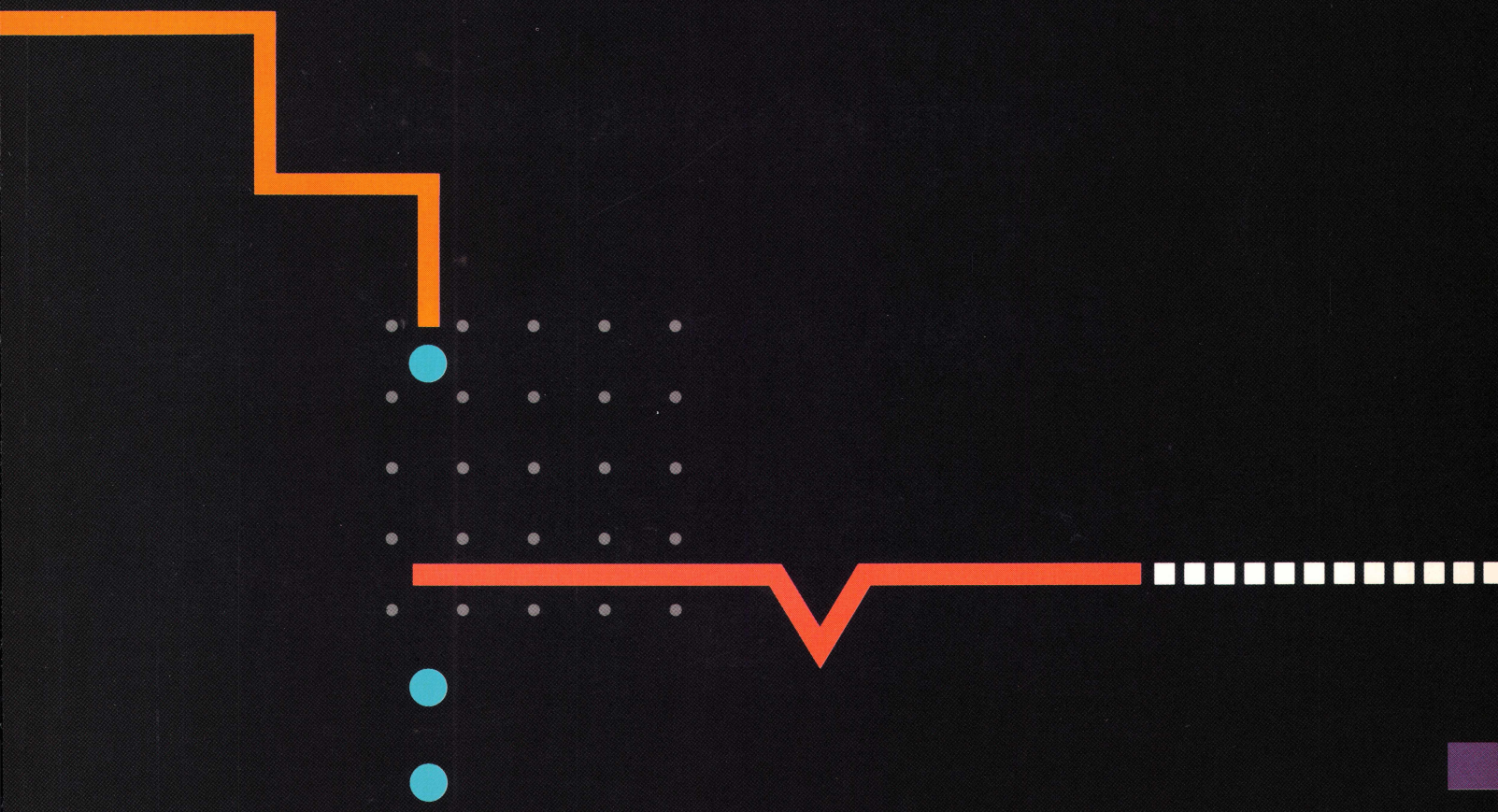
AMIGA®

User Interface Style Guide



AMIGA TECHNICAL REFERENCE SERIES

COMMODORE-AMIGA, INCORPORATED

AMIGA®
User Interface Style Guide

Commodore-Amiga, Incorporated

AMIGA TECHNICAL REFERENCE SERIES

Addison-Wesley Publishing Company, Inc.

Reading, Massachusetts Menlo Park, California New York
Don Mills, Ontario Wokingham, England Amsterdam Bonn Madrid
Mexico City Milan Paris San Juan Singapore Sydney Tokyo

Writers/Editors:
Dan Baker, Mark Green, and David Junod

Contributors:
Richard Buck, Peter Cherna, Eric Cotton, Andy Finkel, Darren Greenwald, Alan Havemose, William Hawes, Paul Higginbottom, Ross Hippely, David Joiner, Willy Langeveld, Jim Mackraz, Keith Masavage, Bryce Nesbitt, Mark Ricci, Carolyn Scheppner, Jeff Scherb, Carol Sullivan, Martin Taillefer, Isabelle Vesey, Deanna Vincent, and Marvin Weinstein.

Designer:
Mark Green

Copy editor:
Susan West

Cover designer:
Hannus Design Associates

Mouse illustrator:
Wilson Harp

Production Notes:
With the exception of the outside cover, this book was desktop published on an Amiga 2500. PostScript files were output to a 2470-dpi lasersetter as film. Software used in this process included: AmigaDOS Release 2, Oxxi's *TurboText*, Carolyn Scheppner's *ScreenSave* v36.11, Electronic Arts' *DeluxePaintIII*, ASDG's *The Art Department*, and Gold Disk's *Professional Page* v1.31. Lasersetter output by Graphics Standard.

Commodore item number: 368193-01

First printing, February 1991

1 2 3 4 5 6 7 8 9-AL-95 94 93 92 91

Library of Congress Cataloging-in-Publication Data

Amiga user interface style guide / Commodore-Amiga, Incorporated.
p. cm. – (Amiga technical reference series)
Includes index.
ISBN 0-201-57757-7
1. Amiga (Computer) 2. User Interfaces (Computer systems)
I. Commodore-Amiga, Inc. II. Series.
QA76.8.A46A45 1991 90-23757
005.265–dc20 CIP

CONTENTS

Preface

Like any written work with a distribution wider than a personal letter, this style guide attempts to be many things to many readers.

After much deliberation we developed the following profiles of the average reader: a current Amiga developer working alone or with one partner; a developer from another platform who would like to develop for the Amiga; a first-time developer; a graphic artist designing a user interface for a developer; a team of developers working for a medium-sized company . . . the list goes on.

So our intention was to write a manual that introduced the Amiga from basics – in terms a non-technical reader could understand. The GUI sections were especially targeted for the layman. Other sections, such as the ARexx chapter, were structured more like reference guides since they will likely be used by readers with more of a technical background.

Release 2 Assumed

This book was written with Release 2 of the Amiga operating system in mind. All functions, examples and elements herein refer to Release 2.

The Gender Question

It's also worth noting that, no, we did not make mistakes in some places and insert feminine pronouns where we meant to put masculine.

English doesn't have a good, genderless, third-person pronoun and it's very awkward to avoid using one. So instead of just using the masculine throughout or alternating chapters with masculine/feminine/masculine/etc., we've changed

or alternating chapters with masculine/feminine/masculine/etc., we've changed genders at random. That way it best mimics reality where some of your users will be men and others will be women.

If you find yourself being startled by this shift – maybe you needed to be. No business can afford to alienate a customer by being unaware of her.

Amiga Mail Updates

Of course this manual, especially in its first edition, isn't the final word on style for Amiga applications. Interim updates may be published in *Amiga Mail*, the bi-monthly newsletter for Amiga developers.

Anyone may subscribe to *Amiga Mail*. If you live in North America and want more subscription information, send a self-addressed stamped envelope to:

CATS-Information
1200 Wilson Drive
West Chester, PA 19380-4231

Elsewhere, write to your local Commodore office. See Appendix B in the back of this manual for addresses. ▲

chapter one
INTRODUCTION

The most we can hope for
is that the oftener things are found together,
the more probable it becomes
that they will be found together another time,
and that,
if they have been found together often enough,
the probability will amount almost *to certainty.*
- Bertrand Russell

The purpose of this book is to describe how the user interface of a standard Amiga application should look and operate. It is intended to be read by current Amiga developers as well as developers who are considering writing and/or designing application software for the Amiga.

This book assumes some familiarity with computers and their interfaces in general, and with the Amiga's graphic user interface (GUI) in particular, but, for the most part, you do not have to be a programmer to understand the material.

Only the behavioral guidelines for an Amiga application are presented here. The details of how to implement them are covered in other volumes of the Amiga Reference Series.

WHAT'S IN THIS BOOK

This document provides the following information:

- the benefits of a standard user interface;
- an overview of the components of the Amiga user interface;
- specifications showing how to use the components of the Amiga user interface to create a standard Amiga application.

The Amiga hardware and system software provide the basic building blocks of the user interface: a mouse and pointer, windows and icons, menus and requesters and more. But it is your software that combines these elements and ultimately determines how the machine will be used.

Non-Stifling Standards

In one sense, this style guide can be considered a book of rules for you to follow when designing application software for the Amiga. It describes the best ways to combine and use elements of the Amiga system software to communicate with the user.

This manual describes the best ways to communicate with the user.

Unlike rule books such as a state's driving code or a company's employee handbook, the style guide's originators don't suggest penalties for violators. In fact, penalties of that sort would be counterproductive. The aim of this book is to establish standards for Amiga applications without stifling creativity. New versions of the Amiga and new types of applications will probably require refinement and expansion of these standards in the future.

That's not to say no penalties exist. In a free, competitive market the only real penalties are financial and self-inflicted. This book has been created under this premise: standardized software will be better for reasons described later in this chapter, and thus, in a competitive situation it should sell better.

In short, these standards were devised to improve your program and the Amiga platform in the eyes of the user.

A Point on the Horizon

No one expects every application to conform to every one of the rules and guidelines in this manual.

These rules have not been created in a vacuum. Many of the standards discussed in this manual have been culled from experience – experience gained through the multitude of Amiga applications released since the Amiga's inception. The writers of this document first looked at what has worked best in specific applications and then tried to transform that raw experience into a standard, efficient and accepted way to do things on the Amiga. The trick was to come up with ideas that worked well *and* would work well in a variety of situations.

The rules in this manual describe an idealized application not yet created.

Clearly there will always be exceptions to these rules. Even applications created by Commodore may not comply with every idea in this book.

The hope is that the idealized application described in this manual will be like a point on the horizon you keep in sight throughout the development of your very real program.

SOME PERSPECTIVE

The user is the most important part of any application. In the past, improving the speed or increasing the capacity of application software has been a major focus of computer programmers. The idea of emphasizing the user interface of a computer system over its performance and capacity is a relatively recent innovation in computer history.

Historically, most users of computer systems were technical people who could tolerate the exacting requirements of expensive computer hardware and their "unfriendly" application designs. Complex commands had to be remembered and typed into a terminal in the proper sequence and thus required these early computers to have a professional staff in order to run them. This made sense because the computing hardware of that era was much more expensive

than human labor. The user was there to serve the needs of these expensive machines.

The advent of microprocessors and the long-term trend to lower priced computers has changed all that. Today's computer user is trying to get work done with the machine, not become an expert on the machine itself. The user probably does not know much about computers and may not even know how to type. The human–machine interface has been reversed, and now computers must serve the needs of the user.

The User's Needs

The user's needs are simple:

- the operation of the software should be predictable;
- learning the software should be easy;
- the software's functions should be easily scannable;
- the software's operations should be consistent throughout tools, applications and similar objects;
- feedback should be provided to the user, so he knows what has happened and what he can do;
- the software should adapt to the user's level of experience.

The user needs: predictability, intuitiveness, accessibility, consistency, feedback, adaptability, and simplicity.

Consistency

The user's needs listed above really all combine into one: consistency. If your interface is consistent with the model, it becomes predictable; if it can be scanned easily, it provides feedback; if it provides feedback in a reasonable manner, the user always feels comfortable and can master increasing levels of complexity.

Consistency in a user interface allows the user to apply previously learned knowledge to each new application. The user will spend less time figuring out how to get work done, and can therefore be more productive. Learning a new application is much easier if it works just like one the user already understands.

The Amiga is a multitasking computer that allows the user to run more than

one application at a time. This makes consistency even more important because the user can easily switch from one application to another. Consistency between applications allows the user to make this switch without having to make a "mental jump" between one way of working and another.

Following standards also makes new applications more familiar; thus the user will probably be less afraid of inadvertently wiping out data or making other non-recoverable mistakes.

Once again:

predictability,
intuitiveness,
accessibility,
consistency,
feedback,
adaptability,
and simplicity.

Filling the Needs

A graphic user interface (GUI) is currently the best method available for simplifying the user interface and meeting the needs of the user.

Among the first computers to use GUIs were the Xerox Star and Apple Lisa. Based on pioneering research performed at the Xerox Palo Alto Research Center in the 1970s, these computers allow the user to issue commands with a mouse and pointer. Resources represented by graphic icons can be activated by pointing to them with the mouse, and actions can be performed through mouse-activated menus.

GUIs provide immediate feedback and scanning ability, so users can tell what their options are and don't have to memorize commands. They allow for enormous growth and adaptability of an application, because levels of functions and commands can be buried yet still be graphically accessible. In short, they provide a user interface that lets the user operate, without learning, the computer – in much the same way that the average driver doesn't need to be versed in internal combustion.

By utilizing standardized tools and objects provided in a GUI, programmers are less likely to invent needlessly different ways of doing things. The interface becomes predictable and consistent.

The Amiga

The Commodore Amiga is a further refinement of this philosophy that puts

the needs of the user foremost. Like many other systems, the Amiga has a GUI with a mouse, pointer, windows and menus which makes it easier for users – especially beginners.

But the Amiga also offers two other built-in interfaces: a text-oriented interface, the Shell, and an inter-process scripting language, ARexx.

Together, these three interfaces provide a powerful and flexible environment for both the novice and the expert. In fact a philosophy for the design of the Amiga user interface might be "Power to the User."

DEVELOPER-ORIENTED BENEFITS TO STANDARDS

The obvious beneficiary of a good user interface design is the user. He can pick up a new application and learn how to use it, scan its functions to see what it does, apply old knowledge to cut the learning curve, and so on.

But a disciplined approach to designing and building an application also has enormous benefits for the software developer.

Some real market-oriented reasons exist for standards too.

Market Acceptance

Marketing is simplified when a product "belongs" in a comfortable environment, sharing features and tools with other popular applications. A user trying out a demo version of your software and your competitor's software in a store rarely has the time to learn a new way of doing things. A familiar environment and controls that work in a predictable manner will allow her to experience what you are really selling – how well your application does its job.

Coexistence

By following the recommended guidelines, applications acquire the ability to inter-operate, thereby increasing each application's value. When applications can inter-operate and share data as well as they can on the Amiga, users can combine off-the-shelf software packages to create custom solutions for the work they want to do with the Amiga.

Secondly, it is important that applications behave well in a multitasking environment. Sticking to the standards assures peaceful coexistence.

Easier Creation and Maintenance

Design is also simplified when you follow standards.

For one thing, Release 2 of the operating system features toolkits such as GadTools which provide you with pre-coded elements of the GUI. Even when elements aren't provided "prefab," standards allow you to spend less time devising and designing environments and more time working on the actual operations of your program. It also allows you more time for testing. For instance, if you are designing a CAD program, chances are that it is computer-aided design which really interests you. Instead of spending a lot of time working on gadgets to control your application, you could be spending more time working on the actual CAD operations.

Also, in the absence of any standards, progress is more difficult because each application will require an individual upgrade plan.

POWER TO THE USER

Clearly, this book was created to sell the idea of standards as much as to simply present those standards. But the reasons for those standards goes beyond an innate need for order originating from some corporate boardroom. The reasons set forth here have been set forth honestly in an attempt to provide more power to the user. This book intends to make clear what features of the user interface will help to make the interface simple – and predictable – and consistent – and adaptable – and intuitive. This in turn should improve the Amiga market for all of us. ▲

chapter two
SOME BASICS

One man's "magic" is another man's engineering.
"Supernatural" is a null word.
- Robert Heinlein

The Amiga is a true multitasking system with three built-in interfaces:

- a graphic user interface (GUI);
- a command line interface (the Shell);
- a scripting language that can handle inter-process communications (ARexx).

The GUI is the Amiga's default interface.

Of these three interfaces, the GUI dominates. By default, the Amiga presents the user with a graphic interface, Workbench, upon startup. Likewise, even a simple Amiga application will generally present a graphic interface of some sort; text editors, for instance, usually include mouse-driven gadgets and menus.

In addition to the GUI, users can control the Amiga through the Shell. The Shell is a text-based interface that preserves the best features of the "old way" of operating computers – by typing in commands. The Shell trades the GUI's ease of use for a finer level of control and greater power.

With Release 2 of the operating system came a third way of interfacing with the Amiga: inter-process communication (IPC) via ARexx. Simply put, ARexx is a scripting language, but it also acts as a central hub which applications can use to send data and commands to each other. ARexx allows software created by dif-

ferent companies to interact, letting the user create custom applications by integrating off-the-shelf software products. For example, with ARexx it is possible to set up a telecommunications package to dial an electronic bulletin board, download financial data, and then pass that data to a separate spreadsheet package for statistical analysis – without user intervention other than the original scripting. ARexx is based on REXX, an IPC language used in various forms across many platforms.

