoint seg1_3202[] = {
    {-6,-9},{-5,-9},{-4,-8},{-4,-7},{-5,-6},{-6,-6},{-7,-7},
    {-7,-8},{-6,-10},{-5,-11},{-2,-12},{2,-12},{5,-11},{6,-10},
    {7,-8},{7,-6},{6,-4},{3,-2},{-2,0},{-4,1},{-6,3},{-7,6},
    {-7,9},
};
static XPoint seg2_3202[] = {
    {5,-10},{6,-8},{6,-6},{5,-4},
};
static XPoint seg3_3202[] = {
    {2,-12},{4,-11},{5,-8},{5,-6},{4,-4},{2,-2},{-2,0},
};
```

```
static XPoint seg4_3202[] = {
    {-7,7},{-6,6},{-4,6},{1,7},{5,7},{7,6},
};
static XPoint seg5_3202[] = {
    {-4,6},{1,8},{5,8},{6,7},
};
static XPoint seg6_3202[] = {
    {-4,6},{1,9},{5,9},{6,8},{7,6},{7,4},
};
static XPoint *char3202[] = {
    seg0_3202,seg1_3202,seg2_3202,seg3_3202,seg4_3202,seg5_3202,
    seg6_3202,
    NULL,
};
static int char_p3202[] = {
    XtNumber(seg0_3202),XtNumber(seg1_3202),XtNumber(seg2_3202),
    XtNumber(seg3_3202),XtNumber(seg4_3202),XtNumber(seg5_3202),
    XtNumber(seg6_3202),
};
static XPoint seg0_3203[] = {
    {-6,-8},{-6,-7},
    {-5,-7},{-5,-8},{-6,-8},
};
static XPoint seg1_3203[] = {
    {-6,-9},{-5,-9},{-4,-8},{-4,-7},{-5,-6},{-6,-6},{-7,-7},
    {-7,-8},{-6,-10},{-5,-11},{-2,-12},{2,-12},{5,-11},{6,-9},
    {6,-6},{5,-4},{2,-3},
};
static XPoint seg2_3203[] = {
    {4,-11},{5,-9},{5,-6},{4,-4},
};
static XPoint seg3_3203[] = {
    {1,-12},{3,-11},{4,-9},{4,-6},{3,-4},{1,-3},
};
static XPoint seg4_3203[] = {
    {-1,-3},{2,-3},{4,-2},{6,0},{7,2},{7,5},{6,7},{5,8},
    {2,9},{-2,9},{-5,8},{-6,7},{-7,5},{-7,4},{-6,3},{-5,3},
    {-4,4},{-4,5},{-5,6},{-6,6},
};
static XPoint seg5_3203[] = {
    {5,0},{6,2},{6,5},{5,7},
};
static XPoint seg6_3203[] = {
    {1,-3},{3,-2},{4,-1},{5,2},{5,5},{4,8},{2,9},
};
static XPoint seg7_3203[] = {
    {-6,4},{-6,5},{-5,5},{-5,4},{-6,4},
};
static XPoint *char3203[] = {
    seg0_3203,seg1_3203,seg2_3203,seg3_3203,seg4_3203,seg5_3203,
    seg6_3203,seg7_3203,
    NULL,
};
```

continues

Listing C.4 Continued

```
static int char_p3203[] = {
    XtNumber(seg0_3203),XtNumber(seg1_3203),XtNumber(seg2_3203),
    XtNumber(seg3_3203),XtNumber(seg4_3203),XtNumber(seg5_3203),
    XtNumber(seg6_3203),XtNumber(seg7_3203),
};
static XPoint seg0_3204[] = {
    {1,-9},{1,9},
};
static XPoint seg1_3204[] = {
    {2,-10},{2,8},
};
static XPoint seg2_3204[] = {
    {3,-12},{3,9},
};
static XPoint seg3_3204[] = {
    {3,-12},{-8,3},{8,3},
};
static XPoint seg4_3204[] = {
    {-2,9},{6,9},
};
static XPoint seg5_3204[] = {
    {1,8},{-1,9},
};
static XPoint seg6_3204[] = {
    {1,7},{0,9},
};
static XPoint seg7_3204[] = {
    {3,7},{4,9},
};
static XPoint seg8_3204[] = {
    {3,8},{5,9},
};
static XPoint *char3204[] = {
    seg0_3204,seg1_3204,seg2_3204,seg3_3204,seg4_3204,seg5_3204,
    seg6_3204,seg7_3204,seg8_3204,
    NULL,
};
static int char_p3204[] = {
    XtNumber(seg0_3204),XtNumber(seg1_3204),XtNumber(seg2_3204),
    XtNumber(seg3_3204),XtNumber(seg4_3204),XtNumber(seg5_3204),
    XtNumber(seg6_3204),XtNumber(seg7_3204),XtNumber(seg8_3204),
};
static XPoint seg0_3205[] = {
    {-5,-12},{-7,-2},{-5,-4},{-2,-5},{1,-5},{4,-4},
    {6,-2},{7,1},{7,3},{6,6},{4,8},{1,9},{-2,9},{-5,8},{-6,7},
    {-7,5},{-7,4},{-6,3},{-5,3},{-4,4},{-4,5},{-5,6},{-6,6},
};
static XPoint seg1_3205[] = {
    {5,-2},{6,0},{6,4},{5,6},
};
```

```
static XPoint seg2_3205[] = {
    {1,-5},{3,-4},{4,-3},{5,0},{5,4},{4,7},{3,8},{1,9},
};
static XPoint seg3_3205[] = {
    {-6,4},{-6,5},{-5,5},{-5,4},{-6,4},
};
static XPoint seg4_3205[] = {
    {-5,-12},{5,-12},
};
static XPoint seg5_3205[] = {
    {-5,-11},{3,-11},
};
static XPoint seg6_3205[] = {
    {-5,-10},{-1,-10},{3,-11},{5,-12},
};
static XPoint *char3205[] = {
    seg0_3205,seg1_3205,seg2_3205,seg3_3205,seg4_3205,seg5_3205,
    seg6_3205,
    NULL,
};
static int char_p3205[] = {
    XtNumber(seg0_3205),XtNumber(seg1_3205),XtNumber(seg2_3205),
    XtNumber(seg3_3205),XtNumber(seg4_3205),XtNumber(seg5_3205),
    XtNumber(seg6_3205),
};
static XPoint seg0_3206[] = {
    {4,-9},{4,-8},{5,-8},{5,-9},
    {4,-9},
};
static XPoint seg1_3206[] = {
    {5,-10},{4,-10},{3,-9},{3,-8},{4,-7},{5,-7},{6,-8},{6,-9},
    {5,-11},{3,-12},{0,-12},{-3,-11},{-5,-9},{-6,-7},{-7,-3},
    {-7,3},{-6,6},{-4,8},{-1,9},{1,9},{4,8},{6,6},{7,3},{7,2},
    {6,-1},{4,-3},{1,-4},{-1,-4},{-3,-3},{-4,-2},{-5,0},
};
static XPoint seg2_3206[] = {
    {-4,-9},{-5,-7},{-6,-3},{-6,3},{-5,6},{-4,7},
};
static XPoint seg3_3206[] = {
    {5,6},{6,4},{6,1},{5,-1},
};
static XPoint seg4_3206[] = {
    {0,-12},{-2,-11},{-3,-10},{-4,-8},{-5,-4},{-5,3},{-4,6},
    {-3,8},{-1,9},
};
static XPoint seg5_3206[] = {
    {1,9},{3,8},{4,7},{5,4},{5,1},{4,-2},{3,-3},{1,-4},
};
static XPoint *char3206[] = {
    seg0_3206,seg1_3206,seg2_3206,seg3_3206,seg4_3206,seg5_3206,
    NULL,
};
```

continues

Listing C.4 Continued