Support all three of the Amiga's interfaces.

Many users consider the choice of interfaces to be one of the best features of the Amiga. Not only does it offer the user the freedom to choose his favorite means of interacting with the Amiga and with your program, it's also an effective way to provide the right tool for the job and the user's level of expertise. For these reasons, your application should support all three interfaces in the manner described in this manual.

BASICS OF THE OPERATING SYSTEM

The following are basic concepts to keep in mind throughout your design work on the Amiga:

Design for the Novice

Conserve the user's energy for creative tasks.

Designing for the lowest common denominator does not mean your application is doomed to mediocrity. In most cases it just means pondering a little longer on how to solve a problem or present a procedure so that almost anyone can grasp it easily. Even power users appreciate it when you give them simple solutions for such things as hard disk installation or navigation through procedural steps. Conserve your user's energy for creative tasks.

Let the System Work for You

In general, if the system will do something for you, let it. For example, GadTools provides pre-programmed gadgets for use in your application. If one of these gadgets suits your needs, use it instead of coding your own. Two other examples of using this built-in support are:

- using Intuition (see page 13) to provide many functions of your application's GUI;
- using DOS ReadArgs to provide support for argument passing.

If the system will do something for you . . . let it.

The Amiga's Resources

Remember that the Amiga is a multitasking system. Any time you allocate a part of the system for your use, you prevent other tasks from using it.

Try your best to economize your use of shared resources. If possible, only obtain a resource when needed and free it as soon as you are done. This applies to just about every computing activity your application can perform. There are only limited amounts of RAM, serial ports, printers and disk drives for all applications to share.

Your application should provide controls that allow the user to temporarily suspend usage of any non-sharable resource it may use, such as the serial or parallel device. For example, a MIDI application should allow the user to suspend serial port usage so she can switch to a terminal package and download a new sample without having to exit the MIDI application.

Speech and Sound

Speech and sound can be excellent ways to provide feedback to the user, especially if the user is visually impaired or if your application includes lengthy processes during which the user might walk away from the system.

However, a significant number of users may be hearing impaired, so don't use sound and speech as the only cue. Use it in conjunction with other cues such

as position, size and rendering.

Sounds can be annoying. Provide a means for the user to turn off sound and speech.

Provide a means for the user to turn off sounds or speech.

Controls

It may be a good idea to provide additional controls so the user can modify the speech or sound in your program. Consider this if speech/sound is integral to the application. If they are secondary, you may just be over-complicating the controls in your program.

Closing Down the Workbench

If your application needs more memory than is available, it may attempt to close Workbench.

If your application does this, it should allow the user to re-enable Workbench from a menu item if more memory becomes available. A preference setting should also be made available so the user can specify that Workbench should always be closed (or not) on entry to the application.

If your application closes Workbench, it must reopen it again before exiting.

Defining Your Basename

A basename is a short, one-word name that you assign to your application. It's usually (and preferably) the same word as the name of your program's executable – the word a user types into the Shell in order to run your program.

All user-accessible names for your program should be built from the basename. These include such things as ARexx ports, public screen name, and configuration settings names.

For example, a database program called MondoBase may have the basename of MBase. Users running the program from the Shell would type MBase on the command line. The program may open on a public screen named *MBase.1* with

an ARexx port of the same name.

Basenames should be logical to the user, easily remembered and short. Making it too short, however, could raise the likelihood of conflicts with other programs. If MondoBase's basename were simply M, it could conflict with MusicBoard's basename.

BASICS OF THE GUI

For a system with three interfaces, a relatively large section of this book is devoted to the GUI. This isn't surprising when you think about the popularity of GUIs and the fact that a GUI designer needs to worry as much about form as function.

The Amiga GUI has two parts: Intuition and Workbench. As a developer, you will be more concerned with Intuition. It consists of the function libraries and tools you will use to create and run GUIs for your application.

Before talking specifically about the elements of the Amiga GUI (in the next five chapters), some basic elements and techniques need to be discussed.

Metaphor

One of the reasons for the success of GUIs is that they greatly reduce the need to memorize commands and command structures. One technique used to accomplish this is the use of metaphor – GUIs try to mimic the traditional work world in some way.

By presenting what can be frighteningly complex workings as familiar, real-world objects and activities, even novice users can understand and use a computer quickly. Hierarchical filing systems become drawers. RAM caches that spool to disk become clipboards. Pulling an ASCII clip from a buffer into a document becomes pasting.

To employ a metaphor in your application, look for some common physical system that is analogous to the behavior of the application.

For example, a 3-D modeling package could be designed to work on wire-frame objects as though they were actually physical objects. The user could rotate the object by "grabbing" a corner of the object and pulling just as one would grab the corner of a real object.

Of course, it is not always necessary or even desirable for every interface to be modeled after some physical object.

Object–Action

Most of the popular GUIs today encourage the use of object–action methodology. This simply means that first the user selects the thing he wants something done to, then he selects what he wants to do with it. For example, first he clicks on the object he wants deleted then he chooses the delete function, rather than choosing delete then clicking on the objects to be deleted.

This is done mainly to prevent modality. Modes are generally avoided in GUIs because they can easily become restrictive and confusing to the user. If the user were to select delete before selecting the object(s), he would be in delete mode and could accidentally delete something he wanted to save – especially if he were to go away and come back later. There is also the problem of how to indicate the last object to delete. In general, any mode is limiting and should be avoided if possible.

Focus

When a user is looking at a computer screen, he is most likely to concentrate his attention on one particular spot. In a word processor, he will most likely be looking at the cursor. Or in a CAD program, the focus may be the currently selected line or polygon. Very often, the pointer itself is the focus.

It's important to identify the user's focus because this can be a very effective

channel of information. For example, you can animate or change the appearance of the focus object to indicate some change in the program's state. The wait or busy pointer is an example of this. If the wait pointer's clock imagery appeared in the title bar instead of on the pointer, it would be far less effective.

Feedback

Users expect to see an immediate reaction to every action. Even if the action will take some time, they expect to see some indication of "I'm working on it" such as the normal pointer changing to the wait pointer.

Try to provide feedback through as many appropriate media as you can. For example, when a user drags a note in a music package, he could hear the pitch change as well as see the note move. After all, the visual graphic of the note is only a representation of the sound of the note – and that's what the user is really interested in.

When the user attempts an action, provide feedback.

It's usually easy to provide instantaneous feedback to simple actions, but for more complex or time-consuming actions, a surrogate feedback may be necessary. A familiar example of this is the way windows are dragged on the Amiga – you don't drag a window, you drag an outline of it. Another example of feedback on time-consuming activities is the word processor that refreshes just the line being worked on as the user is typing, and the whole display whenever there is some processing time to spare. This gives the illusion of responsiveness, which from the user's point of view is as good as the real thing.

Color

Color is a very powerful medium of communication. Be conscious of what message you are communicating.

Keep in mind that simple images with fewer colors usually work better than complicated colorful ones. When choosing default colors for an application, do not use intense combinations like blue/yellow, red/green, red/blue or

green/blue. Use subdued colors instead of fully saturated colors. Color can set the mood of your program and that mood should be one that the user can live with for extended periods of time.

Give Controls

For all the screens you set up within your application, provide the user with the ability to load, edit and save the color palette. When the user makes a change to the color palette, examine the new palette to assign the appropriate colors for rendering.

Color as a Cue

Don't use color as the sole cue – some users work in monochrome.

Be consistent when using color to indicate meaning. Window backgrounds should be in the background color, text in the text color and headings in highlighted text.

Beware of using color alone to convey information – especially in small areas of the display. Some users may work on a monochrome display. Keep in mind also that many people are color blind. For these reasons, try to reserve the use of color only as an extra cue, used in conjunction with other cues such as position, size and rendering.

Check Visibility

The Amiga's Preferences editors allow users to run their applications in two colors. Certain high-resolution monitors have only grey-scale displays. Because of these cases, if your application can open on Workbench or a public custom screen, you should check that your graphics and text can be read on a monochrome or grey-scale display.

Elements should be visible, should continue to provide visible feedback, and should be recognizable as the same element whether in color or monochrome. Basically, you should make sure that all your functionally significant colors contrast with adjacent functionally significant colors. It's more a matter of the color's brightness or luminance than the actual color itself.

Commodore's high-resolution grey-scale monitor (A2024), for instance, uses patterns made up of its four basic greys to make more shades when running in medium and low resolutions. At these resolutions, things look best if you choose a color that maps to a solid, not patterned, grey. But these colors are not obvious.

Here's a sample palette that works well on a grey-scale monitor.

One palette that produces solid greys follows:

Color 0:	Light Grey, RGB=10,8,6
Color 1:	Black, RGB=0,0,2
Color 2:	White, RGB=15,15,15
Color 3:	Dark Grey, RGB=7,7,9

Understandably, some applications that open only on a multicolor custom screen don't need to work in monochrome, but checking for at least grey-scale functionality can help your program's readability. In the magazine industry, for example, many art directors will transform color headlines to grey scale and check the contrast. If the headlines aren't readable while in greys, there is a contrast problem, and the art director will change the colors.

Fonts

Users can specify which font the system should use for the Workbench icon font, the screen font (the preferred system font) and the system default font (a default mono-space font). These are selected with the Font preference editor in Workbench's Prefs drawer.

Your application should try to adjust itself to whatever size the user may select. Menus, windows, requesters and gadgets must all be adjusted to allow the various text sizes to fit.

Keep in mind that the chosen font may be monospaced or proportionally spaced, and not every application can handle a proportional font. A spreadsheet, for example, depends on the font being monospaced to properly handle the columnar data it manipulates. If your application needs a monospaced font it

should honor whatever the user has chosen to be the default system font – that font is guaranteed to be monospaced.

If your application can handle a proportional font, however, it should honor whatever the user has chosen to be the screen font.

Internationalization

Since the Amiga is sold in a world market, internationalization is an important design factor in your application. One of the most important and fundamental issues to consider when thinking about internationalization is the difference in length of the text between English and any other language.

English is one of the most terse languages, and when converting English to, say, German the text is bound to grow 30–50 percent in size.

Commodore has an extensive distribution network for its hardware. If you want to sell your software internationally, it is best to build internationalization into your application from the start.

Another thing to keep in mind is the desirability of building a text table that is centrally located so that it can be adjusted by the user. Do not scatter embedded text throughout your application; consider putting it in a file instead. This way, the text can be easily localized by updating one central table, or even by loading a new text table from disk.

Don't make every text item adjustable. For instance, if your application incorporates a procedural language or ARexx keywords, do not allow these special words to be localized. To do so would severely hurt the ability to share scripts between international users.

If your application provides database-like operations that involve money, numbers, telephone numbers and addresses, remember that these sort of fields vary in usage from country to country. The table below lists some of the differences.

international formats

Language	Date	Time	24hr/12hr	Decimal	1000s Sep.
American English	mm/dd/yy	hh:mm:ss	12 hr	period	comma
Australian English	dd/mm/yy	hh:mm:ss	12 hr	period	comma
British English	dd/mm/yy	hh:mm:ss	12 hr	period	comma
Canadian French	dd/mm/yy	hh:mm:ss	24 hr	comma	[space]
Danish	dd/mm/yy	hh:mm:ss	24 hr	comma	period
Dutch	dd-mm-yy	hh:mm:ss	24 hr	comma	period
Finnish	dd.mm.yy	hh:mm:ss	24 hr	comma	[space]
Flemish	dd-mm-yy	hh:mm:ss	24 hr	comma	period
French	dd.mm.yy	hh:mm:ss	24 hr	comma	[space]
German	dd.mm.yy	hh:mm:ss	24 hr	comma	period
Italian	dd-mm-yy	hh:mm:ss	24 hr	comma	period
Norwegian	dd-mm-yy	hh:mm:ss	24 hr	comma	[space]
Portugese	yy/mm/dd	hh:mm:ss	24 hr	comma	[space]
Spanish	dd/mm/yy	hh:mm:ss	24 hr	comma	period
Swedish	yy-mm-dd	hh.mm.ss	24 hr	comma	[space]
Swiss French	dd.mm.yy	hh:mm:ss	24 hr	period	apostrophe
Swiss German	dd.mm.yy	hh:mm:ss	24 hr	period	apostrophe

The 3-D Look

Release 2 of the Amiga operating system features a three-dimensional look to its GUI. Whenever possible, elements are drawn so that light appears to come from the upper left of the display with shadows cast to the lower right. This gives the illusion of depth.

The outside border of a raised object has thin light lines forming the top and left sides; dark lines form the bottom and right sides. Reverse these colors for a

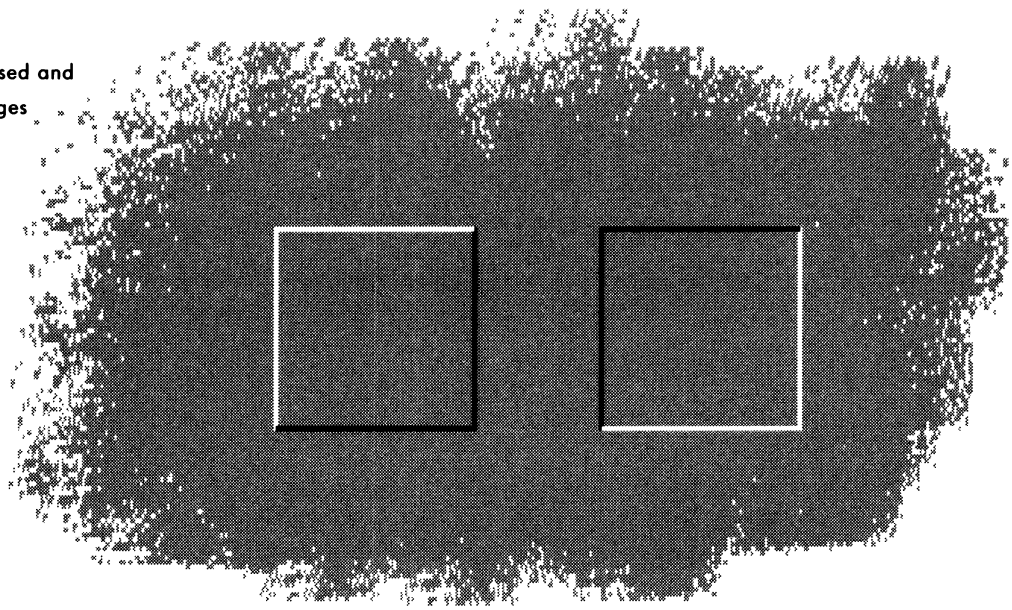


Fig. 2.1: Raised and recessed images

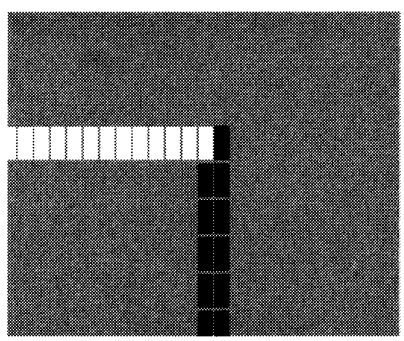


Fig. 2.2: Close-up view of 3-D box corner with a pixel grid superimposed.

recessed image.

Using this simulated depth, you can make images appear to be standing out or recessed in the display, thus giving the user a context cue at the same time.

If an icon or gadget appears raised, this means it is available for use or modifiable. If its default appearance is recessed, this means it is unmodifiable or for display purposes only.

Another context cue: when a user clicks on an icon or a button-like gadget, it should change from a raised image to a recessed, highlighted image. This goes along with the basic idea of providing feedback that the attempted action (clicking on an action gadget, for example) has worked.

Objects that highlight should show distinctly different imagery between highlighted and non-highlighted states. One convention used by Workbench is to display the image in complementary colors.

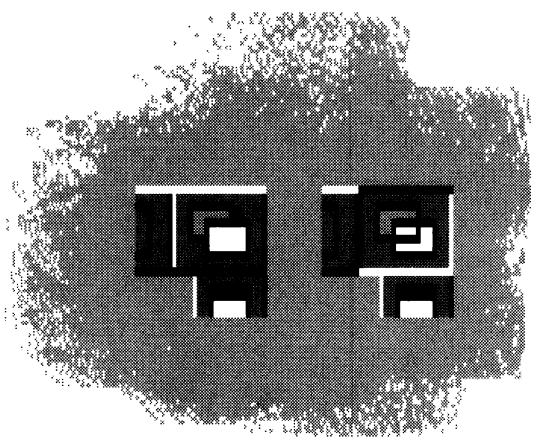
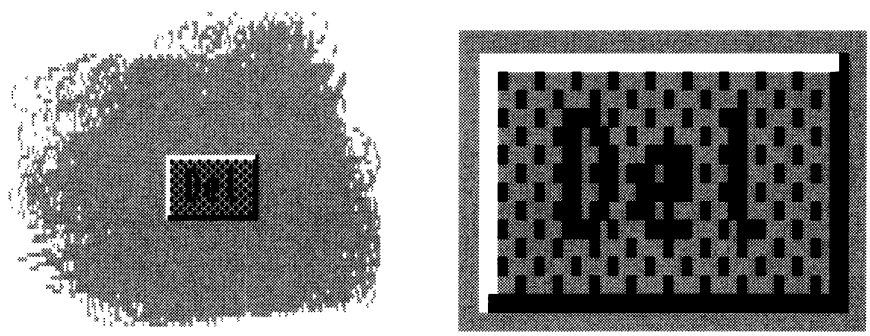


Fig. 2.3: Examples of non-highlighted and highlighted gadgets

Ghosting

When a control is unavailable for selection, it should be obviously unselectable. Do not allow the user to select something that does nothing in response. Intuition will automatically display a ghosting pattern – a grid of dots in the "shadow" color – over an unavailable control. If, however, you need to do it manually, follow the example shown below:

Fig. 2.4: A normal and enlarged view of a ghosted gadget.