```
static int char_p3206[] = {
    XtNumber(seg0_3206),XtNumber(seg1_3206),XtNumber(seg2_3206),
    XtNumber(seg3_3206),XtNumber(seg4_3206),XtNumber(seg5_3206),
};
static XPoint seg0_3207[] = {
    {-7,-12},{-7,-6},
};
static XPoint seg1_3207[] = {
    {7,-12},{7,-9},{6,-6},{2,-1},{1,1},{0,5},{0,9},
};
static XPoint seg2_3207[] = {
    {1,0},{0,2},{-1,5},{-1,9},
};
static XPoint seg3_3207[] = {
    {6,-6},{1,-1},{-1,2},{-2,5},{-2,9},{0,9},
};
static XPoint seg4_3207[] = {
    {-7,-8},{-6,-10},{-4,-12},{-2,-12},{3,-9},{5,-9},{6,-10},
    {7,-12},
};
static XPoint seg5_3207[] = {
    {-5,-10},{-4,-11},{-2,-11},{0,-10},
};
static XPoint seg6_3207[] = {
    {-7,-8},{-6,-9},{-4,-10},{-2,-10},{3,-9},
};
static XPoint *char3207[] = {
    seg0_3207,seg1_3207,seg2_3207,seg3_3207,seg4_3207,seg5_3207,
    seg6_3207,
    NULL,
};
static int char_p3207[] = {
    XtNumber(seg0_3207),XtNumber(seg1_3207),XtNumber(seg2_3207),
    XtNumber(seg3_3207),XtNumber(seg4_3207),XtNumber(seg5_3207),
    XtNumber(seg6_3207),
};
static XPoint seg0_3208[] = {
    {-2,-12},{-5,-11},{-6,-9},{-6,-6},{-5,-4},{-2,-3},{2,-3},
    {5,-4},{6,-6},{6,-9},{5,-11},{2,-12},{-2,-12},
};
static XPoint seg1_3208[] = {
    {-4,-11},{-5,-9},{-5,-6},{-4,-4},
};
static XPoint seg2_3208[] = {
    {4,-4},{5,-6},{5,-9},{4,-11},
};
static XPoint seg3_3208[] = {
    {-2,-12},{-3,-11},{-4,-9},{-4,-6},{-3,-4},{-2,-3},
};
```

```
static XPoint seg4_3208[] = {
    {2,-3},{3,-4},{4,-6},{4,-9},{3,-11},{2,-12},
};
static XPoint seg5_3208[] = {
    {-2,-3},{-5,-2},{-6,-1},{-7,1},{-7,5},{-6,7},{-5,8},{-2,9},
    {2,9},{5,8},{6,7},{7,5},{7,1},{6,-1},{5,-2},{2,-3},
};
static XPoint seg6_3208[] = {
    {-5,-1},{-6,1},{-6,5},{-5,7},
};
static XPoint seg7_3208[] = {
    {5,7},{6,5},{6,1},{5,-1},
};
static XPoint seg8_3208[] = {
    {-2,-3},{-4,-2},{-5,1},{-5,5},{-4,8},{-2,9},
};
static XPoint seg9_3208[] = {
    {2,9},{4,8},{5,5},{5,1},{4,-2},{2,-3},
};
static XPoint *char3208[] = {
    seg0_3208,seg1_3208,seg2_3208,seg3_3208,seg4_3208,seg5_3208,
    seg6_3208,seg7_3208,seg8_3208,seg9_3208,
    NULL,
};
static int char_p3208[] = {
    XtNumber(seg0_3208),XtNumber(seg1_3208),XtNumber(seg2_3208),
    XtNumber(seg3_3208),XtNumber(seg4_3208),XtNumber(seg5_3208),
    XtNumber(seg6_3208),XtNumber(seg7_3208),XtNumber(seg8_3208),
    XtNumber(seg9_3208),
};
static XPoint seg0_3209[] = {
    {-5,5},{-5,6},
    {-4,6},{-4,5},{-5,5},
};
static XPoint seg1_3209[] = {
    {5,-3},{4,-1},{3,0},{1,1},{-1,1},{-4,0},{-6,-2},{-7,-5},
    {-7,-6},{-6,-9},{-4,-11},{-1,-12},{1,-12},{4,-11},{6,-9},
    {7,-6},{7,0},{6,4},{5,6},{3,8},{0,9},{-3,9},{-5,8},{-6,6},
    {-6,5},{-5,4},{-4,4},{-3,5},{-3,6},{-4,7},{-5,7},
};
static XPoint seg2_3209[] = {
    {-5,-2},{-6,-4},{-6,-7},{-5,-9},
};
static XPoint seg3_3209[] = {
    {4,-10},{5,-9},{6,-6},{6,0},{5,4},{4,6},
};
static XPoint seg4_3209[] = {
    {-1,1},{-3,0},{-4,-1},{-5,-4},{-5,-7},{-4,-10},{-3,-11},
    {-1,-12},
};
static XPoint seg5_3209[] = {
    {1,-12},{3,-11},{4,-9},{5,-6},{5,1},{4,5},{3,7},{2,8},
    {0,9},
};
```

continues

Listing C.4 Continued

```
static XPoint *char3209[] = {
    seg0_3209,seg1_3209,seg2_3209,seg3_3209,seg4_3209,seg5_3209,
    NULL,
};
static int char_p3209[] = {
    XtNumber(seg0_3209),XtNumber(seg1_3209),XtNumber(seg2_3209),
    XtNumber(seg3_3209),XtNumber(seg4_3209),XtNumber(seg5_3209),
};
static XPoint seg0_3210[] = {
    {0,6},{-1,7},{-1,8},{0,9},{1,9},{2,8},{2,7},{1,6},
    {0,6},
};
static XPoint seg1_3210[] = {
    {0,7},{0,8},{1,8},{1,7},{0,7},
};
static XPoint *char3210[] = {
    seg0_3210,seg1_3210,
    NULL,
};
static int char_p3210[] = {
    XtNumber(seg0_3210),XtNumber(seg1_3210),
};
static XPoint seg0_3211[] = {
    {2,8},{1,9},{0,9},
    {-1,8},{-1,7},{0,6},{1,6},{2,7},{2,10},{1,12},{-1,13},
};
static XPoint seg1_3211[] = {
    {0,7},{0,8},{1,8},{1,7},{0,7},
};
static XPoint seg2_3211[] = {
    {1,9},{2,10},
};
static XPoint seg3_3211[] = {
    {2,8},{1,12},
};
static XPoint *char3211[] = {
    seg0_3211,seg1_3211,seg2_3211,seg3_3211,
    NULL,
};
static int char_p3211[] = {
    XtNumber(seg0_3211),XtNumber(seg1_3211),XtNumber(seg2_3211),
    XtNumber(seg3_3211),
};
static XPoint seg0_3212[] = {
    {0,-5},{-1,-4},{-1,-3},{0,-2},{1,-2},{2,-3},
    {2,-4},{1,-5},{0,-5},
};
static XPoint seg1_3212[] = {
    {0,-4},{0,-3},{1,-3},{1,-4},{0,-4},
};
```

```
static XPoint seg2_3212[] = {
    {0,6},{-1,7},{-1,8},{0,9},{1,9},{2,8},{2,7},{1,6},
    {0,6},
};
static XPoint seg3_3212[] = {
    {0,7},{0,8},{1,8},{1,7},{0,7},
};
static XPoint *char3212[] = {
    seg0_3212,seg1_3212,seg2_3212,seg3_3212,
    NULL,
};
static int char_p3212[] = {
    XtNumber(seg0_3212),XtNumber(seg1_3212),XtNumber(seg2_3212),
    XtNumber(seg3_3212),
};
static XPoint seg0_3213[] = {
    {0,-5},{-1,-4},{-1,-3},
    {0,-2},{1,-2},{2,-3},{2,-4},{1,-5},{0,-5},
};
static XPoint seg1_3213[] = {
    {0,-4},{0,-3},{1,-3},{1,-4},{0,-4},
};
static XPoint seg2_3213[] = {
    {2,8},{1,9},{0,9},{-1,8},{-1,7},{0,6},{1,6},{2,7},
    {2,10},{1,12},{-1,13},
};
static XPoint seg3_3213[] = {
    {0,7},{0,8},{1,8},{1,7},{0,7},
};
static XPoint seg4_3213[] = {
    {1,9},{2,10},
};
static XPoint seg5_3213[] = {
    {2,8},{1,12},
};
static XPoint *char3213[] = {
    seg0_3213,seg1_3213,seg2_3213,seg3_3213,seg4_3213,seg5_3213,
    NULL,
};
static int char_p3213[] = {
    XtNumber(seg0_3213),XtNumber(seg1_3213),XtNumber(seg2_3213),
    XtNumber(seg3_3213),XtNumber(seg4_3213),XtNumber(seg5_3213),
};
static XPoint seg0_3214[] = {
    {0,-12},{-1,-11},{-1,-9},{0,-1},
};
static XPoint seg1_3214[] = {
    {0,-12},{0,2},{1,2},
};
static XPoint seg2_3214[] = {
    {0,-12},{1,-12},{1,2},
};
```

continues

Listing C.4 Continued