GadTools

If you are developing for the Release 2 environment, you don't need to worry about incorporating the 3-D look into your gadgets – GadTools will do it for you. GadTools is a toolkit that supplies pre-programmed, standard application gadgets for use in your application.

Whenever possible you should let GadTools work for you. Even beyond the standardization of function it provides, this tool frees you to spend more time and effort on the crux of your application.

The System as Your Guide

In the absence of any system-provided support, or if the special needs of your application require you to build your own elements, use existing system elements as a general guide to style and function.

The Mouse

Having graphics on the screen that represent something to the user is good, but you need a way for the user to interact with those graphics. Although a number of devices can be used to do this on the Amiga (including a touch screen, a joystick or even the keyboard), most often the user will interface with the GUI via a two-button mouse.

Using a mouse can become a seamless, unconscious act for most users in the same way that a typist can type the letter "k" without consciously thinking: "Let's see . . . right hand, third finger from the right." Typists are able to do that because typewriter manufacturers make their keyboards consistent. Likewise, it is important that your software follows rules of mouse usage so that the GUI is "mechanically" consistent.

Follow basic rules for the mouse so that its use becomes instinctive.

The standard Amiga mouse has two buttons: the selection button (left) and the menu button (right).

The selection button (the left mouse button) selects or operates gadgets, icons, windows and screens. It can also be used to select special objects unique to your application (e.g., the note object in a music package) or to set the pen "down" in a drawing operation.

The menu button (the right mouse button) activates the menu system and

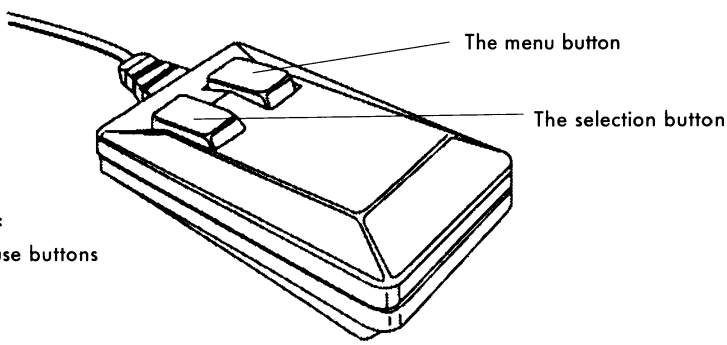


Fig. 2.5:
The mouse buttons

allows the user to choose a menu item. The menu button may also be used to abort a selection button operation. For example, the user is dragging a window around by holding in the selection button and moving the mouse. If he decides that this action was a mistake and wants the window back where it was originally, he can press the menu button and the window should snap back to its original position.

Activating Tools

Clicking on a tool gadget, such as the Fill tool in a paint package, with the selection button should activate the tool. If appropriate, a double-click could activate a settings editor for the tool so that the user can fine-tune the way the tool works.

Triggering a Process

Actions should be triggered on the release of the selection button.

When the selection of something will trigger a process, the action should occur on the release of the mouse button – not the downpress. This way the user can change his mind about the operation and move the mouse away from the object before he releases the button.

Current Objects

When an object is selected, that object is considered to be the current object. Some examples are: the highlighted section in a CAD program, a note in a music program, or the character under the cursor in a word processor. When the user chooses an action (via a menu item, for example) that action is carried out on the current object. For example, in a desktop publishing program the user can click on a text box and make that the current object. If the user selects an action such as "erase" or "copy," that action is performed on that text box.

Shift Selecting

In some instances, a user can select more than one item at once by using shift selection. The user clicks on an item with the selection button. While holding the

Shift key down, the user can then select any number of objects. The result is called a *selected group.* Some examples of objects that could be grouped are: columnar text boxes in a DTP program, notes in a music program and text items in a list.

Some actions that can be carried out on current objects will still apply to a group, such as copy or delete, but others may not. For example, you can have a requester showing the attributes of a current object but it would be difficult to have such a requester for a group of dissimilar objects.

Dragging

Objects can also be grouped by dragging. Dragging, a common mouse operation on many platforms, involves holding the selection button down while moving the mouse. There are two types of dragging: contiguous and non-contiguous.

A good example of contiguous dragging can be found in a word processor. The user clicks on a letter then, without releasing the selection button, moves the cursor somewhere else in the text. The first letter he clicked on becomes the "anchor" point and all the text between the anchor point and the point where the selection button was released should be highlighted. The highlighted text is now a selected group.

Objects can also be grouped contiguously by the use of the marquee. When the mouse is dragged on Workbench, for example, an animated dotted line, the marquee, marks the area covered by the mouse. All the icons within the marquee will be selected.

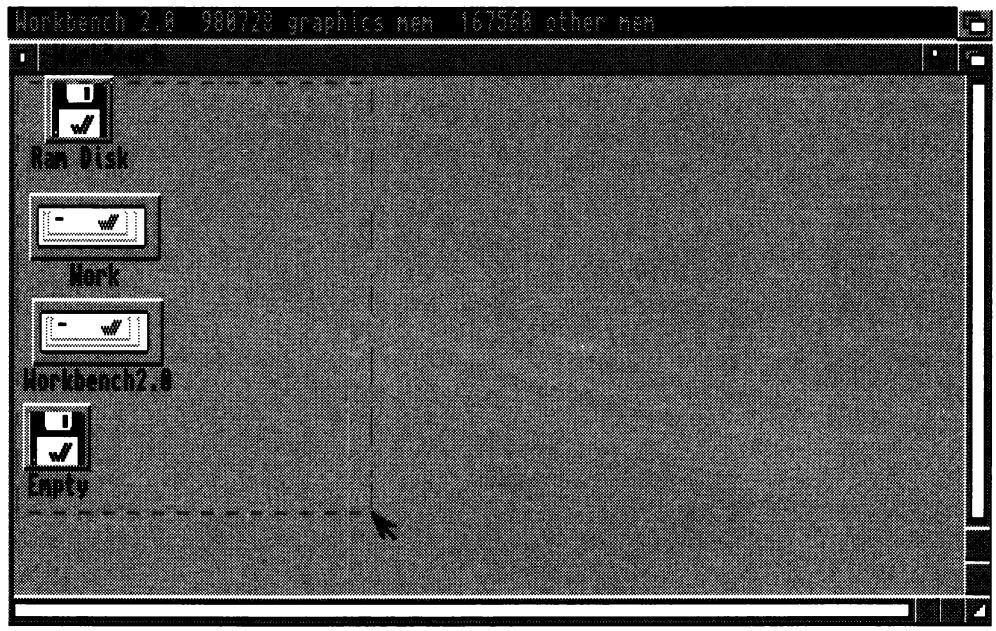


Fig. 2.6: Grouping by using the marquee.

Less common is non-contiguous dragging. One example of this is a program that allows the user to drag select a section of text and then deselect individual lines by clicking on them. The deselected line can be in between two selected lines.

In both styles of dragging, the display should scroll automatically if the mouse goes beyond the boundaries of the display region.

Four Ways to Highlight Text

There are four methods you can use in your program to highlight text: dragging (see the previous section), extended selection, multiple-clicking and selecting all.

With extended selection, the user clicks the cursor in the text (this becomes the anchor point), then shift-clicks the cursor at another place in the text. The area between the anchor point and where he shift-clicked will be highlighted. Note that the user can adjust the window's scroll bar between clicks and thus highlight a very wide range.

If you employ the multiple-click method, the user can select a word by dou-

ble-clicking on it, or select a paragraph by triple-clicking on it. (Note: these are not timed clicks.) A "word" can be defined simply as anything with spaces around it, or you can use a more complex definition. A fourth click should revert back to a normal cursor.

Selecting All would feature a menu item that would select all the characters in the document. This action shouldn't select the document as well; that is, any actions chosen for the selected group should not act upon the document itself. For example, if the user chooses select all and then erase, all the characters should vanish from the document's window but the window should remain.

There should only be one current object or one selected group at one time.

Current Objects vs. Selected Groups

Current objects and selected groups should not co-exist in the same application. In other words, no matter how many projects your application may run simultaneously, there should be only one current object *or* one selected group at any one time. If the user has a current object, then creates a group through shift selection, there should no longer be a current object.

Selection Context

Confine the user's ability to select to a single context or "level." For example, don't let the user select documents and characters at the same time. You can allow her to select any number of characters in a document, or any number of documents, but not both at once.

The Pointer

The Amiga uses a pointer that the user can redesign. The default pointer looks like this:

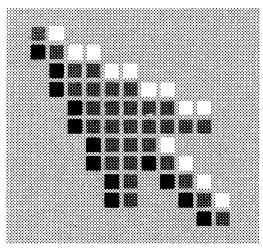


Fig. 2.7:
The default pointer with a superimposed pixel grid.

Your application can use a custom pointer instead of the standard pointer. The arrangement of color intensity in your custom pointer should be consistent with that of the standard pointer – especially since two different applications can share the same screen.

Technical note: here's how the colors should be set up:

Color 0	*Transparent*
Color 1	*Medium intensity*
Color 2	*Darkest*
Color 3	*Brightest*

The pointer should be framed by either color 1 or color 3 to ensure its contrast on most screens.

Waiting

When your application is busy with a task and temporarily cannot accept input, another pointer, known as a wait pointer, appears.

Currently the wait pointer used by Workbench is not accessible by developers. Shown here is a pixel-by-pixel view of Workbench's wait pointer if you want to make yours look similar.

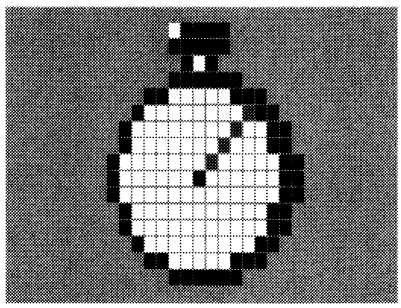


Fig. 2.8: Workbench's wait pointer with a superimposed pixel grid.

If the wait is for a measured amount of work, such as in a ray-tracing program, your application should provide a requester that shows the progress of the activity. The progress requester should have a gadget on it allowing the user to stop the activity.

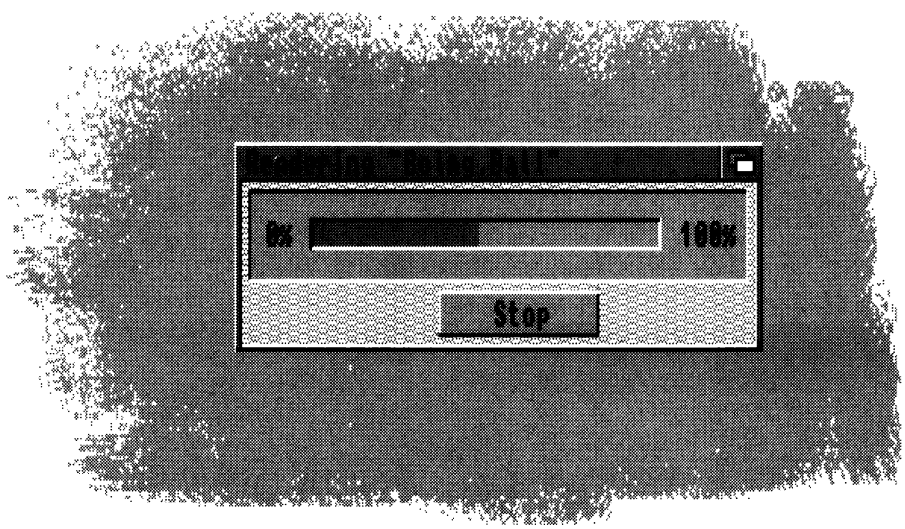


Fig. 2.9: A sample progress requester indicating the progress of a time-consuming activity.

Use a progress requester like the one shown on the previous page instead of an animated pointer to indicate that the application is busy. Here's why: A woman is rendering a simple object with a ray-tracing package. While waiting for that to finish, she decides to write a letter in a word processing package that opens on the same screen but in a different window. If the ray-tracing program displayed a progress requester, she could see when the rendering was done. But if it used an animated pointer, that pointer would disappear when she clicked in the word processing window.

Don't tie a wait pointer to the progress requester (i.e., when the user clicks on the progress requester the pointer changes to a wait pointer). If you do, the user may think she can't activate the "Stop" gadget.

Don't show multiple cycles in the same progress bar. For example, if the progress bar reads "Calculating . . .", fills up and then reads "Rendering . . ." you've falsely built up the hopes of the user. Instead, use multiple bars if knowing the progress of each task is important to the user, or a single bar that is filled only once if knowing when the job is done is all that is important.

Pointers with Purpose

Pointers can be used to give the user a context cue.

For instance, if your application supports different tools, such as the sketch tool or the fill tool in a paint program, the pointer imagery can reflect the currently selected tool.

Or, if your window is divided into distinct areas with different uses, it may be more appropriate to have the pointer image reflect its current purpose based on its position. The pointer for a CAD program, for example, might be a crosshair while it is over the drawing area, but turn into the standard arrow when it is over the control panel area.

Resolutions

The Amiga hardware and software offers a variety of resolutions for the user to choose from. The resolution refers to the number of pixels in that mode. For instance, a common Amiga resolution of 640 x 200 refers the number of pixels in height and width, respectively.

If you allow the user to choose resolutions, make sure that your GUI fits comfortably on whatever screen the user has chosen. A good way to do this is by checking it on a least common denominator resolution – a non-interlaced, non-overscan NTSC screen (640 x 200) with a Topaz 8 font. You don't need to set these as defaults; just make sure that what you create will look and function normally when set to these settings.

Design your GUI on a 640 x 200 screen with the Topaz 8 font.

If your original target audience will most likely be using PAL, it's still a good idea to design with a 640 x 200 NTSC screen in mind so your distribution isn't limited later.

Respect the choices the user has made in his Preferences setup for resolution and how large a display should be set up.

Technical note: If your application is text-oriented, make sure you use the text overscan setting. If it's graphic-oriented, use the "standard" overscan setting. The user can set both of these by using Workbench's Overscan preferences editor. ▲

chapter three
SCREENS

The Buddha, the Godhead,
resides quite as comfortably in the circuits
of a digital computer
or the gears of a cycle transmission
as he does at the top of a mountain
or in the petals of a flower.
- Robert Pirsig

By organizing raw data into neat and friendly metaphors, GUIs make using a computer more intuitive and comprehensible. Still, sifting through the many possibilities in an effort to get to a desired task can often lead the user to wonder: "Where am I?" and "What should I do next?" The fact that the Amiga multitasks – that is, it allows the user to run a number of applications at the same time – only increases the need for strong context cues.

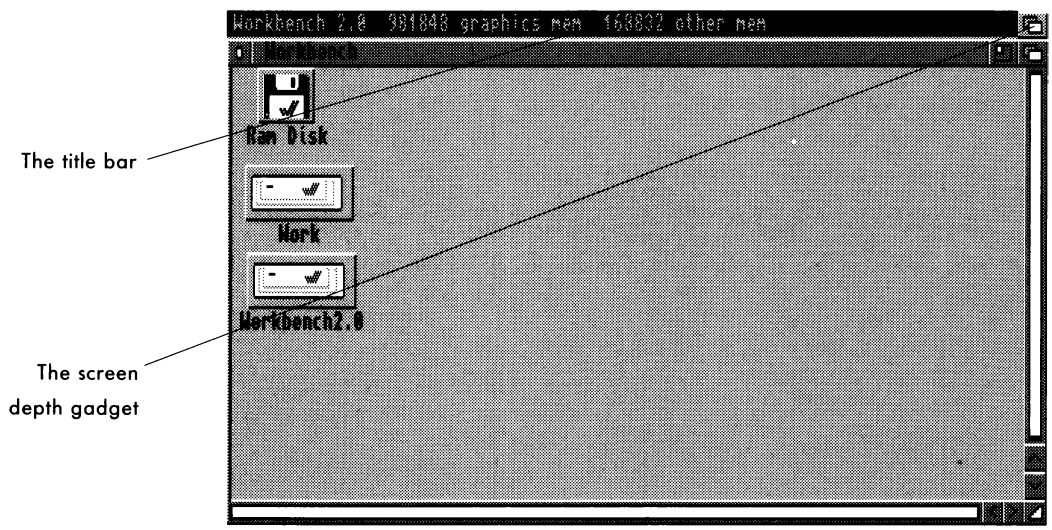
When your application is run, it indicates the new context by either opening up in a window or on its own screen. Thus, screens and windows provide the main cues that tell the user where they are at any given moment.

DEFINING SCREENS

Screens are unique to the Amiga. Other platforms have a single environment filling the monitor view or perhaps extending beyond that in height and width. On the Amiga, a user can have multiple screens, each an environment unto itself with its own palette, resolution and fonts – running at the same time.

Typically, screens are at least as wide as the monitor display and have a single title bar at the top of the screen which is shared by all the applications that operate within that screen. It is possible, however, to have screens that are larger than the display area (known as virtual screens), or to have a screen that is not as tall as the monitor display. Applications will sometimes use the shorter screens as control panels in a different resolution than the display area. Workbench is the default screen a user is presented with upon booting the machine.