```
static XPoint seg3_3214[] = {
    {1,-12},{2,-11},{2,-9},{1,-1},
};
static XPoint seg4_3214[] = {
    {0,6},{-1,7},{-1,8},{0,9},{1,9},{2,8},{2,7},{1,6},
    {0,6},
};
static XPoint seg5_3214[] = {
    {0,7},{0,8},{1,8},{1,7},{0,7},
};
static XPoint *char3214[] = {
    seg0_3214,seg1_3214,seg2_3214,seg3_3214,seg4_3214,seg5_3214,
    NULL,
};
static int char_p3214[] = {
    XtNumber(seg0_3214),XtNumber(seg1_3214),XtNumber(seg2_3214),
    XtNumber(seg3_3214),XtNumber(seg4_3214),XtNumber(seg5_3214),
};
static XPoint seg0_3215[] = {
    {-5,-7},{-5,-8},{-4,-8},
    {-4,-6},{-6,-6},{-6,-8},{-5,-10},{-4,-11},{-2,-12},{2,-12},
    {5,-11},{6,-10},{7,-8},{7,-6},{6,-4},{5,-3},{1,-1},
};
static XPoint seg1_3215[] = {
    {5,-10},{6,-9},{6,-5},{5,-4},
};
static XPoint seg2_3215[] = {
    {2,-12},{4,-11},{5,-9},{5,-5},{4,-3},{3,-2},
};
static XPoint seg3_3215[] = {
    {0,-1},{0,2},{1,2},{1,-1},{0,-1},
};
static XPoint seg4_3215[] = {
    {0,6},{-1,7},{-1,8},{0,9},{1,9},{2,8},{2,7},{1,6},
    {0,6},
};
static XPoint seg5_3215[] = {
    {0,7},{0,8},{1,8},{1,7},{0,7},
};
static XPoint *char3215[] = {
    seg0_3215,seg1_3215,seg2_3215,seg3_3215,seg4_3215,seg5_3215,
    NULL,
};
static int char_p3215[] = {
    XtNumber(seg0_3215),XtNumber(seg1_3215),XtNumber(seg2_3215),
    XtNumber(seg3_3215),XtNumber(seg4_3215),XtNumber(seg5_3215),
};
static XPoint seg0_3216[] = {
    {2,-12},{0,-11},{-1,-9},
    {-1,-6},{0,-5},{1,-5},{2,-6},{2,-7},{1,-8},{0,-8},{-1,-7},
};
```

```
static XPoint seg1_3216[] = {
    {0,-7},{0,-6},{1,-6},{1,-7},{0,-7},
};
static XPoint seg2_3216[] = {
    {0,-11},{-1,-7},
};
static XPoint seg3_3216[] = {
    {-1,-9},{0,-8},
};
static XPoint *char3216[] = {
    seg0_3216,seg1_3216,seg2_3216,seg3_3216,
    NULL,
};
static int char_p3216[] = {
    XtNumber(seg0_3216),XtNumber(seg1_3216),XtNumber(seg2_3216),
    XtNumber(seg3_3216),
};
static XPoint seg0_3217[] = {
    {2,-10},{1,-9},{0,-9},{-1,-10},{-1,-11},{0,-12},
    {1,-12},{2,-11},{2,-8},{1,-6},{-1,-5},
};
static XPoint seg1_3217[] = {
    {0,-11},{0,-10},{1,-10},{1,-11},{0,-11},
};
static XPoint seg2_3217[] = {
    {1,-9},{2,-8},
};
static XPoint seg3_3217[] = {
    {2,-10},{1,-6},
};
static XPoint *char3217[] = {
    seg0_3217,seg1_3217,seg2_3217,seg3_3217,
    NULL,
};
static int char_p3217[] = {
    XtNumber(seg0_3217),XtNumber(seg1_3217),XtNumber(seg2_3217),
    XtNumber(seg3_3217),
};
static XPoint seg0_3218[] = {
    {9,-3},{9,-4},{8,-4},{8,-2},{10,-2},{10,-4},
    {9,-5},{8,-5},{7,-4},{6,-2},{4,3},{2,6},{0,8},{-2,9},
    {-6,9},{-8,8},{-9,6},{-9,3},{-8,1},{-2,-3},{0,-5},{1,-7},
    {1,-9},{0,-11},{-2,-12},{-4,-11},{-5,-9},{-5,-6},{-4,-3},
    {-2,0},{2,5},{5,8},{7,9},{9,9},{10,7},{10,6},
};
static XPoint seg1_3218[] = {
    {-7,8},{-8,6},{-8,3},{-7,1},{-6,0},
};
static XPoint seg2_3218[] = {
    {0,-5},{1,-9},
};
static XPoint seg3_3218[] = {
    {1,-7},{0,-11},
};
```

continues

Listing C.4 Continued

```
static XPoint seg4_3218[] = {
    {-4,-11},{-5,-7},
};
static XPoint seg5_3218[] = {
    {-4,-4},{-2,-1},{2,4},{5,7},{7,8},
};
static XPoint seg6_3218[] = {
    {-4,9},{-6,8},{-7,6},{-7,3},{-6,1},{-2,-3},
};
static XPoint seg7_3218[] = {
    {-5,-9},{-4,-5},{-1,-1},{3,4},{6,7},{8,8},{9,8},{10,7},
};
static XPoint *char3218[] = {
    seg0_3218,seg1_3218,seg2_3218,seg3_3218,seg4_3218,seg5_3218,
    seg6_3218,seg7_3218,
    NULL,
};
static int char_p3218[] = {
    XtNumber(seg0_3218),XtNumber(seg1_3218),XtNumber(seg2_3218),
    XtNumber(seg3_3218),XtNumber(seg4_3218),XtNumber(seg5_3218),
    XtNumber(seg6_3218),XtNumber(seg7_3218),
};
static XPoint seg0_3219[] = {
    {-2,-16},{-2,13},
};
static XPoint seg1_3219[] = {
    {2,-16},{2,13},
};
static XPoint seg2_3219[] = {
    {6,-7},{6,-8},{5,-8},{5,-6},{7,-6},{7,-8},{6,-10},{5,-11},
    {2,-12},{-2,-12},{-5,-11},{-7,-9},{-7,-6},{-6,-4},{-3,-2},
    {3,0},{5,1},{6,3},{6,6},{5,8},
};
static XPoint seg3_3219[] = {
    {-6,-6},{-5,-4},{-3,-3},{3,-1},{5,0},{6,2},
};
static XPoint seg4_3219[] = {
    {-5,-11},{-6,-9},{-6,-7},{-5,-5},{-3,-4},{3,-2},{6,0},{7,2},
    {7,5},{6,7},{5,8},{2,9},{-2,9},{-5,8},{-6,7},{-7,5},{-7,3},
    {-5,3},{-5,5},{-6,5},{-6,4},
};
static XPoint *char3219[] = {
    seg0_3219,seg1_3219,seg2_3219,seg3_3219,seg4_3219,
    NULL,
};
static int char_p3219[] = {
    XtNumber(seg0_3219),XtNumber(seg1_3219),XtNumber(seg2_3219),
    XtNumber(seg3_3219),XtNumber(seg4_3219),
};
```

```
static XPoint seg0_3220[] = {
    {9,-16},{-9,16},{-8,16},
};
static XPoint seg1_3220[] = {
    {9,-16},{10,-16},{-8,16},
};
static XPoint *char3220[] = {
    seg0_3220,seg1_3220,
    NULL,
};
static int char_p3220[] = {
    XtNumber(seg0_3220),XtNumber(seg1_3220),
};
static XPoint seg0_3221[] = {
    {3,-16},{1,-14},{-1,-11},{-3,-7},{-4,-2},
    {-4,2},{-3,7},{-1,11},{1,14},{3,16},
};
static XPoint seg1_3221[] = {
    {-1,-10},{-2,-7},{-3,-3},{-3,3},{-2,7},{-1,10},
};
static XPoint seg2_3221[] = {
    {1,-14},{0,-12},{-1,-9},{-2,-3},{-2,3},{-1,9},{0,12},{1,14},
};
static XPoint *char3221[] = {
    seg0_3221,seg1_3221,seg2_3221,
    NULL,
};
static int char_p3221[] = {
    XtNumber(seg0_3221),XtNumber(seg1_3221),XtNumber(seg2_3221),
};
static XPoint seg0_3222[] = {
    {-3,-16},{-1,-14},{1,-11},{3,-7},{4,-2},{4,2},{3,7},{1,11},
    {-1,14}, {-3,16},
};
static XPoint seg1_3222[] = {
    {1,-10},{2,-7},{3,-3},{3,3},{2,7},{1,10},
};
static XPoint seg2_3222[] = {
    {-1,-14},{0,-12},{1,-9},{2,-3},{2,3},{1,9},{0,12},{-1,14},
};
static XPoint *char3222[] = {
    seg0_3222,seg1_3222,seg2_3222,
    NULL,
};
static int char_p3222[] = {
    XtNumber(seg0_3222),XtNumber(seg1_3222),XtNumber(seg2_3222),
};
static XPoint seg0_3223[] = {
    {0,-12},{-1,-11},{1,-1},{0,0},
};
static XPoint seg1_3223[] = {
    {0,-12},{0,0},
};
```

continues

Listing C.4 Continued