Fig. 3.1: The Workbench Screen



Screens cannot be resized. New screens usually appear in front of existing screens. The user can access screens in the back by dragging the front screen down or flipping through the screens by using the screen depth gadget or keyboard combinations.

Types of Screens

Your application can open on one of three types of screens: the Workbench screen, a public custom screen that your application shares with other programs, or your own private custom screen.

When your application uses the Workbench screen, it opens a window on the Workbench, using the palette, resolution and fonts that are defined in the Workbench Prefs.

The Workbench screen is a public screen – that is, it can be used by any application. If your program needs a different resolution or palette than the user has chosen for his Workbench preferences, it should open on a public custom screen unless its requirements are restrictive enough to warrant a private custom screen.

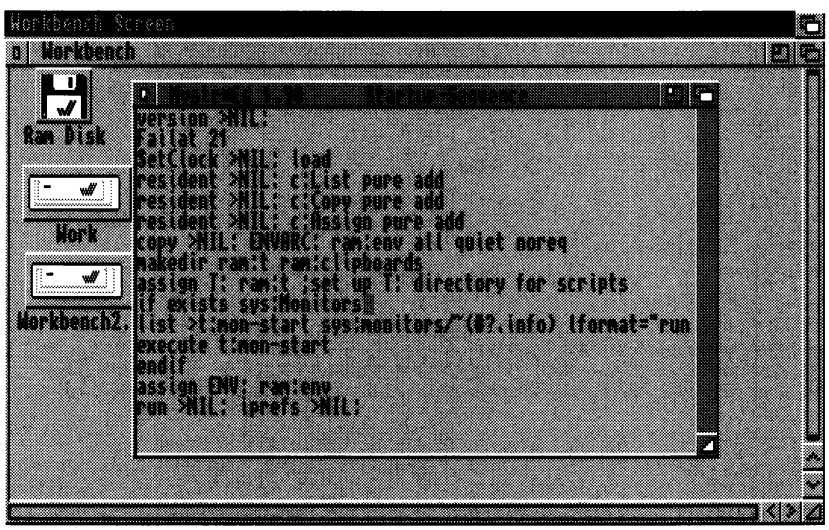


Fig. 3.2:
A text editor
open on
the Workbench
screen

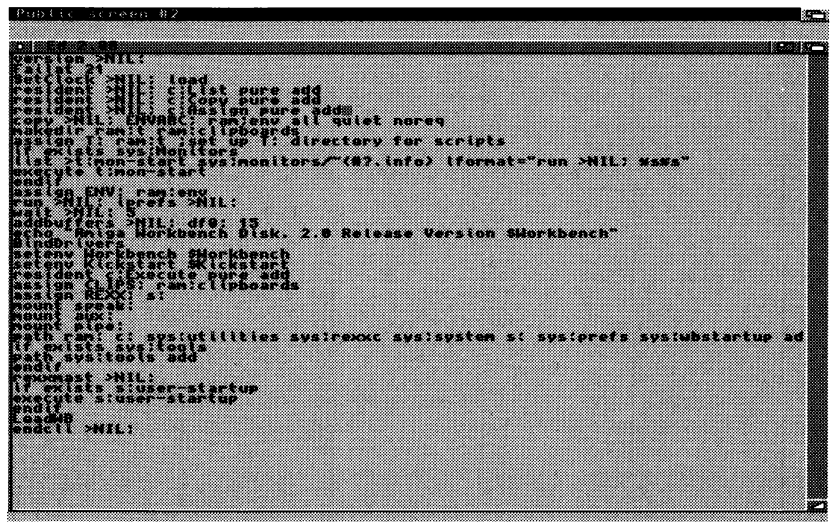


Fig. 3.3: A text editor opened on an interlaced public custom screen

By keeping your custom screen public, you allow users to access other, perhaps supportive, applications without having to flip your application to the rear. Or, if the user is already running an application on a public screen with the proper palette and resolution requirements, he can open your program on that screen.

A private custom screen is one that you set up to your specifications and which only your application may use. Private custom screens should be used only when your application has unusual rendering or resolution requirements, or when you need to be able to operate on the whole screen directly. An example of this would be an animation program that needs to switch view ports rapidly in order to get smooth motion.

Technical note: If your application opens a custom screen, make sure you redirect requesters to the custom screen – this applies to relevant DOS requesters as well as your application's requesters.

RESPECT USER CHOICE

Let the user decide, if possible, whether to open your application on Workbench or on another screen. If he chooses to open it on a custom public screen, let him choose whether it will be a new screen or one that has already been created by a different application.

Likewise, respect choices the user has already made. All custom screens, whether public or private, should default to the basic parameters established in Workbench's preferences, unless your application has special requirements.

If your application provides overscan capabilities you should respect the settings that the user has established in the Overscan editor found in Workbench's Prefs directory.

SCREEN DESIGN

Screens should have a depth gadget. If you have room, you should try not to obscure the screen depth gadget from view by opening windows that cover up the screen's depth gadget.

The screen your application opens on should open in front of any other screens that are open.

Auto-scrolling

Applications that open screens larger than the display area should provide the ability to auto-scroll. Moving the mouse to any of the display bounds should automatically scroll the screen to show more information in that area.

Screens that open larger than the display area should offer an auto-scrolling function.

Naming Your Public Screens

Public screens are identified by their name. To name a public screen, use your application's basename (see page 12) followed by an invocation count. The name should be in upper-case.

For instance, a terminal package with the basename Axelterm that opens its own public screen should name the screen AXELTERM.1. On a system with a multi-serial port card, the user may decide to run a second copy of the package. That second public screen should be named AXELTERM.2.

Some applications may use two screens during the same invocation of the program. For example, a paint program named VanGogh (with the basename VGOGH) may use one screen for the "canvas" and another screen with a different resolution for the control panel. In that case, the screens could be named VGOGHPAD.1 and VGOGHPANEL.1. ▲

chapter four

WINDOWS AND REQUESTERS

The dogs did bark, the children screamed,
Up flew the windows all;
And every soul bawled out, Well done!
As loud as he could bawl.
- William Cowper

The last chapter noted that context cues on the Amiga come in the form of screens, windows and requesters. Whereas screens represent general environments, windows and requesters show the user where he is much more specifically.

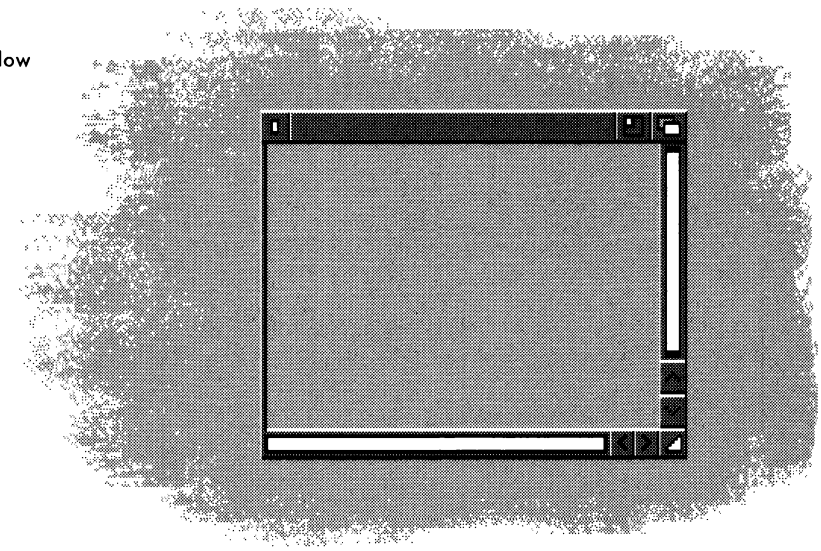
WINDOWS

A window is simply a graphic rectangle on an Amiga screen, but it tells the user that everything within its boundaries is linked by a common bond – be it a common directory, program or document. If he returns to that same window later, he knows from experience what that window can be used for.

A Safe Place to Click

Try to provide a safe place on each window where the user can activate it

Fig. 4.1: A window



without altering any previous work. For instance, if the user were to return to a text processor window after checking his email, he should be able to activate the window and not have his cursor move. The title bar is a good area for this.

If your application uses the whole window for operations and has no window title bar (like some paint packages), let the first click of the mouse on an inactive window activate the window instead of doing an operation.

Size and Position

Each window opened by your application should be able to fit within a medium resolution screen, 640 x 200, with the Topaz 8 font. These don't need to be the defaults, just a working guideline. See *Resolutions* on page 31 for more information on this.

Your application should provide a default location and size for its windows. By default the windows should open within the current view area of the screen (remember that some users work with virtual screens) and should be scaled per the current resolution. You should, however, let the user override these defaults both temporarily and permanently.

For example, if the user moves or resizes a window during a session and later closes the window, your application should remember those settings and use them if the user reopens the window within that session (a "session" being the time between when the user starts the application and when he chooses the Quit Program menu item). In this case, your program should remember the user-specified positions only for the length of the session.

Let the user store his preferred window position "permanently" via the Save Settings menu option (see Chapter Six for more information on menus). Unlike the scenario in the previous paragraph where the position is remembered only for that session, Save Settings saves the position for use each time the application is opened.

The Save Settings menu option allows the user to specify a default size and position for windows.

Opening at Different Resolutions

Save Settings should also save the screen width and height so that the window rectangle can be scaled, if necessary, according to the screen size.

For instance, imagine that the user has set his preferred window position to the lower half of a 640 x 400 screen. If he later runs the same application on a 640 x 200 screen, the window will have to be adjusted to fit properly on the screen. The first action to take would be to move the window to a place on the screen where it will fit. If this doesn't take care of the problem, the window should be scaled.

In the example above, the screen height was halved, so the position of the window's top edge should be moved up halfway to the top of the screen. If the window has a sizing gadget, the window size should be scaled as well. Be sure to take into consideration the font used in the window title bar when computing the window's maximum/minimum sizes.

If your window can't be scaled and doesn't fit in a certain resolution, consider restricting the screen's resolution that the window opens on. Try, though, to respect whatever choices the user makes.

Opening on a Virtual Screen

Screens can be many times the size of the actual display area. When opening a window within one of these screens, make sure the window is positioned by default in the onscreen display area, but if the user changes those defaults via the Save Settings menu option, respect the user's settings, even if it causes your window to open offscreen. He may have a reason for wanting the window to appear where it does.

Successive Windows

When your application opens a number of project windows, you may want to position each successive window slightly lower than the previous one – preferably, the height of the title bar plus one pixel. This will leave the window depth arrangement gadget open.

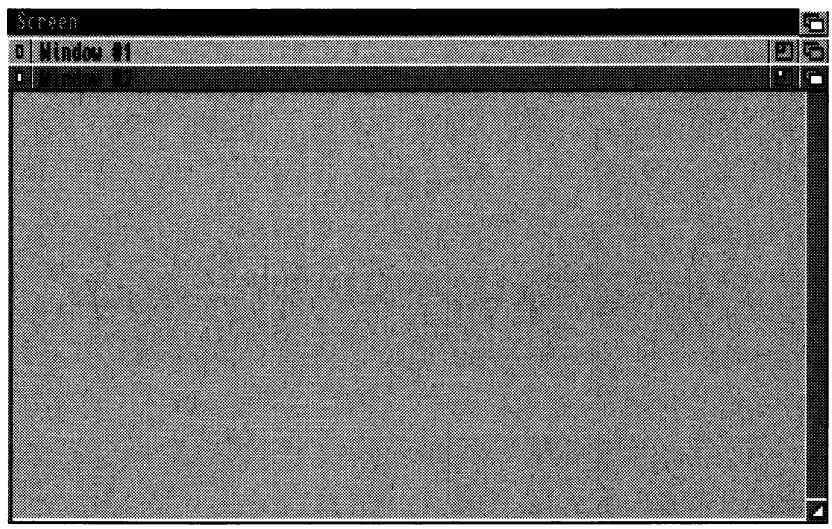
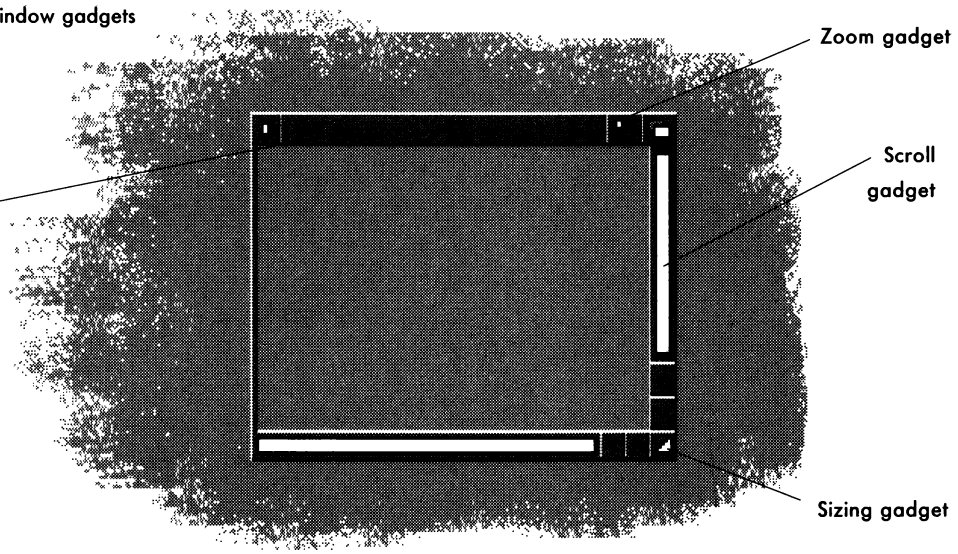


Fig. 4.2: Overlapping windows

Window Gadgets

Here is a specific discussion of when and how to use system gadgets on your windows. More about gadgets can be found in the next chapter.

Fig. 4.3: Window gadgets



Dragging Windows

Whenever possible, windows should be draggable.

Requesters and support windows should always have a title bar that can be used to drag the window. An immovable support window could block important information that relates to it. A Find and Replace window, for example, may obscure the view and thus the spelling of the word the user wants to search for in the document.

Sizing and Scrolling Gadgets

When a window contains a view or editing area, like in a text editor, it should have a sizing gadget so the user can adjust the window to show more information. If the entire window is used as the view area, a scroll gadget should be added in the right-hand border of the window.

Zooming Windows

The first click on the window's zoom gadget changes the window to its alternate size; the next click restores it to the size it was before the gadget was clicked.

You will have to specify what the small size of your windows will be; Intuition will handle the rest. If the window is sizable and your application opens full size, the setting should be the minimum size of the window, which can vary according to your application's needs. If your application opens small, the zoom gadget's alternate size should be initialized to full size. If the window isn't sizable, the minimum size should be the height of the title bar and whatever width is necessary to fit the title of the window; or you may choose to omit the zoom gadget altogether.

AppWindows

The active part of an AppWindow (see Chapter 7 for a discussion of AppWindows) should be indicated by an icon drop box gadget. This gadget should be an outlined rectangle. If your AppWindow has different areas that perform different operations, each area should have a separate icon drop box.

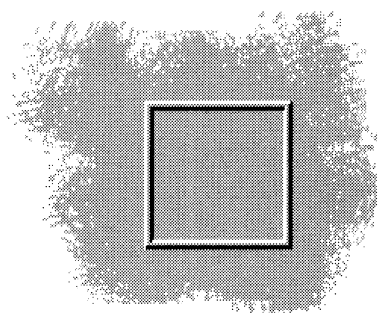


Fig. 4.4: An icon drop box gadget

REQUESTERS

Many times when attempting a task, a user will reach a time of choice, a fork in the road. At these junctures, good road signs can be provided by requesters. Requesters can be broadly defined as sub-windows that allow the user to control options, access files and confirm actions.

Types of Requesters

Requesters can be modal or non-modal.

Non-modal requesters, also called support windows, act basically like windows. The choices and/or information are presented to the user, but she can temporarily ignore the requester and continue entering data if she wishes.

Modal requesters block input to the application until options are chosen or actions are confirmed.

Use Non-Modal if Possible

Generally, you should use non-modal requesters unless there is some reason to block further input. Modal requesters confine the user's movements – that is always something to avoid if possible.

However, there are times when a modal requester is preferable. One example is a requester found in the utility HDToolbox. When the user tries to save changes she has made to a hard disk's partitions, a requester appears warning that saving the changes will wipe out all the data on the hard disk. Naturally, this requester should and will block further activity until the user acknowledges the warning.

Use the Wait Pointer with Modal Requesters

While a modal requester is on the screen, the parent window or screen should set its pointer to a wait pointer.