```
static XPoint seg2_3223[] = {
    {0,-12},{1,-11},{-1,-1},{0,0},
};
static XPoint seg3_3223[] = {
    {-5,-9},{-4,-9},{4,-3},{5,-3},
};
static XPoint seg4_3223[] = {
    {-5,-9},{5,-3},
};
static XPoint seg5_3223[] = {
    {-5,-9},{-5,-8},{5,-4},{5,-3},
};
static XPoint seg6_3223[] = {
    {5,-9},{4,-9},{-4,-3},{-5,-3},
};
static XPoint seg7_3223[] = {
    {5,-9},{-5,-3},
};
static XPoint seg8_3223[] = {
    {5,-9},{5,-8},{-5,-4},{-5,-3},
};
static XPoint *char3223[] = {
    seg0_3223,seg1_3223,seg2_3223,seg3_3223,seg4_3223,seg5_3223,
    seg6_3223,seg7_3223,seg8_3223,
    NULL,
};
static int char_p3223[] = {
    XtNumber(seg0_3223),XtNumber(seg1_3223),XtNumber(seg2_3223),
    XtNumber(seg3_3223),XtNumber(seg4_3223),XtNumber(seg5_3223),
    XtNumber(seg6_3223),XtNumber(seg7_3223),XtNumber(seg8_3223),
};
static XPoint seg0_3224[] = {
    {-8,-1},{9,-1},{9,0},
};
static XPoint seg1_3224[] = {
    {-8,-1},{-8,0},{9,0},
};
static XPoint *char3224[] = {
    seg0_3224,seg1_3224,
    NULL,
};
static int char_p3224[] = {
    XtNumber(seg0_3224),XtNumber(seg1_3224),
};
static XPoint seg0_3225[] = {
    {0,-9},{0,8},{1,8},
};
static XPoint seg1_3225[] = {
    {0,-9},{1,-9},{1,8},
};
```

```
static XPoint seg2_3225[] = {
    {-8,-1},{9,-1},{9,0},
};
static XPoint seg3_3225[] = {
    {-8,-1},{-8,0},{9,0},
};
static XPoint *char3225[] = {
    seg0_3225,seg1_3225,seg2_3225,seg3_3225,
    NULL,
};
static int char_p3225[] = {
    XtNumber(seg0_3225),XtNumber(seg1_3225),XtNumber(seg2_3225),
    XtNumber(seg3_3225),
};
static XPoint seg0_3226[] = {
    {-8,-5},{9,-5},{9,-4},
};
static XPoint seg1_3226[] = {
    {-8,-5},{-8,-4},{9,-4},
};
static XPoint seg2_3226[] = {
    {-8,3},{9,3},{9,4},
};
static XPoint seg3_3226[] = {
    {-8,3},{-8,4},{9,4},
};
static XPoint *char3226[] = {
    seg0_3226,seg1_3226,seg2_3226,seg3_3226,
    NULL,
};
static int char_p3226[] = {
    XtNumber(seg0_3226),XtNumber(seg1_3226),XtNumber(seg2_3226),
    XtNumber(seg3_3226),
};
static XPoint seg0_3228[] = {
    {-4,-12},{-5,-11},{-5,-5},
};
static XPoint seg1_3228[] = {
    {-4,-11},{-5,-5},
};
static XPoint seg2_3228[] = {
    {-4,-12},{-3,-11},{-5,-5},
};
static XPoint seg3_3228[] = {
    {5,-12},{4,-11},{4,-5},
};
static XPoint seg4_3228[] = {
    {5,-11},{4,-5},
};
static XPoint seg5_3228[] = {
    {5,-12},{6,-11},{4,-5},
};
```

continues

Listing C.4 Continued

```
static XPoint *char3228[] = {
    seg0_3228,seg1_3228,seg2_3228,seg3_3228,seg4_3228,seg5_3228,
    NULL,
};
static int char_p3228[] = {
    XtNumber(seg0_3228),XtNumber(seg1_3228),XtNumber(seg2_3228),
    XtNumber(seg3_3228),XtNumber(seg4_3228),XtNumber(seg5_3228),
};
static XPoint seg0_3229[] = {
    {-1,-12},{-3,-11},{-4,-9},{-4,-7},{-3,-5},{-1,-4},{1,-4},
    {3,-5},{4,-7},{4,-9},{3,-11},{1,-12},{-1,-12},
};
static XPoint seg1_3229[] = {
    {-1,-12},{-4,-9},{-3,-5},{1,-4},{4,-7},{3,-11},{-1,-12},
};
static XPoint seg2_3229[] = {
    {1,-12},{-3,-11},{-4,-7},{-1,-4},{3,-5},{4,-9},{1,-12},
};
static XPoint *char3229[] = {
    seg0_3229,seg1_3229,seg2_3229,
    NULL,
};
static int char_p3229[] = {
    XtNumber(seg0_3229),XtNumber(seg1_3229),XtNumber(seg2_3229),
};
static XPoint seg0_3249[] = {
    {-8,8},
};
static XPoint *char3249[] = {
    seg0_3249,
    NULL,
};
static int char_p3249[] = {
    XtNumber(seg0_3249),
};
static XPoint seg0_3250[] = {
    {2,-12},{-1,-11},{-3,-9},{-5,-6},{-6,-3},{-7,1},{-7,4},
    {-6,7},{-5,8},{-3,9},{-1,9},{2,8},{4,6},{6,3},{7,0},{8,-4},
    {8,-7},{7,-10},{6,-11},{4,-12},{2,-12},
};
static XPoint seg1_3250[] = {
    {-1,-10},{-3,-8},{-4,-6},{-5,-3},{-6,1},{-6,5},{-5,7},
};
static XPoint seg2_3250[] = {
    {2,7},{4,5},{5,3},{6,0},{7,-4},{7,-8},{6,-10},
};
static XPoint seg3_3250[] = {
    {2,-12},{0,-11},{-2,-8},{-3,-6},{-4,-3},{-5,1},{-5,6},
    {-4,8},{-3,9},
};
```

```
static XPoint seg4_3250[] = {
    {-1,9},{1,8},{3,5},{4,3},{5,0},{6,-4},{6,-9},{5,-11},
    {4,-12},
};
static XPoint *char3250[] = {
    seg0_3250,seg1_3250,seg2_3250,seg3_3250,seg4_3250,
    NULL,
};
static int char_p3250[] = {
    XtNumber(seg0_3250),XtNumber(seg1_3250),XtNumber(seg2_3250),
    XtNumber(seg3_3250),XtNumber(seg4_3250),
};
static XPoint seg0_3251[] = {
    {2,-8},{-3,9},{-1,9},
};
static XPoint seg1_3251[] = {
    {5,-12},{3,-8},{-2,9},
};
static XPoint seg2_3251[] = {
    {5,-12},{-1,9},
};
static XPoint seg3_3251[] = {
    {5,-12},{2,-9},{-1,-7},{-3,-6},
};
static XPoint seg4_3251[] = {
    {2,-8},{0,-7},{-3,-6},
};
static XPoint *char3251[] = {
    seg0_3251,seg1_3251,seg2_3251,seg3_3251,seg4_3251,
    NULL,
};
static int char_p3251[] = {
    XtNumber(seg0_3251),XtNumber(seg1_3251),XtNumber(seg2_3251),
    XtNumber(seg3_3251),XtNumber(seg4_3251),
};
static XPoint seg0_3252[] = {
    {-3,-7},{-3,-8},{-2,-8},{-2,-6},{-4,-6},{-4,-8},{-3,-10},
    {-2,-11},{1,-12},{4,-12},{7,-11},{8,-9},{8,-7},{7,-5},
    {5,-3},{-5,3},{-7,5},{-9,9},
};
static XPoint seg1_3252[] = {
    {6,-11},{7,-9},{7,-7},{6,-5},{4,-3},{1,-1},
};
static XPoint seg2_3252[] = {
    {4,-12},{5,-11},{6,-9},{6,-7},{5,-5},{3,-3},{-5,3},
};
static XPoint seg3_3252[] = {
    {-8,7},{-7,6},{-5,6},{0,7},{5,7},{6,6},
};
static XPoint seg4_3252[] = {
    {-5,6},{0,8},{5,8},
};
```

continues

Listing C.4 Continued

```
static XPoint seg5_3252[] = {
    {-5,6},{0,9},{3,9},{5,8},{6,6},{6,5},
};
static XPoint *char3252[] = {
    seg0_3252,seg1_3252,seg2_3252,seg3_3252,seg4_3252,seg5_3252,
    NULL,
};
static int char_p3252[] = {
    XtNumber(seg0_3252),XtNumber(seg1_3252),XtNumber(seg2_3252),
    XtNumber(seg3_3252),XtNumber(seg4_3252),XtNumber(seg5_3252),
};
static XPoint seg0_3253[] = {
    {-3,-7},{-3,-8},{-2,-8},{-2,-6},{-4,-6},{-4,-8},{-3,-10},
    {-2,-11},{1,-12},{4,-12},{7,-11},{8,-9},{8,-7},{7,-5},
    {6,-4},{4,-3},{1,-2},
};
static XPoint seg1_3253[] = {
    {6,-11},{7,-9},{7,-7},{6,-5},{5,-4},
};
static XPoint seg2_3253[] = {
    {4,-12},{5,-11},{6,-9},{6,-7},{5,-5},{3,-3},{1,-2},
};
static XPoint seg3_3253[] = {
    {-1,-2},{1,-2},{4,-1},{5,0},{6,2},{6,5},{5,7},{3,8},
    {0,9},{-3,9},{-6,8},{-7,7},{-8,5},{-8,3},{-6,3},{-6,5},
    {-7,5},{-7,4},
};
static XPoint seg4_3253[] = {
    {4,0},{5,2},{5,5},{4,7},
};
static XPoint seg5_3253[] = {
    {1,-2},{3,-1},{4,1},{4,5},{3,7},{2,8},{0,9},
};
static XPoint *char3253[] = {
    seg0_3253,seg1_3253,seg2_3253,seg3_3253,seg4_3253,seg5_3253,
    NULL,
};
static int char_p3253[] = {
    XtNumber(seg0_3253),XtNumber(seg1_3253),XtNumber(seg2_3253),
    XtNumber(seg3_3253),XtNumber(seg4_3253),XtNumber(seg5_3253),
};
static XPoint seg0_3254[] = {
    {5,-8},
    {0,9},{2,9},
};
static XPoint seg1_3254[] = {
    {8,-12},{6,-8},{1,9},
};
static XPoint seg2_3254[] = {
    {8,-12},{2,9},
};
```

```
static XPoint seg3_3254[] = {
    {8,-12},{-8,3},{8,3},
};
static XPoint *char3254[] = {
    seg0_3254,seg1_3254,seg2_3254,seg3_3254,
    NULL,
};
static int char_p3254[] = {
    XtNumber(seg0_3254),XtNumber(seg1_3254),XtNumber(seg2_3254),
    XtNumber(seg3_3254),
};
static XPoint seg0_3255[] = {
    {-1,-12},{-6,-2},
};
static XPoint seg1_3255[] = {
    {-1,-12},{9,-12},
};
static XPoint seg2_3255[] = {
    {-1,-11},{7,-11},
};
static XPoint seg3_3255[] = {
    {-2,-10},{3,-10},{7,-11},{9,-12},
};
static XPoint seg4_3255[] = {
    {-6,-2},{-5,-3},{-2,-4},{1,-4},{4,-3},{5,-2},{6,0},{6,3},
    {5,6},{3,8},{-1,9},{-4,9},{-6,8},{-7,7},{-8,5},{-8,3},
    {-6,3},{-6,5},{-7,5},{-7,4},
};
static XPoint seg5_3255[] = {
    {4,-2},{5,0},{5,3},{4,6},{2,8},
};
static XPoint seg6_3255[] = {
    {1,-4},{3,-3},{4,-1},{4,3},{3,6},{1,8},{-1,9},
};
static XPoint *char3255[] = {
    seg0_3255,seg1_3255,seg2_3255,seg3_3255,seg4_3255,seg5_3255,
    seg6_3255,
    NULL,
};
static int char_p3255[] = {
    XtNumber(seg0_3255),XtNumber(seg1_3255),XtNumber(seg2_3255),
    XtNumber(seg3_3255),XtNumber(seg4_3255),XtNumber(seg5_3255),
    XtNumber(seg6_3255),
};
static XPoint seg0_3256[] = {
    {7,-8},{7,-9},{6,-9},{6,-7},{8,-7},{8,-9},{7,-11},{5,-12},
    {2,-12},{-1,-11},{-3,-9},{-5,-6},{-6,-3},{-7,1},{-7,4},
    {-6,7},{-5,8},{-3,9},{0,9},{3,8},{5,6},{6,4},{6,1},{5,-1},
    {4,-2},{2,-3},{-1,-3},{-3,-2},{-4,-1},{-5,1},
};
static XPoint seg1_3256[] = {
    {-2,-9},{-4,-6},{-5,-3},{-6,1},{-6,5},{-5,7},
};
```

continues

Listing C.4 Continued

```
static XPoint seg2_3256[] = {
    {4,6},{5,4},{5,1},{4,-1},
};
static XPoint seg3_3256[] = {
    {2,-12},{0,-11},{-2,-8},{-3,-6},{-4,-3},{-5,1},{-5,6},
    {-4,8},{-3,9},
};
static XPoint seg4_3256[] = {
    {0,9},{2,8},{3,7},{4,4},{4,0},{3,-2},{2,-3},
};
static XPoint *char3256[] = {
    seg0_3256,seg1_3256,seg2_3256,seg3_3256,seg4_3256,
    NULL,
};
static int char_p3256[] = {
    XtNumber(seg0_3256),XtNumber(seg1_3256),XtNumber(seg2_3256),
    XtNumber(seg3_3256),XtNumber(seg4_3256),
};
static XPoint seg0_3257[] = {
    {-4,-12},{-6,-6},
};
static XPoint seg1_3257[] = {
    {9,-12},{8,-9},{6,-6},{2,-1},{0,2},{-1,5},{-2,9},
};
static XPoint seg2_3257[] = {
    {0,1},{-2,5},{-3,9},
};
static XPoint seg3_3257[] = {
    {6,-6},{0,0},{-2,3},{-3,5},{-4,9},{-2,9},
};
static XPoint seg4_3257[] = {
    {-5,-9},{-2,-12},{0,-12},{5,-9},
};
static XPoint seg5_3257[] = {
    {-3,-11},{0,-11},{5,-9},
};
static XPoint seg6_3257[] = {
    {-5,-9},{-3,-10},{0,-10},{5,-9},{7,-9},{8,-10},{9,-12},
};
static XPoint *char3257[] = {
    seg0_3257,seg1_3257,seg2_3257,seg3_3257,seg4_3257,seg5_3257,
    seg6_3257,
    NULL,
};
static int char_p3257[] = {
    XtNumber(seg0_3257),XtNumber(seg1_3257),XtNumber(seg2_3257),
    XtNumber(seg3_3257),XtNumber(seg4_3257),XtNumber(seg5_3257),
    XtNumber(seg6_3257),
};
static XPoint seg0_3258[] = {
    {1,-12},{-2,-11},{-3,-10},{-4,-8},{-4,-5},{-3,-3},{-1,-2},
    {2,-2},{5,-3},{7,-4},{8,-6},{8,-9},{7,-11},{5,-12},{1,-12},
};
```

```
static XPoint seg1_3258[] = {
    {3,-12},{-2,-11},
};
static XPoint seg2_3258[] = {
    {-2,-10},{-3,-8},{-3,-4},{-2,-3},
};
static XPoint seg3_3258[] = {
    {-3,-3},{0,-2},
};
static XPoint seg4_3258[] = {
    {1,-2},{5,-3},
};
static XPoint seg5_3258[] = {
    {6,-4},{7,-6},{7,-9},{6,-11},
};
static XPoint seg6_3258[] = {
    {7,-11},{3,-12},
};
static XPoint seg7_3258[] = {
    {1,-12},{-1,-10},{-2,-8},{-2,-4},{-1,-2},
};
static XPoint seg8_3258[] = {
    {2,-2},{4,-3},{5,-4},{6,-6},{6,-10},{5,-12},
};
static XPoint seg9_3258[] = {
    {-1,-2},{-5,-1},{-7,1},{-8,3},{-8,6},{-7,8},{-4,9},{0,9},
    {4,8},{5,7},{6,5},{6,2},{5,0},{4,-1},{2,-2},
};
static XPoint seg10_3258[] = {
    {0,-2},{-5,-1},
};
static XPoint seg11_3258[] = {
    {-4,-1},{-6,1},{-7,3},{-7,6},{-6,8},
};
static XPoint seg12_3258[] = {
    {-7,8},{-2,9},{4,8},
};
static XPoint seg13_3258[] = {
    {4,7},{5,5},{5,2},{4,0},
};
static XPoint seg14_3258[] = {
    {4,-1},{1,-2},
};
static XPoint seg15_3258[] = {
    {-1,-2},{-3,-1},{-5,1},{-6,3},{-6,6},{-5,8},{-4,9},
};
static XPoint seg16_3258[] = {
    {0,9},{2,8},{3,7},{4,5},{4,1},{3,-1},{2,-2},
};
static XPoint *char3258[] = {
    seg0_3258,seg1_3258,seg2_3258,seg3_3258,seg4_3258,
    seg5_3258,seg6_3258,seg7_3258,seg8_3258,seg9_3258,
```

continues

Listing C.4 Continued