The Modified Project Requester

Whenever the user makes changes to a project currently in memory and subsequently selects an action such as New, Open or Quit, any changes made to the current project will be lost. In that case, a modified project requester similar to the one below should be presented to the user:

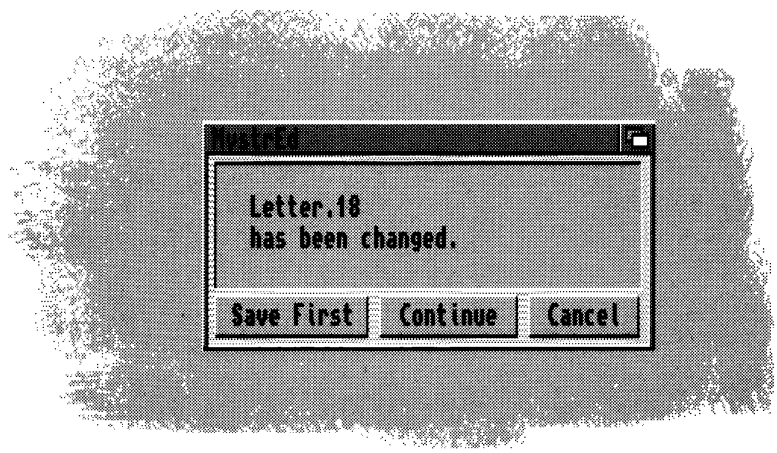


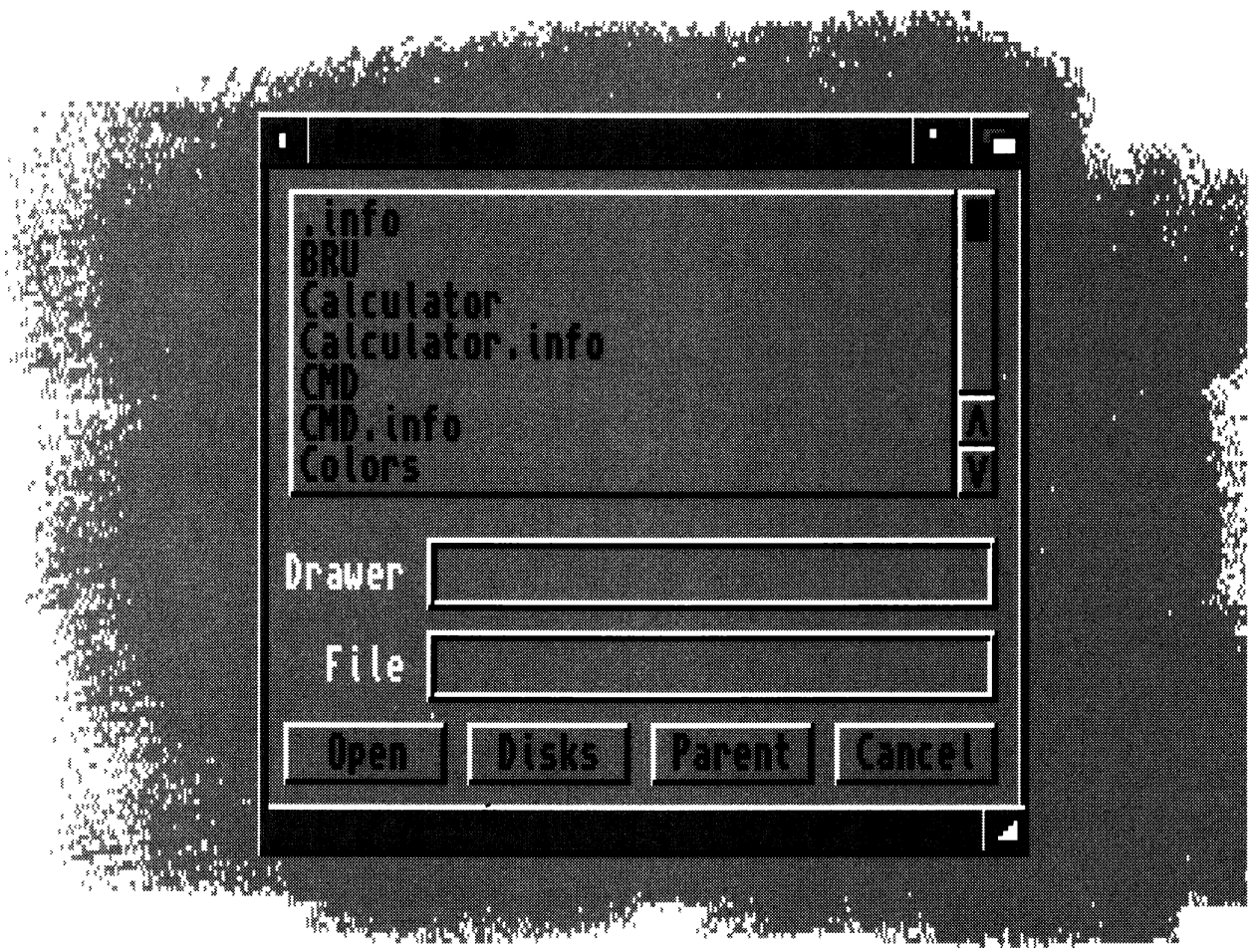
Fig. 4.5: A requester which should be used if the user tries to leave a file before saving the changes

Requester Design

If possible, use the standard requesters provided in the ASL library. Requesters are base-level operations and a user shouldn't have to learn new requester operations with each new software package he buys.

If you need to create your own, follow the ASL design. Here's an example of the ASL file requester:

Fig. 4.6:
The standard ASL file requester shown here is the actual size it would be on a 640 x 200 screen.



Spend some time on the wording and design of your requesters. Make sure they communicate clearly and succinctly what choices the user has and what will happen as a result of his choice.

Draggable

Both types of requesters should be draggable. Even if the user can't enter data to the program, there may be information on the screen that he needs to see and a non-draggable requester may cover that information.

Where to Open Requesters

Requesters should be opened adjoining or within the boundaries of the parent window. Use coordinates relative to the parent window rather than absolute coordinates when positioning a child window.

Positioning is especially important when the user is working with an extremely large virtual screen or a high-resolution monitor such as Commodore's A2024. On a monitor like the A2024, not limiting your requesters to the parent window boundaries can cause an annoying break in the work flow. Here's an example: the user is working in the lower right corner and you open a requester

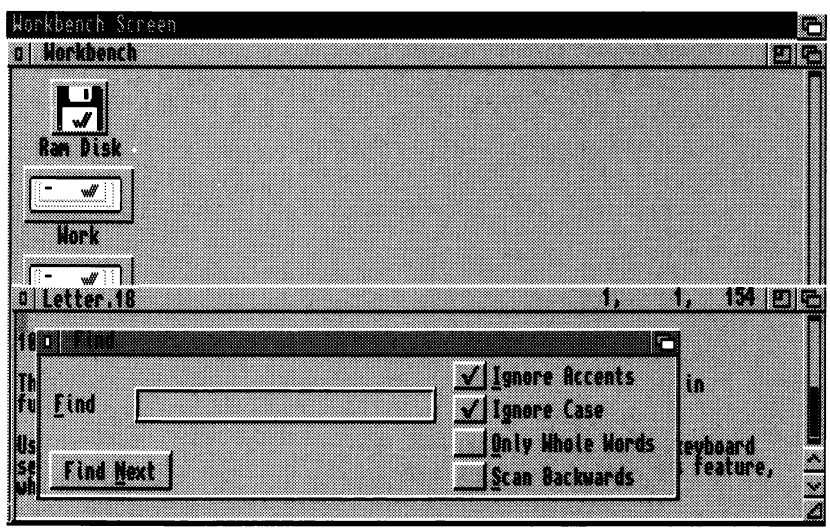


Fig. 4.7: A find and replace support window opening within the boundaries of the parent window.

in the upper left. Relatively, that's a lot of screen real estate to move the pointer across. On a virtual screen the requester may not even show up on the current monitor view, leaving the user to wonder if his action had a result at all.

Always Give a Safe Way Out

Operations should always provide a safe way for the user to back out of a requester. Normally this is handled through a "Cancel" action gadget in the lower right of the requester.

Give Directions

When your program produces a requester indicating its failure to find a file, the requester's title bar should name the program that is looking for the file and the specific name of the file should be given in the text of the requester. For instance, getting the following requester at boot up could severely limit any troubleshooting:

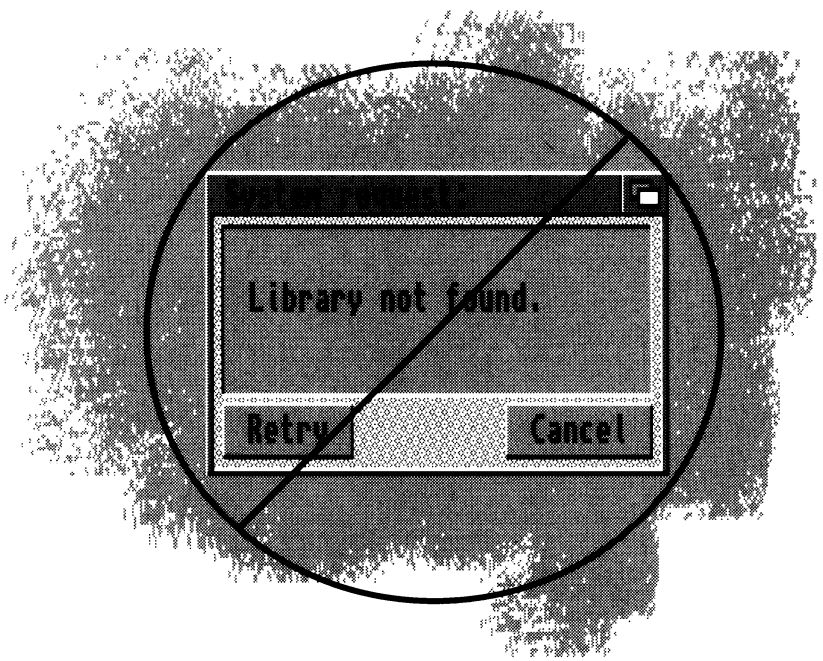


Fig. 4.8:
The wrong way to do a requester stating that a file can't be found. (See the next page for the correct way.)

A better requester would be:

Fig. 4.9: The correct way to tell the user that your application can't find a file.

OK and OK

Don't create requesters with two identical action gadgets, such as OK and OK. A single gadget for each choice will suffice. ▲

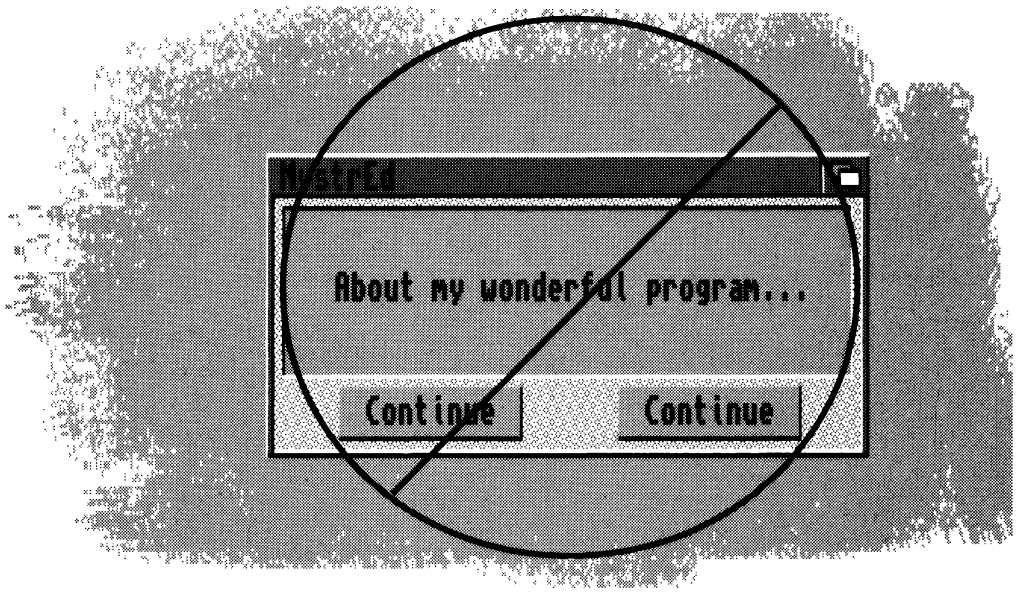


Fig. 4.10: Another incorrect requester. This one gives two identical action gadgets and no way for the user to back out of the operation.

chapter five

GADGETS

As for Clothing . . .
perhaps we are led oftener by the love of novelty
and a regard for the opinions of men, in procuring it,
than by a true utility . . .
Kings and Queens who wear a suit but once,
though made by some tailor or dressmaker to their majesties,
cannot know the comfort of wearing a suit that fits.
- Henry David Thoreau

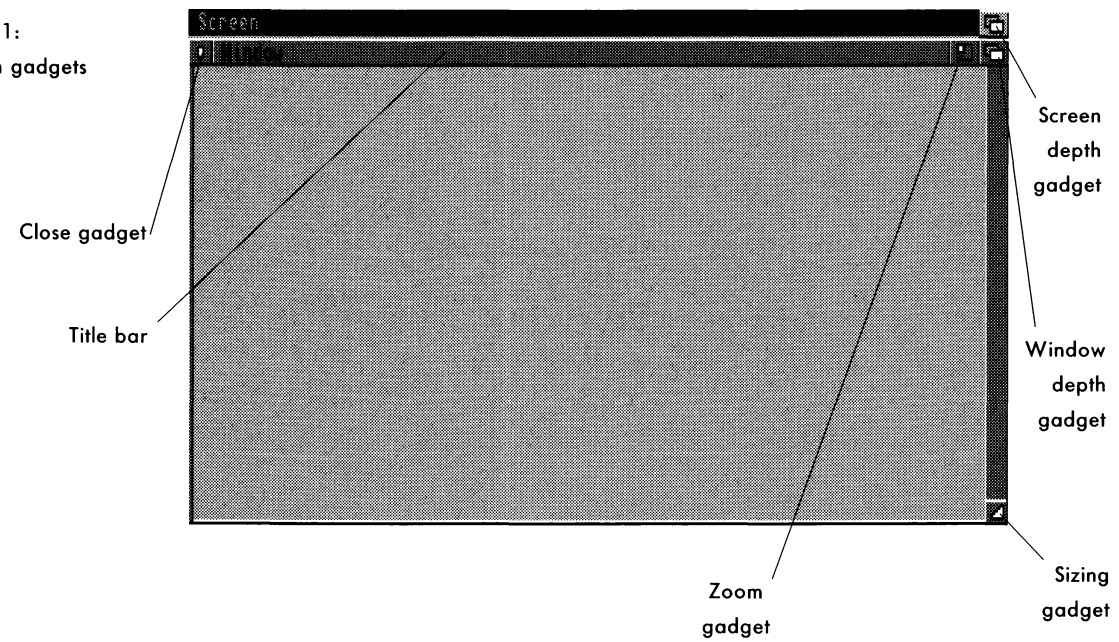
Gadgets are graphic symbols that represent a specific action or control. They are usually contained in a window or requester.

There are two basic types of gadgets on the Amiga: system gadgets and application gadgets.

SYSTEM GADGETS

System gadgets control aspects of the GUI environment like window size and screen positioning. System gadgets can be thought of basically as environment maintenance tools. This screen's in the way – click it to the back. This window

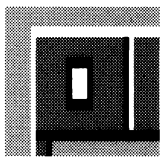
Fig. 5.1:
System gadgets



needs to be bigger – click and drag on the sizing gadget.

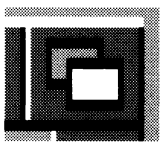
System gadgets are highly standardized in when and how they are used. Intuition will handle most of their functions. The table that follows lists the gadgets and how they operate. This is mainly for quick reference. Since the gadgets are tied closely to windows and screens, consult those chapters for more detailed information.

system gadgets



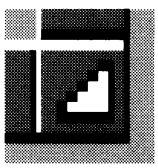
CLOSE GADGET

Located on a window's upper left corner, the close gadget removes the window from the screen and quits whatever program or file the window was running.



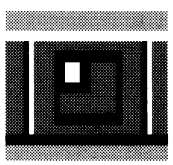
DEPTH GADGET

Located on a window or screen's upper right corner, the depth gadget adjusts which window or screen is in front of all the others. For example, if a window is in front of all the other windows on that screen, clicking once on the depth gadget will put it behind all the other windows. Clicking again on that window's depth gadget will bring that window in front of all the other windows.



SIZING GADGET

Located on a window's lower right corner, the sizing gadget allows the user to resize the window by clicking on the gadget and dragging. Not all windows have or need a sizing gadget although its use and support is strongly encouraged.



ZOOM GADGET

Located on windows next to the depth gadget, the zoom gadget allows users to quickly reduce a window to its minimum size to temporarily get it out of the way. When it is needed again, another click on the zoom gadget will bring the window back to the size it had been. Not all windows have or need a zoom gadget although its use and support is strongly encouraged.

This is only a reference listing of system gadgets. For more information about these see the chapters on Screens and Windows.

APPLICATION GADGETS

Application gadgets represent choices the user can make and thus proceed with his task. As a developer, you will have to make more decisions about how and when to use application gadgets than you will with system gadgets.

Design

Like other aspects of the GUI, gadget layout and size should be based on a 640 x 200 screen resolution with the Topaz 8 font. See *Resolutions* on page 31 for more information.

At run-time, your application should check the size of the font and the screen resolution to determine if the gadgets used will fit in the window with the user's preferred settings. If not, revert to the Topaz 8 font.

Labeling

Put a label on gadgets whose purpose is not immediately obvious. Labels should be terse without being obscure – preferably one to three words. Ponder on these a while and test them if possible before incorporating them into your program. Capitalization should follow good grammatical sense for the language being used. If in doubt, refer to a writing or journalism style guide for your language. The glossary in the back of this manual gives the English capitalization of many Amiga terms.