```
    seg10_3258,seg11_3258,seg12_3258,seg13_3258,
    seg14_3258,seg15_3258,seg16_3258,
    NULL,
};
static int char_p3258[] = {
    XtNumber(seg0_3258), XtNumber(seg1_3258), XtNumber(seg2_3258),
    XtNumber(seg3_3258), XtNumber(seg4_3258), XtNumber(seg5_3258),
    XtNumber(seg6_3258), XtNumber(seg7_3258), XtNumber(seg8_3258),
    XtNumber(seg9_3258), XtNumber(seg10_3258),XtNumber(seg11_3258),
    XtNumber(seg12_3258),XtNumber(seg13_3258),XtNumber(seg14_3258),
    XtNumber(seg15_3258),XtNumber(seg16_3258),
};
static XPoint seg0_3259[] = {
    {6,-4},{5,-2},{4,-1},{2,0},{-1,0},{-3,-1},{-4,-2},{-5,-4},
    {-5,-7},{-4,-9},{-2,-11},{1,-12},{4,-12},{6,-11},{7,-10},
    {8,-7},{8,-4},{7,0},{6,3},{4,6},{2,8},{-1,9},{-4,9},
    {-6,8},{-7,6},{-7,4},{-5,4},{-5,6},{-6,6},{-6,5},
};
static XPoint seg1_3259[] = {
    {-3,-2},{-4,-4},{-4,-7},{-3,-9},
};
static XPoint seg2_3259[] = {
    {6,-10},{7,-8},{7,-4},{6,0},{5,3},{3,6},
};
static XPoint seg3_3259[] = {
    {-1,0},{-2,-1},{-3,-3},{-3,-7},{-2,-10},{-1,-11},{1,-12},
};
static XPoint seg4_3259[] = {
    {4,-12},{5,-11},{6,-9},{6,-4},{5,0},{4,3},{3,5},{1,8},
    {-1,9},
};
static XPoint *char3259[] = {
    seg0_3259,seg1_3259,seg2_3259,seg3_3259,seg4_3259,
    NULL,
};
static int char_p3259[] = {
    XtNumber(seg0_3259),XtNumber(seg1_3259),XtNumber(seg2_3259),
    XtNumber(seg3_3259),XtNumber(seg4_3259),
};
static XPoint seg0_3260[] = {
    {-2,6},{-3,7},{-3,8},{-2,9},{-1,9},{0,8},{0,7},{-1,6},
    {-2,6},
};
static XPoint seg1_3260[] = {
    {-2,7},{-2,8},{-1,8},{-1,7},{-2,7},
};
static XPoint *char3260[] = {
    seg0_3260,seg1_3260,
    NULL,
};
```

```
static int char_p3260[] = {
    XtNumber(seg0_3260),XtNumber(seg1_3260),
};
static XPoint seg0_3261[] = {
    {-1,9},{-2,9},{-3,8},
    {-3,7},{-2,6},{-1,6},{0,7},{0,9},{-1,11},{-2,12},{-4,13},
};
static XPoint seg1_3261[] = {
    {-2,7},{-2,8},{-1,8},{-1,7},{-2,7},
};
static XPoint seg2_3261[] = {
    {-1,9},{-1,10},{-2,12},
};
static XPoint *char3261[] = {
    seg0_3261,seg1_3261,seg2_3261,
    NULL,
};
static int char_p3261[] = {
    XtNumber(seg0_3261),XtNumber(seg1_3261),XtNumber(seg2_3261),
};
static XPoint seg0_3262[] = {
    {1,-5},{0,-4},{0,-3},{1,-2},{2,-2},
    {3,-3},{3,-4},{2,-5},{1,-5},
};
static XPoint seg1_3262[] = {
    {1,-4},{1,-3},{2,-3},{2,-4},{1,-4},
};
static XPoint seg2_3262[] = {
    {-2,6},{-3,7},{-3,8},{-2,9},{-1,9},{0,8},{0,7},{-1,6},
    {-2,6},
};
static XPoint seg3_3262[] = {
    {-2,7},{-2,8},{-1,8},{-1,7},{-2,7},
};
static XPoint *char3262[] = {
    seg0_3262,seg1_3262,seg2_3262,seg3_3262,
    NULL,
};
static int char_p3262[] = {
    XtNumber(seg0_3262),XtNumber(seg1_3262),XtNumber(seg2_3262),
    XtNumber(seg3_3262),
};
static XPoint seg0_3263[] = {
    {1,-5},{0,-4},{0,-3},
    {1,-2},{2,-2},{3,-3},{3,-4},{2,-5},{1,-5},
};
static XPoint seg1_3263[] = {
    {1,-4},{1,-3},{2,-3},{2,-4},{1,-4},
};
static XPoint seg2_3263[] = {
    {-1,9},{-2,9},{-3,8},{-3,7},{-2,6},{-1,6},{0,7},{0,9},
    {-1,11},{-2,12},{-4,13},
};
```

continues

Listing C.4 Continued

```
static XPoint seg3_3263[] = {
    {-2,7},{-2,8},{-1,8},{-1,7},{-2,7},
};
static XPoint seg4_3263[] = {
    {-1,9},{-1,10},{-2,12},
};
static XPoint *char3263[] = {
    seg0_3263,seg1_3263,seg2_3263,seg3_3263,seg4_3263,
    NULL,
};
static int char_p3263[] = {
    XtNumber(seg0_3263),XtNumber(seg1_3263),XtNumber(seg2_3263),
    XtNumber(seg3_3263),XtNumber(seg4_3263),
};
static XPoint seg0_3264[] = {
    {4,-12},{3,-12},{2,-11},{0,2},
};
static XPoint seg1_3264[] = {
    {4,-11},{3,-11},{0,2},
};
static XPoint seg2_3264[] = {
    {4,-11},{4,-10},{0,2},
};
static XPoint seg3_3264[] = {
    {4,-12},{5,-11},{5,-10},{0,2},
};
static XPoint seg4_3264[] = {
    {-2,6},{-3,7},{-3,8},{-2,9},{-1,9},{0,8},{0,7},{-1,6},
    {-2,6},
};
static XPoint seg5_3264[] = {
    {-2,7},{-2,8},{-1,8},{-1,7},{-2,7},
};
static XPoint *char3264[] = {
    seg0_3264,seg1_3264,seg2_3264,seg3_3264,seg4_3264,
    seg5_3264,
    NULL,
};
static int char_p3264[] = {
    XtNumber(seg0_3264),XtNumber(seg1_3264),XtNumber(seg2_3264),
    XtNumber(seg3_3264),XtNumber(seg4_3264),XtNumber(seg5_3264),
};
static XPoint seg0_3265[] = {
    {-3,-7},{-3,-8},{-2,-8},{-2,-6},{-4,-6},{-4,-8},{-3,-10},
    {-2,-11},{1,-12},{5,-12},{8,-11},{9,-9},{9,-7},{8,-5},
    {7,-4},{5,-3},{1,-2},{-1,-1},{-1,1},{1,2},{2,2},
};
static XPoint seg1_3265[] = {
    {3,-12},{8,-11},
};
```

```
static XPoint seg2_3265[] = {
    {7,-11},{8,-9},{8,-7},{7,-5},{6,-4},{4,-3},
};
static XPoint seg3_3265[] = {
    {5,-12},{6,-11},{7,-9},{7,-7},{6,-5},{5,-4},{1,-2},{0,-1},
    {0,1},{1,2},
};
static XPoint seg4_3265[] = {
    {-2,6},{-3,7},{-3,8},{-2,9},{-1,9},{0,8},{0,7},{-1,6},
    {-2,6},
};
static XPoint seg5_3265[] = {
    {-2,7},{-2,8},{-1,8},{-1,7},{-2,7},
};
static XPoint *char3265[] = {
    seg0_3265,seg1_3265,seg2_3265,seg3_3265,seg4_3265,
    seg5_3265,
    NULL,
};
static int char_p3265[] = {
    XtNumber(seg0_3265),XtNumber(seg1_3265),XtNumber(seg2_3265),
    XtNumber(seg3_3265),XtNumber(seg4_3265),XtNumber(seg5_3265),
};
static XPoint seg0_3266[] = {
    {5,-12},{3,-11},{2,-10},
    {1,-8},{1,-6},{2,-5},{3,-5},{4,-6},{4,-7},{3,-8},{2,-8},
};
static XPoint seg1_3266[] = {
    {3,-11},{2,-9},{2,-8},
};
static XPoint seg2_3266[] = {
    {2,-7},{2,-6},{3,-6},{3,-7},{2,-7},
};
static XPoint *char3266[] = {
    seg0_3266,seg1_3266,seg2_3266,
    NULL,
};
static int char_p3266[] = {
    XtNumber(seg0_3266),XtNumber(seg1_3266),XtNumber(seg2_3266),
};
static XPoint seg0_3267[] = {
    {4,-9},{3,-9},{2,-10},
    {2,-11},{3,-12},{4,-12},{5,-11},{5,-9},{4,-7},{3,-6},{1,-5},
};
static XPoint seg1_3267[] = {
    {3,-11},{3,-10},{4,-10},{4,-11},{3,-11},
};
static XPoint seg2_3267[] = {
    {4,-9},{4,-8},{3,-6},
};
static XPoint *char3267[] = {
    seg0_3267,seg1_3267,seg2_3267,
    NULL,
};
```

continues

Listing C.4 Continued

```
static int char_p3267[] = {
    XtNumber(seg0_3267),XtNumber(seg1_3267),XtNumber(seg2_3267),
};
static XPoint seg0_3268[] = {
    {10,-3},{10,-4},{9,-4},{9,-2},{11,-2},{11,-4},{10,-5},{9,-5},
    {7,-4},{5,-2},{0,6},{-2,8},{-4,9},{-7,9},{-10,8},{-11,6},
    {-11,4},{-10,2},{-9,1},{-7,0},{-2,-2},{0,-3},
    {2,-5}, {3,-7},{3,-9},{2,-11},{0,-12},{-2,-11},{-3,-9},
    {-3,-6},{-2,0},{-1,3},{0,5},{2,8},{4,9},{6,9},{7,7},{7,6},
};
static XPoint seg1_3268[] = {
    {-6,9},{-10,8},
};
static XPoint seg2_3268[] = {
    {-9,8},{-10,6},{-10,4},{-9,2},{-8,1},{-6,0},
};
static XPoint seg3_3268[] = {
    {-2,-2},{-1,1},{2,7},{4,8},
};
static XPoint seg4_3268[] = {
    {-7,9},{-8,8},{-9,6},{-9,4},{-8,2},{-7,1},{-5,0},{0,-3},
};
static XPoint seg5_3268[] = {
    {-3,-6},{-2,-3},{-1,0},{1,4},{3,7},{5,8},{6,8},{7,7},
};
static XPoint *char3268[] = {
    seg0_3268,seg1_3268,seg2_3268,seg3_3268,seg4_3268,
    seg5_3268,
    NULL,
};
static int char_p3268[] = {
    XtNumber(seg0_3268),XtNumber(seg1_3268),XtNumber(seg2_3268),
    XtNumber(seg3_3268),XtNumber(seg4_3268),XtNumber(seg5_3268),
};
static XPoint seg0_3269[] = {
    {2,-16},{-6,13},
};
static XPoint seg1_3269[] = {
    {7,-16},{-1,13},
};
static XPoint seg2_3269[] = {
    {8,-7},{8,-8},{7,-8},{7,-6},{9,-6},{9,-8},{8,-10},{7,-11},
    {4,-12},{0,-12},{-3,-11},{-5,-9},{-5,-6},{-4,-4},{-2,-2},
    {4,1},{5,3},{5,6},{4,8},
};
static XPoint seg3_3269[] = {
    {-4,-6},{-3,-4},{4,0},{5,2},
};
```

```
static XPoint seg4_3269[] = {
    {-3,-11},{-4,-9},{-4,-7},{-3,-5},{3,-2},{5,0},{6,2},{6,5},
    {5,7},{4,8},{1,9},{-3,9},{-6,8},{-7,7},{-8,5},{-8,3},
    {-6,3},{-6,5},{-7,5},{-7,4},
};
static XPoint *char3269[] = {
    seg0_3269,seg1_3269,seg2_3269,seg3_3269,seg4_3269,
    NULL,
};
static int char_p3269[] = {
    XtNumber(seg0_3269),XtNumber(seg1_3269),XtNumber(seg2_3269),
    XtNumber(seg3_3269),XtNumber(seg4_3269),
};
static XPoint seg0_3270[] = {
    {13,-16},{-13,16},{-12,16},
};
static XPoint seg1_3270[] = {
    {13,-16},{14,-16},{-12,16},
};
static XPoint *char3270[] = {
    seg0_3270,seg1_3270,
    NULL,
};
static int char_p3270[] = {
    XtNumber(seg0_3270),XtNumber(seg1_3270),
};
static XPoint seg0_3271[] = {
    {8,-16},{6,-15},{3,-13},{0,-10},{-2,-7},
    {-4,-3},{-5,1},{-5,6},{-4,10},{-3,13},{-1,16},
};
static XPoint seg1_3271[] = {
    {1,-10},{-1,-7},{-3,-3},{-4,2},{-4,10},
};
static XPoint seg2_3271[] = {
    {8,-16},{5,-14},{2,-11},{0,-8},{-1,-6},{-2,-3},{-3,1},
    {-4,10},
};
static XPoint seg3_3271[] = {
    {-4,2},{-3,11},{-2,14},{-1,16},
};
static XPoint *char3271[] = {
    seg0_3271,seg1_3271,seg2_3271,seg3_3271,
    NULL,
};
static int char_p3271[] = {
    XtNumber(seg0_3271),XtNumber(seg1_3271),
    XtNumber(seg2_3271),XtNumber(seg3_3271),
};
static XPoint seg0_3272[] = {
    {1,-16},{3,-13},{4,-10},{5,-6},
    {5,-1},{4,3},{2,7},{0,10},{-3,13},{-6,15},{-8,16},
};
```

continues

Listing C.4 Continued

```
static XPoint seg1_3272[] = {
    {4,-10},{4,-2},{3,3},{1,7},{-1,10},
};
static XPoint seg2_3272[] = {
    {1,-16},{2,-14},{3,-11},{4,-2},
};
static XPoint seg3_3272[] = {
    {4,-10},{3,-1},{2,3},{1,6},{0,8},{-2,11},{-5,14},{-8,16},
};
static XPoint *char3272[] = {
    seg0_3272,seg1_3272,seg2_3272,seg3_3272,
    NULL,
};
static int char_p3272[] = {
    XtNumber(seg0_3272),XtNumber(seg1_3272),
    XtNumber(seg2_3272),XtNumber(seg3_3272),
};
static XPoint seg0_3273[] = {
    {2,-12},{1,-11},{3,-1},{2,0},
};
static XPoint seg1_3273[] = {
    {2,-12},{2,0},
};
static XPoint seg2_3273[] = {
    {2,-12},{3,-11},{1,-1},{2,0},
};
static XPoint seg3_3273[] = {
    {-3,-9},{-2,-9},{6,-3},{7,-3},
};
static XPoint seg4_3273[] = {
    {-3,-9},{7,-3},
};
static XPoint seg5_3273[] = {
    {-3,-9},{-3,-8},{7,-4},{7,-3},
};
static XPoint seg6_3273[] = {
    {7,-9},{6,-9},{-2,-3},{-3,-3},
};
static XPoint seg7_3273[] = {
    {7,-9},{-3,-3},
};
static XPoint seg8_3273[] = {
    {7,-9},{7,-8},{-3,-4},{-3,-3},
};
static XPoint *char3273[] = {
    seg0_3273,seg1_3273,seg2_3273,seg3_3273,
    seg4_3273,seg5_3273,seg6_3273,seg7_3273,seg8_3273,
    NULL,
};
```