Ghosting

As with other elements of the GUI, a gadget that is temporarily unavailable for selection should be obviously disabled. Don't allow the user to select something that does nothing in response. Follow the ghosting standard shown in Chapter Two.

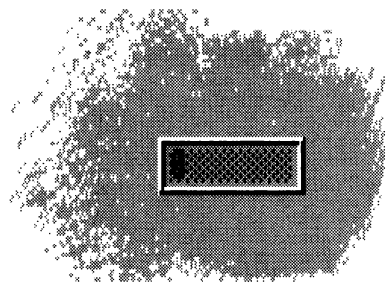


Fig. 5.2:
A ghosted
text gadget

Keyboard Equivalents

As a convenience, a key may be bound to each gadget so that its actions can be controlled from the keyboard as well as through the mouse.

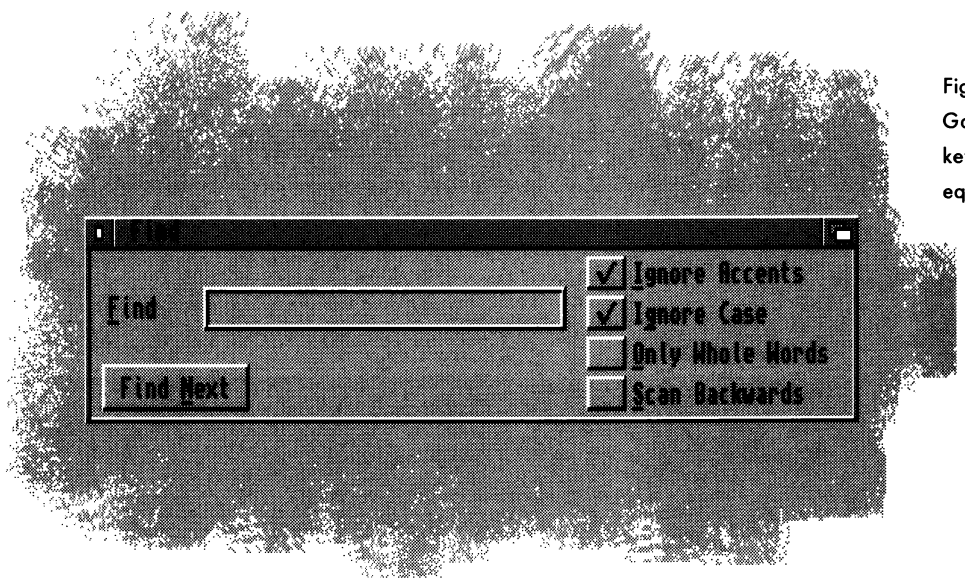


Fig. 5.3:
Gadgets with
keyboard
equivalents

Use a logical letter from the gadget label as the key control for the gadget. For instance, a gadget labeled "Spacing" could use "S" as its keyboard control, whereas a gadget labeled "Get Fonts" may use "F". The letter you use should be underlined on the gadget.

Restrictions

Never provide keyboard equivalents for asynchronous requesters. An asynchronous requester is one that occurs due to an action of the application rather than an action of the user. In a hard disk backup program, for example, asynchronous requesters appear asking for a new floppy disk when the current disk is full.

Because asynchronous requesters can appear when the user isn't expecting them, keyboard equivalents can lead to the user choosing an unwanted action. Backup programs, to use that example again, don't require a lot of user attention, so there's a good chance the user will go to another application. If the backup program's requester comes up asking for a new floppy disk while the user is typing in a word processor, unwanted actions could easily occur.

Left-Amiga-V and left-Amiga-B will, respectively, activate the extreme left and extreme right gadgets in a requester.

An asynchronous requester will have two default keyboard equivalents, however, that are built into the system. Left-Amiga-V and left-Amiga-B will, respectively, activate the extreme left and extreme right gadgets at the bottom of a requester.

Another restriction on keyboard equivalents for gadgets: don't use the Return key to activate the OK gadget. This applies to both synchronous and asynchronous requesters.

Visual Feedback

Visual feedback should be given when a keyboard equivalent is used. This feedback should be the same as the feedback given when the mouse is used. The specific feedback for each gadget is listed with the gadget description later in this chapter.

Grouping

Gadgets should be grouped logically on your requesters or support windows. Some general premises about grouping follow. Use these premises and your own common sense when grouping elements.

The general rule is that gadgets should be grouped according to function. On a print requester, for example, gadgets controlling text would go in one area, while those affecting graphics would go in another.

Another rule is to place commonly used functions within easy reach, especially on a crowded control panel.

Don't put dangerous controls, such as Delete or Format, near commonly used controls.

Think of the user's work flow when you order the gadgets. Try to emulate a logical and intuitive order. If there is an order of events, start in the upper left and work to the lower right (depending on local language, of course).

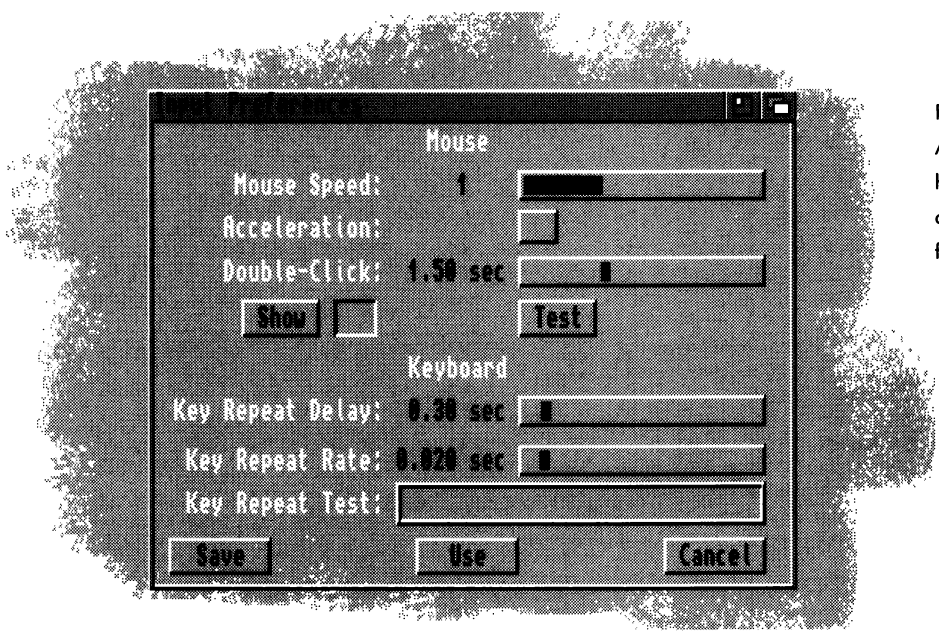


Fig. 5.4:
A requester that has grouped gadgets according to function.

APPLICATION GADGETS BY TYPE

Each of the Amiga's application gadgets has a particular use and limitation. This section gives an overview of each type. Each type of gadget can be recognized by its appearance. Do not make gadgets that look like these standard gadgets but act differently.

This section lists application gadgets and their functions.

Action Gadget

Action gadgets (often referred to as buttons) are graphic rectangles usually containing a few words. Clicking on an action gadget should perform the activity named on the action gadget.

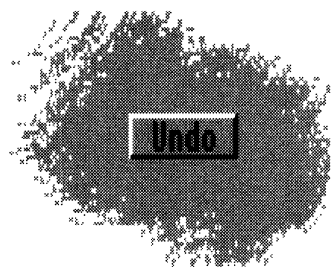


Fig. 5.5:
An action gadget

Labels

Make the label descriptive. "OK" and "Cancel" may not always be the best choice. Use friendly, less technical terms (i.e., "Stop," rather than "Abort").

The user shouldn't have to read an entire body of text before deciding which action gadget to press. Instead, a carefully selected label should tell the user what the gadget does in one to three words.

Triggering the Action

The action should be triggered on the release of the mouse's select button – not the downpress. This gives the user the chance to "roll off" the gadget before activating it.

More Choices

When an action gadget brings up another window or requester, the label should end in an ellipsis (three periods).

Fig. 5.6:
A gadget which brings up another window or requester should have a label ending in an ellipsis.

Use of Cancel

An action gadget labeled "Cancel" should only be used if it actually allows the user to back out, leaving the application exactly as it was before the requester appeared. For example, "Cancel" is not appropriate for a gadget on a requester that is displayed while printing is occurring. "Stop" would be a better label.

Gadget Placement

The positive choice, or continuation of the requested action, should be displayed on the lower left side of requester/window, while the negative choice or discontinuation of the action should be displayed on the lower right.

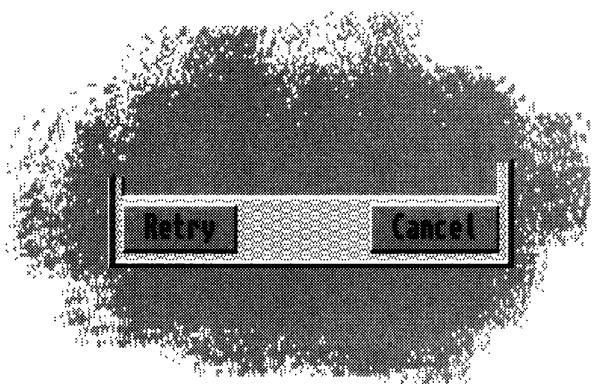


Fig. 5.7:
The positive choice should be placed in the lower left, while the negative choice should go in the lower right corner.

Keystroke Activation

If an action gadget is activated through a keystroke, the gadget should appear to be pressed in on the downpress of the key. On release of the key, the gadget should appear to come back out.

Check Box

A check box is a small square that toggles from being blank to having a check mark in it. Check boxes are appropriate whenever you need to present an option that may be turned off or on.

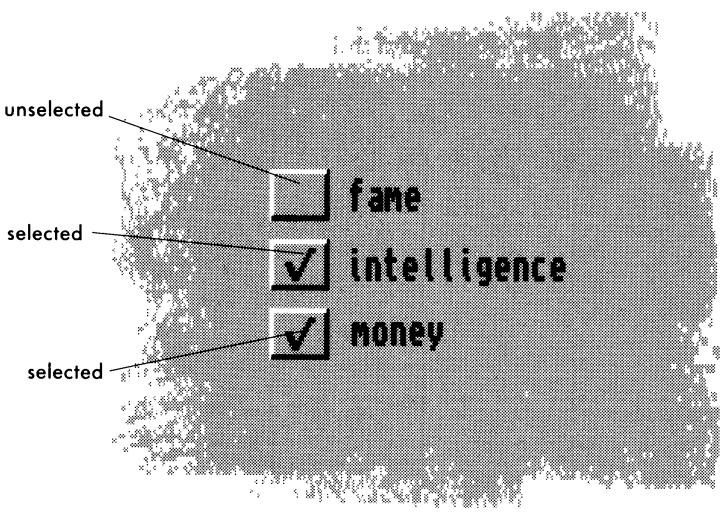


Fig. 5.8:
Check box gadgets

Keystroke Activation

If activated through a keystroke, the state of the check mark should toggle within the box.

Scroll Gadget

Scroll gadgets are used to adjust the position of a view. By themselves, scroll gadgets are used to adjust a large display area within a window's view, such as a text file that won't all fit within one window's view. Scroll gadgets are also a component of a scrolling list gadget (see next section).

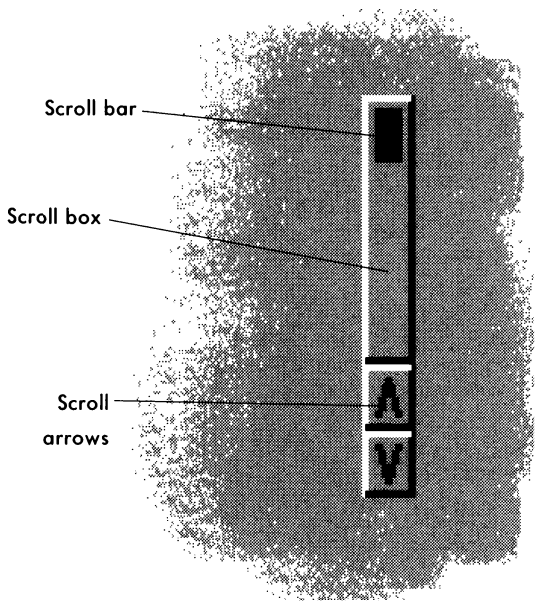


Fig. 5.9:
A scroll gadget

A scroll gadget is comprised of the scroll bar, scroll box and scroll arrows.

Any window that displays only a portion of the file's entire contents should have scroll gadgets. If your application allows the editing of files that are wider than the window, it would be good to have a horizontal scroll gadget as well.

Moving Through in Steps

The display area should be updated immediately as the scroll bar is dragged.

When the user clicks in the scroll box (the rectangle containing the scroll bar) but not directly on the scroll bar, the display should move in viewfuls. For instance, if the user clicks in the scroll box above the scroll bar, the user's view of the list should move up so that the line which was first is now at the bottom of

the view. The inverse applies when the user clicks below the scroll bar. This lets the user walk through the list in steps without missing anything on the list. Leaving one line from the previous view assures the user that he hasn't missed anything.

Scrolling List

A scrolling list features a view box showing textual names of files or objects accompanied by a scroll gadget to the right of the view box. If the list is longer than can be accommodated by the view box, the user can move through the list using the scroll gadget.

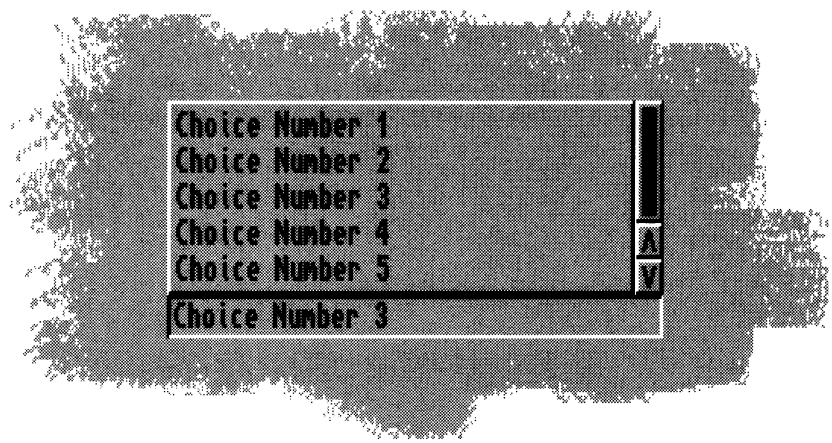


Fig. 5.10:
A scrolling list

A scrolling list should be used whenever you need to present the user with a variable list of objects. Probably the most common example is found in the requester presented when a user chooses to open or save a file.

Custom Scrolling Gadgets

On the Amiga, only one of the items in the system-supplied scrolling list may be selected at a time. Using custom code, it is possible to create a scrolling list gadget that supports multiple selection. If implemented, that gadget should still

follow the multiple selection guidelines for text covered in Chapter Two.

Keystroke Activation

The scrolling list gadget reacts differently to shifted and unshifted keystrokes. An unshifted keystroke should cause the list to scroll forward through the choices. A shifted keystroke should cause the list to scroll backwards through the choices.

Note: Using the Shift key in tandem with another key should never be the *only* way to do things since it is usually a choice that is not immediately apparent to the user. In the above case, it is backed up by the mouse.

Radio Buttons

Radio buttons are a group of mutually exclusive gadgets – one and no more than one is always selected. Use this when the user must choose one option from a short list of possibilities.

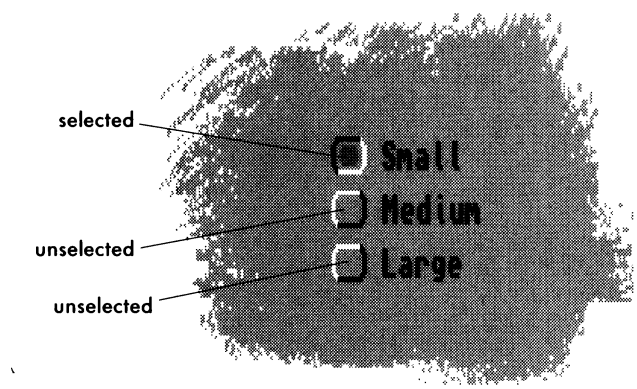


Fig. 5.11:
Radio buttons

Radio buttons are similar to cycle gadgets in functionality. Each has its benefits and drawbacks. See page 64 for a discussion on this subject.

Keystroke Activation

Radio buttons react differently to shifted and unshifted keystrokes. An unshifted keystroke should cause the highlighted button to cycle in one direction.

A shifted keystroke should cause the highlighted button to cycle in the opposite direction.

Cycle Gadget

Like radio buttons, cycle gadgets allow the user to choose one option from several but only the selected option is visible.

Cycle gadgets should be used to set attributes, not trigger actions. Never use a cycle gadget for an on/off choice – use a check box instead.

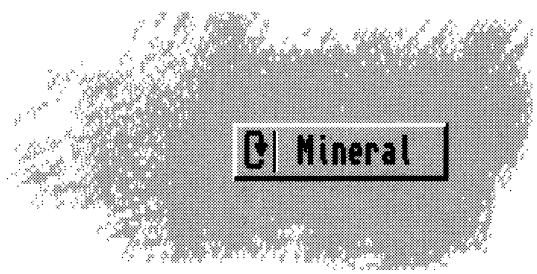


Fig. 5.12:
A cycle gadget

Keystroke Activation

The cycle gadget reacts differently to shifted and unshifted keystrokes. An unshifted keystroke should cause the gadget to cycle forward through the choices. A shifted keystroke should cause the gadget to cycle backwards through the choices.

Cycle Gadgets vs. Radio Buttons vs. Scrolling Lists

If the user needs to choose one option from a choice of three or more, cycle gadgets, radio buttons and scrolling lists are viable options. Which one you choose to implement depends on your application, your preference and common sense. Consider the following points:

A cycle gadget presents a cleaner interface. The selected choice is quickly

evident and the unwanted choices are hidden away. Theoretically it's also able to handle a larger number of choices, although users will probably have trouble remembering a really large number of choices. Large numbers of choices may work well in a cycle gadget if they are ordered choices, such as the months of the year.

In general though, options with more than about a dozen choices should use a scrolling list.

Radio buttons, present a clear choice to the user – the possible choices are always visible. Radio buttons work well with a small number of options. They are probably also slightly more intuitive than cycle gadgets.

Color Selection Gadget

The color selection gadget (sometimes referred to as the palette gadget) allows the user to pick a color from a set palette.

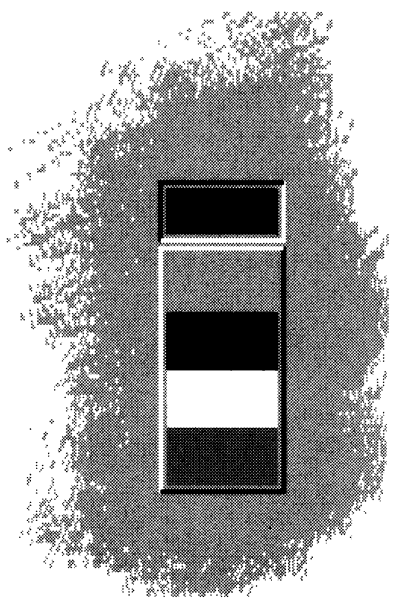


Fig. 5.13:
A color
selection gadget

Keystroke Activation

The color selection gadget reacts differently to shifted and unshifted keystrokes. An unshifted keystroke should cause the gadget to cycle forward through the choices. A shifted keystroke should cause the gadget to cycle backwards through the choices.

Slider

Sliders are used to choose a value in a given range. Usually, this value represents a level or an intensity, such as volume or color.

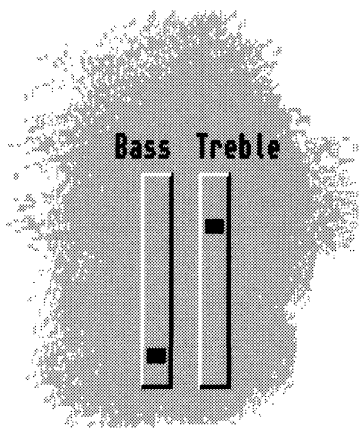


Fig. 5.14:
Slider gadgets

Sliders look similar to scrollers without arrows, but when the user clicks in the slider box above or below the slider bar, the slider value should change in single increments rather than entire views.

Keystroke Activation

The slider reacts differently to shifted and unshifted keystrokes. An unshifted keystroke should cause the horizontal bar in the slider to move up. A shifted keystroke should cause the horizontal bar to move down. When the horizontal bar reaches either the top or the bottom it should change directions automatically. Don't tie the direction solely to a shifted keystroke.

Text Gadget

Text gadgets (sometimes referred to as string gadgets) are rectangular boxes used to accept keyboard input for alphanumeric fields.

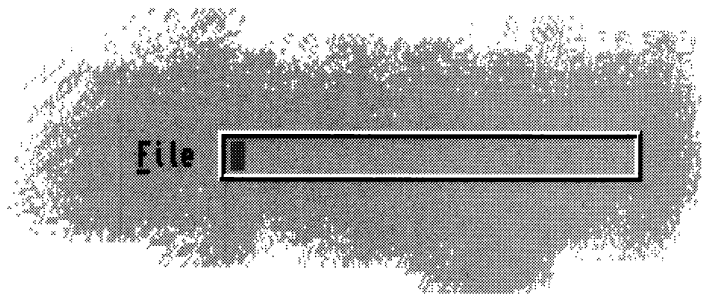


Fig. 5.15:
A text gadget

Activating Text Gadgets

When a window or requester appears containing a text gadget, have the gadget immediately activated and ready for keyboard input if:

- there are no gadgets on the requester that can be activated by the keyboard;
- the window or requester appeared in direct response to user activity (i.e., it is not asynchronous).

If the window or requester has other gadgets that can be activated from the keyboard, the user should also be able to activate the text gadget from the keyboard. In this case, it should not be activated by default.

The user should not have to select the text gadget with the mouse before typing in an entry (if she chose an action that directly called that requester up). It may seem like a small delay to go from keyboard to mouse to keyboard, but all breaks in the flow of your program should be minimized.

Ordering of Text Gadgets

When a window contains a series of text gadgets, activate the gadget in the far upper left region of the window first (depending on the scanning direction of the local language).

Moving Through Fields

Let the user move through fields with the Tab key.

Let the user move through fields with the Tab key. When the user presses Tab, activate the next text or number gadget in the series. When the user reaches the end of the series and presses Tab, the cursor should return to the first entry gadget. This function is supported by Intuition.

Shift-Tab should activate the previous gadget in the series.

Display Box

A display box is a rectangle that shows non-editable textual or numeric information.

Display boxes look similar to text gadgets, but since they are read-only, they appear recessed. This follows the 3-D conventions given in Chapter Two.

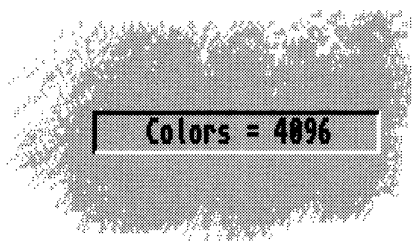


Fig. 5.16:
A display gadget

Icon Drop Box

Icon drop boxes are used for manipulation of icons. If a user drags an icon to an icon drop box, the image of the icon is copied into the box. Depending on the function assigned to that icon drop box, the icon imagery may then be redrawn or used to represent other files.

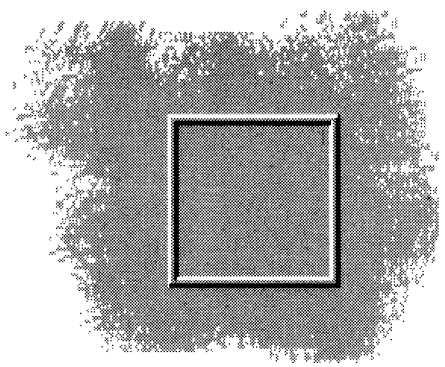


Fig. 5.17:
An icon drop box

Custom Gadgets

Sometimes a system-supported control does not provide the exact function you need. In that case, you may choose to create a custom gadget.

If a custom gadget is an extension to an existing gadget type, then you should try to emulate the features of the existing gadget.

For example, a custom multi-line text gadget should support the same keyboard functions that the standard single-line text gadget does. Right-Amiga-X should erase the text and Right-Amiga-Q should undo changes. Refer to Chapter Ten for more information on the keyboard conventions used in text gadgets.

All custom gadgets should support a minimum of three images: normal, selected and disabled. The gadget images must be usable on a monochrome screen. .

Custom Icons

Custom icons are application-specific gadgets that can be moved and manipulated. The note object in a music package is one example.

Take care when using custom icons alongside action gadgets that have graphical labels. Since it is not clear which symbols trigger an action when clicked and which symbols move when clicked, users could be triggering unwanted actions. If you use both in the same application, be sure the user can tell at a glance which is which. Grouping them spatially is one possible solution. Color cues may provide a solution. Boxing them may be a third solution. ▲

chapter six
MENUS

Remove not the ancient landmark,
which thy fathers have set.
- Proverbs

Menus are used either to invoke an action from a fixed list of choices or to set an option within an application.

The Amiga's menu system is a very appealing part of the user interface for a number of reasons:

- Even novice users can quickly understand how a menu works.
- Menus allow the user to browse through the set of possible actions that can be performed. This gives an outline-like overview of what functions are offered by a program.
- They give the user familiar landmarks that can always be brought into view by pressing the menu button.
- The menu system keeps the Amiga's GUI from being too cluttered since menus stay neatly tucked away until the user wants to see them.

Menu Anatomy

The menu system consists of the menu bar which shows the name of each menu, a drop-down menu under each name showing the list of choices (also known as menu items), and an optional group of secondary choices called a submenu, which are attached to a menu item. Submenus should appear to the right of the menu.

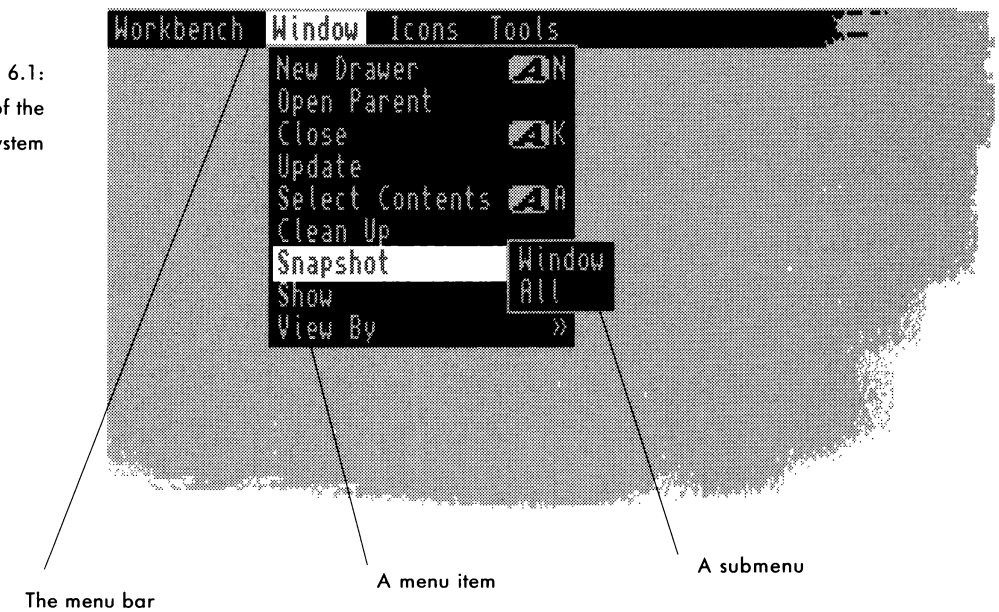


Fig. 6.1: Elements of the menu system

GENERAL RULES

Throughout your design of a menu system, keep the user in mind. Think of what the user will have to go through to choose a menu item. Working with a mouse is simple for simple tasks, but when menus get long or are complicated and illogically arranged, working with a mouse can be very frustrating indeed.

Strive for Fixed Menus

When possible, your menus should offer a fixed set of visible choices. A large, variable set of choices such as fonts can make a menu unwieldy. Usually, a variable set of choices should be presented in either a requester or a scrolling list gadget. There are rare exceptions such as the standard User menu described later in this chapter.

Multiple Selection

The user should be able to select a number of menu items at one time by clicking the multiple items with the selection button before releasing the menu button.

Anticipate conflicting choices that can occur during multiple selection. If the user chooses an item and then chooses one that conflicts with the first, the second item should override the first.

Let Intuition Work for You

Many of the functions described here can be handled for you automatically through Intuition. Let Intuition do the work for you whenever possible rather than coding your own menus.

MENU DESIGN

Use the same criteria for designing the menu system as other elements of the GUI: design it on a 640 x 200 screen with the Topaz 8 font. At run-time, your application should test to see if the menu will fit in the user's preferred font and screen size. If not, revert to the Topaz 8 font.

Colors

When possible, menus should use dark text on a light background. Note: the examples shown in this chapter don't follow this rule because the version of Release 2 they were taken from had color constraints that prevented this rule from being obeyed.

Font

A uniform font is recommended for all the menu items within an application. A change in font style could be appropriate, however, if it symbolizes a change in the style of text in your application. For example, in a word processor, a menu called Style may list the type styles Bold, Italic and Underline. Those menu items may be rendered in bold, italic and underline text, respectively.

Toggle Items

Some menu items turn an option on or off. To show this, use a check mark that toggles visible and invisible in a space in front of the item. Don't add a submenu with items such as "on" and "off."

Toggle items should be indented to allow room for the check mark. This will give the user a visual cue that it is a toggle item.

Fig. 6.2: A toggle menu item shown in both states

Organization

Within a menu, items should be grouped according to *function*. Distinct groups should be separated with a separator bar. "Function" can be defined by these three rules:

Separate the toggle items from the non-toggle items. This will help the user quickly distinguish one type from the other. It will also look better if the left-hand side of the menu text doesn't indent more than once.

When ordering menu items, make sure to separate commonly used items from dangerous ones.

Group similar choices together. For example, if a menu includes the items

- Save as ASCII
- Save as MystrEd doc
- Quit
- New
- Save as IFF clip

the three Save as... items should be grouped together.

Frequently used items should be placed towards the top of the menu. When ordering the items make sure to separate commonly used items from dangerous ones. It's relatively easy for a user to choose the wrong menu item by mistake – try to anticipate such mistakes and their possible outcomes and arrange your menus to avoid them.

Limit the Size

Try to limit the number of items in a menu to about a dozen. Submenus should have about a half dozen items at most. The utility of a menu decreases as its length increases.

On the Menu Bar

When it comes to ordering the menus themselves on the menu bar, remember that the user can most easily access the outside menus. Put menus that are used less towards the middle.

Follow the order given in the Standard Menus section of this chapter for any standard menus.

Ghosting

Whenever a menu or menu item is inappropriate or unavailable for selection, it should be ghosted. Never allow the user to select something that does nothing in response.

Fig. 6.3: A menu with ghosted items

Labeling

Menu and menu item labels should be terse – preferably one to three words. Use the capitalization rules that apply in the language at hand.

Menu items that represent an action should reflect that action in the item's label. For instance, "Print" is better than "Printer." Try, also, to keep the terminology user-friendly and non-technical.

Don't repeat a menu's title on every item's label. For instance, in a Macros menu use the label Load... rather than Load Macros... Repeat the menu title, however, if user comprehension will suffer.

Toggle Items?

As discussed earlier, indenting toggle items lets the user know that they

represent a choice. Another cue you can give to the user is to end the label in a question mark. In some Settings menus, for example, an item labeled "Create icons?" is preceded by a check mark (or placeholder) and ended with a question mark. Both are cues to the user that this is a toggle item.

Ellipsis

When a menu item brings up a window or requester, an ellipsis (three dots) should be appended to the menu item's label.

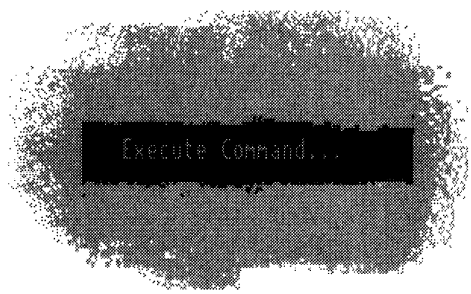


Fig. 6.4:
A menu item that brings up a requester or another window

Submenus

When an item has a submenu, the symbol » should be placed on the far right side of the menu item. These should be flush right (see the next section).

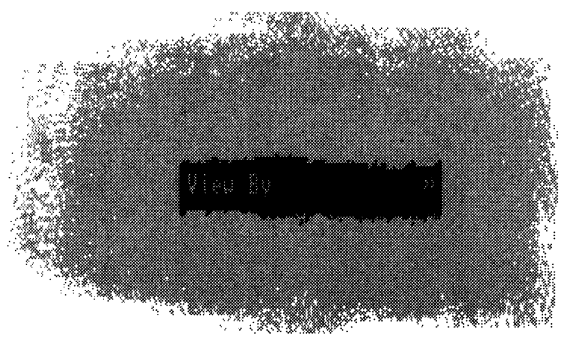


Fig. 6.5:
A menu item that brings up a submenu

Lining Up on the Right

If you have any symbols or text that you want to line up on the right side of the menu item, make sure it is flush right (i.e., right justified) or it won't line up properly when used with a proportional font.

STANDARD MENUS

Some menus and menu items appear so often in applications that their use and design can be standardized. Using standard menus will enhance the feeling of consistency across applications and help the user feel more comfortable with your program.

Listed below are standard menus that your application should use whenever appropriate. The order in which the menus and their items are presented is an important part of the specification.