```
static int char_p3273[] = {
    XtNumber(seg0_3273),XtNumber(seg1_3273),XtNumber(seg2_3273),
    XtNumber(seg3_3273),XtNumber(seg4_3273),XtNumber(seg5_3273),
    XtNumber(seg6_3273),XtNumber(seg7_3273),XtNumber(seg8_3273),
};
static XPoint seg0_3274[] = {
    {-8,-1},{9,-1},{9,0},
};
static XPoint seg1_3274[] = {
    {-8,-1},{-8,0},{9,0},
};
static XPoint *char3274[] = {
    seg0_3274,seg1_3274,
    NULL,
};
static int char_p3274[] = {
    XtNumber(seg0_3274),XtNumber(seg1_3274),
};
static XPoint seg0_3275[] = {
    {0,-9},{0,8},{1,8},
};
static XPoint seg1_3275[] = {
    {0,-9},{1,-9},{1,8},
};
static XPoint seg2_3275[] = {
    {-8,-1},{9,-1},{9,0},
};
static XPoint seg3_3275[] = {
    {-8,-1},{-8,0},{9,0},
};
static XPoint *char3275[] = {
    seg0_3275,seg1_3275,seg2_3275,seg3_3275,
    NULL,
};
static int char_p3275[] = {
    XtNumber(seg0_3275),XtNumber(seg1_3275),
    XtNumber(seg2_3275),XtNumber(seg3_3275),
};
static XPoint seg0_3276[] = {
    {-8,-5},{9,-5},{9,-4},
};
static XPoint seg1_3276[] = {
    {-8,-5},{-8,-4},{9,-4},
};
static XPoint seg2_3276[] = {
    {-8,3},{9,3},{9,4},
};
static XPoint seg3_3276[] = {
    {-8,3},{-8,4},{9,4},
};
static XPoint *char3276[] = {
    seg0_3276,seg1_3276,seg2_3276,seg3_3276,
    NULL,
};
```

continues

Listing C.4 Continued

```
static int char_p3276[] = {
    XtNumber(seg0_3276),XtNumber(seg1_3276),
    XtNumber(seg2_3276),XtNumber(seg3_3276),
};
static XPoint seg0_3278[] = {
    {-2,-12},{-3,-11},{-5,-5},
};
static XPoint seg1_3278[] = {
    {-2,-11},{-5,-5},
};
static XPoint seg2_3278[] = {
    {-2,-12},{-1,-11},{-5,-5},
};
static XPoint seg3_3278[] = {
    {8,-12},{7,-11},{5,-5},
};
static XPoint seg4_3278[] = {
    {8,-11},{5,-5},
};
static XPoint seg5_3278[] = {
    {8,-12},{9,-11},{5,-5},
};
static XPoint *char3278[] = {
    seg0_3278,seg1_3278,seg2_3278,seg3_3278,seg4_3278,
    seg5_3278,
    NULL,
};
static int char_p3278[] = {
    XtNumber(seg0_3278),XtNumber(seg1_3278),XtNumber(seg2_3278),
    XtNumber(seg3_3278),XtNumber(seg4_3278),XtNumber(seg5_3278),
};
static XPoint seg0_3279[] = {
    {1,-12},{-1,-11},{-2,-9},{-2,-7},{-1,-5},{1,-4},
    {3,-4},{5,-5},{6,-7},{6,-9},{5,-11},{3,-12},{1,-12},
};
static XPoint seg1_3279[] = {
    {1,-12},{-2,-9},{-1,-5},{3,-4},{6,-7},{5,-11},{1,-12},
};
static XPoint seg2_3279[] = {
    {3,-12},{-1,-11},{-2,-7},{1,-4},{5,-5},{6,-9},{3,-12},
};
static XPoint *char3279[] = {
   seg0_3279,seg1_3279,seg2_3279,
   NULL,
};
static int char_p3279[] = {
   XtNumber(seg0_3279),XtNumber(seg1_3279),XtNumber(seg2_3279),
};

/*
 * a full char set (and charP's)
 */
```

```
int * VCharPSet[] = {
    char_p501,  char_p502,  char_p503,  char_p504,
    char_p505,  char_p506,  char_p507,  char_p508,
    char_p509,  char_p510,  char_p511,  char_p512,
    char_p513,  char_p514,  char_p515,  char_p516,
    char_p517,  char_p518,  char_p519,  char_p520,
    char_p521,  char_p522,  char_p523,  char_p524,
    char_p525,  char_p526,  char_p551,  char_p552,
    char_p553,  char_p554,  char_p555,  char_p556,
    char_p557,  char_p558,  char_p559,  char_p560,
    char_p561,  char_p562,  char_p563,  char_p564,
    char_p565,  char_p566,  char_p567,  char_p568,
    char_p569,  char_p570,  char_p571,  char_p572,
    char_p573,  char_p574,  char_p575,  char_p576,
    char_p601,  char_p602,  char_p603,  char_p604,
    char_p605,  char_p606,  char_p607,  char_p608,
    char_p609,  char_p610,  char_p611,  char_p612,
    char_p613,  char_p614,  char_p615,  char_p616,
    char_p617,  char_p618,  char_p619,  char_p620,
    char_p621,  char_p622,  char_p623,  char_p624,
    char_p625,  char_p626,  char_p651,  char_p652,
    char_p653,  char_p654,  char_p655,  char_p656,
    char_p657,  char_p658,  char_p659,  char_p660,
    char_p661,  char_p662,  char_p663,  char_p664,
    char_p665,  char_p666,  char_p667,  char_p668,
    char_p669,  char_p670,  char_p671,  char_p672,
    char_p673,  char_p674,  char_p675,  char_p676,
    char_p699,  char_p700,  char_p701,  char_p702,
    char_p703,  char_p704,  char_p705,  char_p706,
    char_p707,  char_p708,  char_p709,  char_p710,
    char_p711,  char_p712,  char_p713,  char_p714,
    char_p715,  char_p717,  char_p718,  char_p719,
    char_p720,  char_p721,  char_p722,  char_p723,
    char_p724,  char_p725,  char_p726,  char_p730,
    char_p731,  char_p733,  char_p734,  char_p804,
    char_p834,  char_p840,  char_p844,  char_p845,
    char_p847,  char_p850,  char_p855,  char_p866,
    char_p999,  char_p2219, char_p2223, char_p2224,
    char_p2225, char_p2226, char_p2229, char_p2241,
    char_p2242, char_p2246, char_p2262, char_p2271,
    char_p2273, char_p2275, char_p2750, char_p2751,
    char_p2752, char_p2753, char_p2754, char_p2755,
    char_p2756, char_p2757, char_p2758, char_p2759,
    char_p2761, char_p2762, char_p2763, char_p2764,
    char_p2765, char_p2766, char_p2767, char_p2768,
    char_p2769, char_p2770, char_p2771, char_p2772,
    char_p2773, char_p2778, char_p3001, char_p3002,
    char_p3003, char_p3004, char_p3005, char_p3006,
    char_p3007, char_p3008, char_p3009, char_p3010,
    char_p3011, char_p3012, char_p3013, char_p3014,
    char_p3015, char_p3016, char_p3017, char_p3018,
    char_p3019, char_p3020, char_p3021, char_p3022,
    char_p3023, char_p3024, char_p3025, char_p3026,
```

continues

Listing C.4 Continued

```
   char_p3051, char_p3052, char_p3053, char_p3054,
   char_p3055, char_p3056, char_p3057, char_p3058,
   char_p3059, char_p3060, char_p3061, char_p3062,
   char_p3063, char_p3064, char_p3065, char_p3066,
   char_p3067, char_p3068, char_p3069, char_p3070,
   char_p3071, char_p3072, char_p3073, char_p3074,
   char_p3075, char_p3076, char_p3101, char_p3102,
   char_p3103, char_p3104, char_p3105, char_p3106,
   char_p3107, char_p3108, char_p3109, char_p3110,
   char_p3111, char_p3112, char_p3113, char_p3114,
   char_p3115, char_p3116, char_p3117, char_p3118,
   char_p3119, char_p3120, char_p3121, char_p3122,
   char_p3123, char_p3124, char_p3125, char_p3126,
   char_p3151, char_p3152, char_p3153, char_p3154,
   char_p3155, char_p3156, char_p3157, char_p3158,
   char_p3159, char_p3160, char_p3161, char_p3162,
   char_p3163, char_p3164, char_p3165, char_p3166,
   char_p3167, char_p3168, char_p3169, char_p3170,
   char_p3171, char_p3172, char_p3173, char_p3174,
   char_p3175, char_p3176, char_p3199, char_p3200,
   char_p3201, char_p3202, char_p3203, char_p3204,
   char_p3205, char_p3206, char_p3207, char_p3208,
   char_p3209, char_p3210, char_p3211, char_p3212,
   char_p3213, char_p3214, char_p3215, char_p3216,
   char_p3217, char_p3218, char_p3219, char_p3220,
   char_p3221, char_p3222, char_p3223, char_p3224,
   char_p3225, char_p3226, char_p3228, char_p3229,
   char_p3249, char_p3250, char_p3251, char_p3252,
   char_p3253, char_p3254, char_p3255, char_p3256,
   char_p3257, char_p3258, char_p3259, char_p3260,
   char_p3261, char_p3262, char_p3263, char_p3264,
   char_p3265, char_p3266, char_p3267, char_p3268,
   char_p3269, char_p3270, char_p3271, char_p3272,
   char_p3273, char_p3274, char_p3275, char_p3276,
   char_p3278, char_p3279 };
#endif /* VEC_FONTS_INC_H */
```

Index