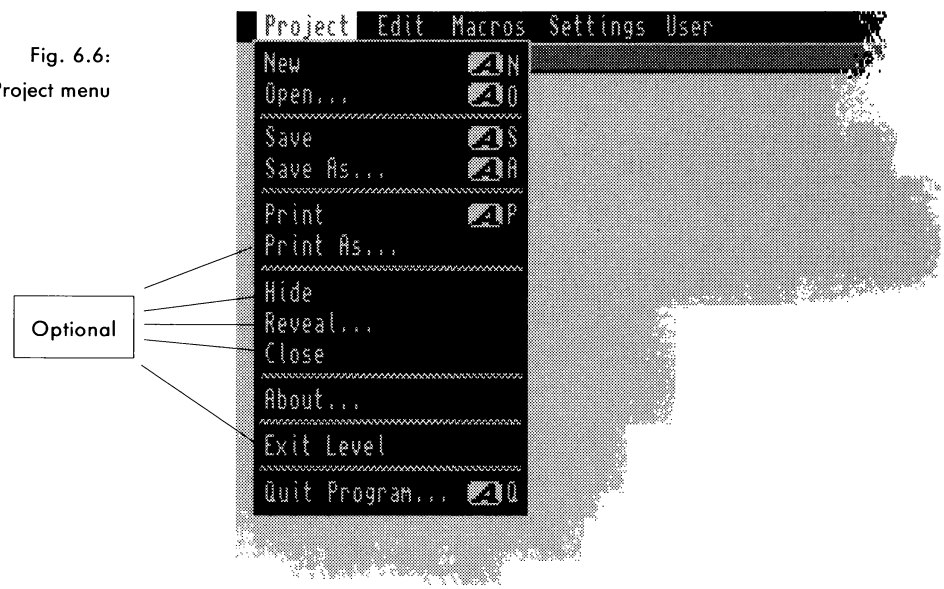


Fig. 6.6:
The Project menu

Project Menu

Applications that create, edit and save data should have a Project menu. The Project menu should be the first menu on the left side of the menu bar and should contain as many of these items as appropriate – in the order shown.

Print and Print As... are optional items for applications that support printing. Hide, Reveal... and Close are optional items for multiple-project applications. Exit [Level] is an option for multi-level programs.

New

New should open a blank, untitled project.

Open...

Open... should bring up a file requester.

If an unsaved project currently exists, the modified project requester (Save project?) should be brought up when Open... is selected – unless your application supports multiple projects.

Save

If the project had been saved previously, Save should copy the current version over the old version.

If the project has never been saved, Save should bring up a file requester.

Save As...

Save As... should bring up a file requester to prompt the user for a name, and then save the file.

Print

Print should send the project to the printer using the current settings. The current settings may either be those found in Workbench's Prefs directory or individual program settings previously saved by the user.

Print As...

Print As... brings up a requester that can be used to set your program's print options.

Hide

Hide, Reveal..., and Close... are optional items for multi-project applications. Exit [level] is an option for multi-level programs.

This menu item is for programs that can handle multiple projects at one time. Hide should remove a project's window from the screen. If the user selects Quit Program before returning to a hidden project that has unsaved changes, the modified project requester should come up asking if she wants to save the project first. A separate modified project requester should be presented for each unsaved, hidden project.

Reveal...

Reveal is for programs that can handle multiple projects at one time. Reveal... should bring up a requester with a scrolling list gadget containing all opened projects whether hidden or not. Although its true purpose is to return hidden projects to the screen, it is more user-friendly to allow the user to view all open projects from this item. The chosen project's window should open and jump in front of any other windows. If no projects are hidden, this item should be ghosted.

Close...

This menu item is for programs that can handle multiple projects at one time. Close... should remove the current project's window from the screen. If there are unsaved changes, a Modified Project requester should come up first.

About...

About... should bring up a window with information about your program and the current project. What information is included is up to you, but it should include at least the version number of your software. Some suggested information to provide: ARexx port name, project size, tool name.

Exit [Level]

Exit [Level] should allow the user to leave the current level, such as the slide editing level in an electronic slide show program, and return to the next highest level. The word [level] on the label can be replaced with the name of the current level.

Quit [Program]...

Quit [Program]... should exit the entire program. The word [Program] can be replaced by the name of your application.

A separate modified project requester should come up for any open or hidden projects with unsaved changes.

Quit [Program]... should not be modal; i.e., it should be available at any level.

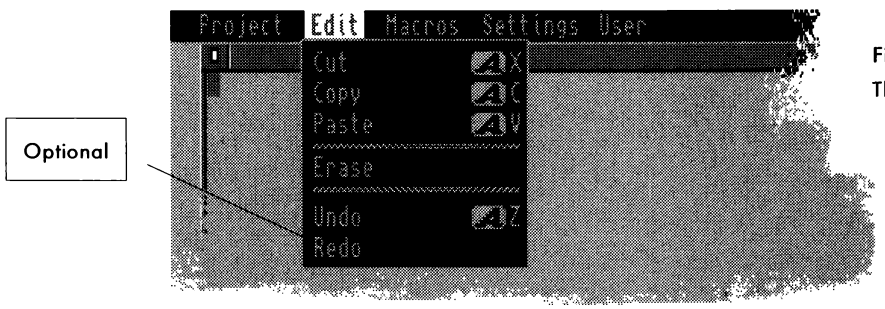


Fig. 6.7:
The Edit menu

Edit Menu

Any application that can perform editing functions should have the Edit menu in the second position. The following items represent the suggested list with Redo as an optional item. A basic rule is that any menu item that operates on blocks of data should go in the Edit menu.

You should support the Clipboard as your buffer for cut-and-paste operations. This will allow the user to share the clips between applications. See the chapter on Data Sharing for more information on the Clipboard.

Cut

Cut should remove the highlighted text or current object and place it in the buffer.

Copy

Copy should retain the highlighted text or current object in the project while also placing a duplicate of it in the buffer. The text should then be unhighlighted to show that the copy took place.

Paste

Paste should take whatever is in the buffer and place it at the insertion point in the project window.

Erase

Erase should remove the highlighted text or current object.

Undo

Undo should restore the project to the state it was in before the user's last action. If, for instance, the user imported a text clip by mistake in a word processor, Undo should automatically take it out.

Some programs support multiple steps of Undo. Using the example in the previous paragraph, say the user had deleted a paragraph before inserting the text clip. Selecting Undo twice should take out the mistaken text clip and return her original paragraph.

Redo

Redo is an option for programs supporting multiple steps of Undo. In a program that doesn't support multiple steps of Undo, hitting Undo twice would take the user back to where she was before she selected Undo – it's cyclical. She imports a text clip, selects Undo and the text clip is gone. Select Undo again and the text clip is back again. In this program, Undo undoes Undo.

But in applications with multiple steps of Undo, the user can undo anything except Undo. She just keeps going back another step instead of going through cyclical actions. So Redo is there to undo any mistaken Undos.

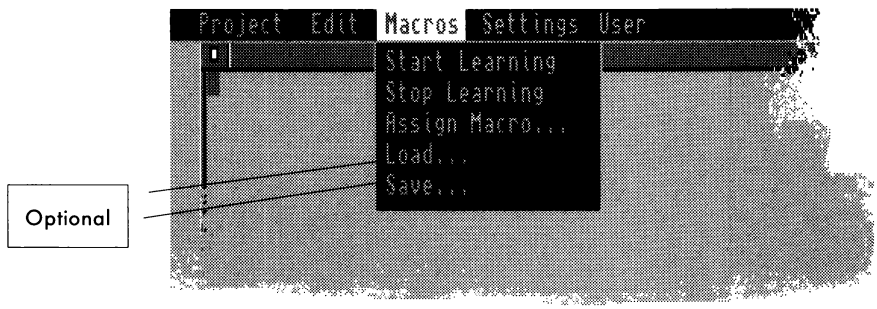


Fig. 6.8:
The Macros menu

Macros Menu

If your application supports macros, it should use ARexx, the Amiga's built-in scripting language, rather than its own internal scripting language. See Chapter Nine for more information on ARexx.

The Macros menu should allow the user to create and assign macros. Optionally, you can include the ability to load and save macros as well.

Start Learning

When Start Learning is selected, all subsequent user actions should be written in ARexx script form to an internal buffer – until Stop Learning is selected.

Feedback to the user is essential, so it would be a good idea to let the user know that the Start Learning command was accepted. The indication can be subtle but should be noticeable if the user wants to look for it. One idea is to put a small glyph, letter or word in the project's title bar.

Stop Learning

When Stop Learning is selected, your program should stop writing user actions to the internal buffer created by Start Learning.

Any visual feedback that originated with Start Learning should go away.

Assign Macro...

Assign Macro... should bring up a requester through which the user can assign the script created by the previous two commands.

Macros can be implemented a number of ways. One longtime favorite is to assign the macro to the function keys, the ten keys across the top of the keyboard, or to some other keyboard combination. Or the macro can be assigned to the User menu (see section later in this chapter). Or the macro can be given a name and saved to disk; from there it can be implemented via an ARexx command console. Some programs may choose to support one, some or all of these methods. Your Assign Macro... requester should reflect all the possible choices your program supports.

Load...

Load... should bring up a requester listing the macros that can be loaded into memory. This, along with Save..., forms an optional set of items.

Save...

The macros design as discussed above assumes that key- and menu-activated macros will be assigned and used on a per-session basis. The next time the user returns to your program, the macros he had assigned to keys and menus will no longer exist.

Save... should allow the user to name a macro and save it to disk. At another session, the macro could be loaded and then assigned to a key or menu. This, along with Load..., forms an optional set of items.

Otherwise, if the user wishes to save the macros and their assigned positions, he should do it through the Save Settings item found in the Settings menu.

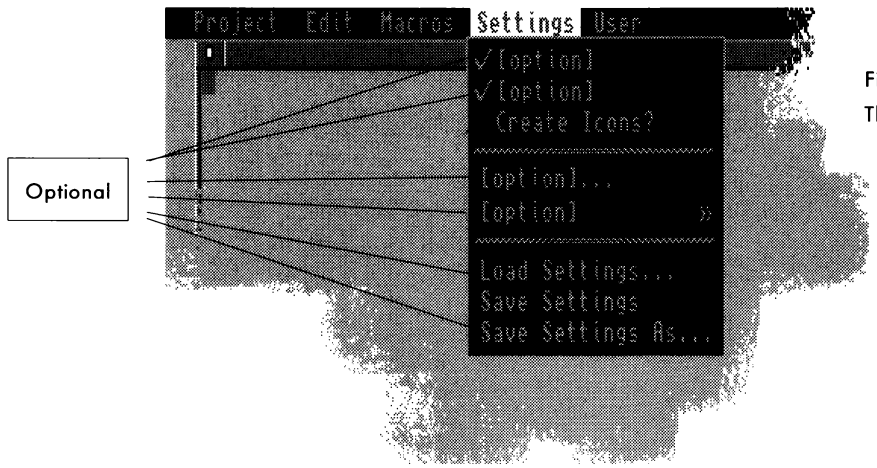


Fig. 6.9:
The Settings menu

Settings Menu

Provide a Settings menu if you allow the user to change and save aspects of your program's environment. This menu should be located in the second position from the right.

Consult Chapter Twelve for more information about preferences in general.

Two items should be accessible through this menu (if your program supports them): a toggle item asking if icons should be created and an overall Save Settings item. They may be directly in the menu or in a requester brought up by a menu item.

[options]

Listed in this menu should be any specific options you may wish to handle through a menu item, such as what screen resolution to operate in or baud rate for a terminal package. Or you could have an option labeled Set Settings... that would bring up a control panel which could be used to set all the program's options.

[Options] can be toggle items, or they can bring up submenus or requesters – it all depends on your application's needs.

Whatever these options are, they would change the user's preferences for that session only. To save them as defaults the user would have to use one of the Save Settings items.

Note: the word [option] shown here is only a placeholder to be replaced with a descriptive menu label.

Save Settings

Save Settings should save the user's settings as your application's default preferences. Each time your application is opened thereafter, it should use these user-specified settings.

By default, Save Settings should save the new settings in the current settings file. See Chapter Twelve, *Preferences*, for more information on saving settings files.

Save Settings As...

Save Settings As... is an optional item that brings up a file requester. Save Settings As... will be used when the user wishes to save multiple settings files.

There are a couple of reasons why this option may be helpful to users. A user may save different settings for different types of projects; e.g., settings.vid for projects being output to video and settings.prt for projects being output to a printer. Also, if your application is being used on a network, users may wish to save personal settings.

Load Settings...

Load Settings... is an optional item that should allow the user to change from her current settings to a new set of previously-saved settings.

Load Settings... should be included if you think your users will want or need multiple settings files.

Create Icons?

Create Icons? is a toggle item that should allow the user to choose whether a .info file will be created with each project. A .info file contains a graphic image for the file's icon. Users who don't use the GUI as a filing system may find .info files to be extraneous and a waste of storage space. Your program should provide this menu item.

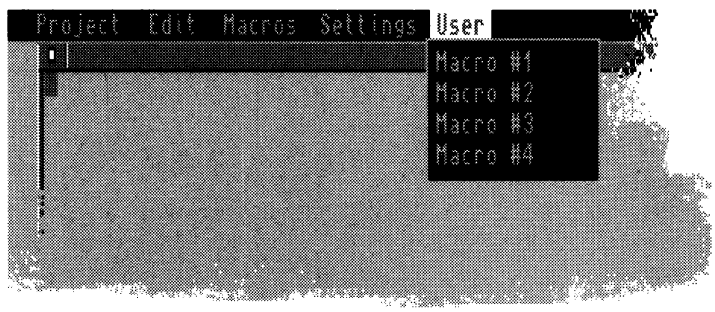


Fig. 6.10: A sample User menu

User Menu

The User menu is a variable-length menu that contains user-defined macros and any other optional features a user may want to add as a menu item. This should be located at the far right. ▲

chapter seven

WORKBENCH

To understand a man is to afford
him immense satisfaction.
The moment he is understood
he begins to feel comforted.
- Jean Guibert

In a way, Workbench is just another program – but as the default interface on the Amiga, it's one program that many users will pass through on their way to your application. Although it has other facets and functions, it's the interface between your program and Workbench that is of concern in this chapter.

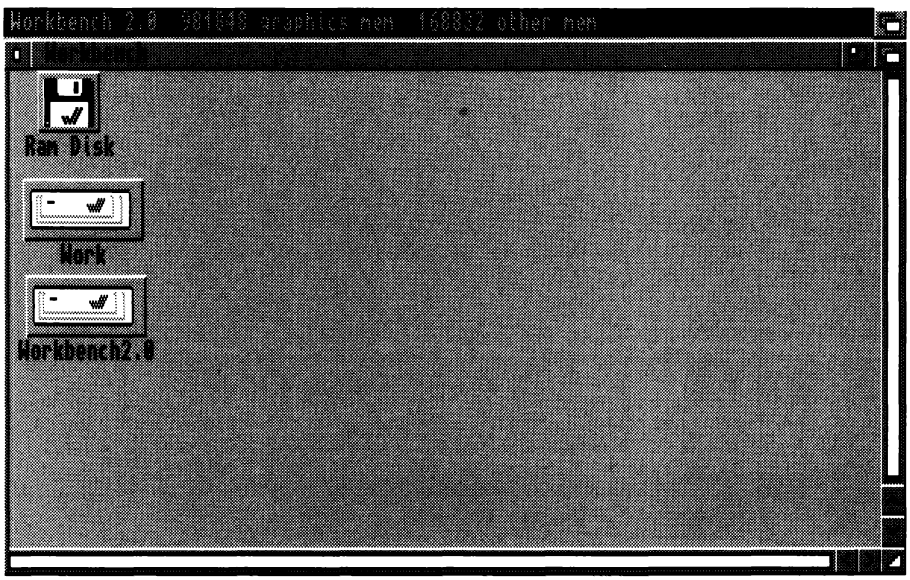


Fig. 7.1:
The Workbench
screen

ICONS

Icons are pictorial representations of directories, files, applications, objects or actions. Your program should have icons for anything the user can access including the main program itself, documentation files, and any tools that may accompany the program.

The .info File

The icon imagery is found in a file bearing the suffix ".info". For example, the icon for a data file called "myletter" would be found, along with some other information, in the file "myletter.info".

Icon Design

A good icon quickly communicates the function it represents. The calculator icon and the SetMap icon are good examples.

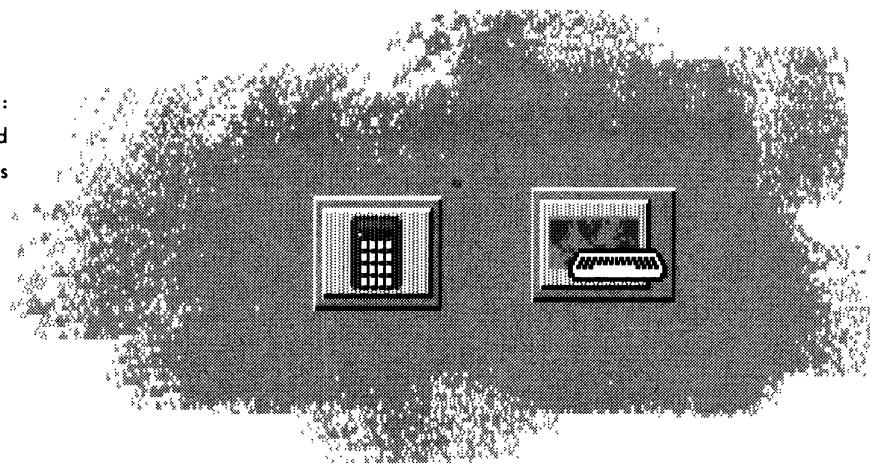


Fig. 7.2: The Calculator and SetMap icons

Size

Icons should be small. The maximum recommended size for an icon is 80 pixels wide by 40 pixels high. Large icons take up valuable screen and disk

space, make for an unprofessional Workbench look and, in general, can be just plain annoying.

The 3-D Look

Icons should be designed with the light source coming from the upper left-hand corner. They should be viewable in one bitplane as well as two.

Text in Icons

Before you use words in an icon, think about how extensive a distribution you would like to see your product achieve. It's better to have an icon communicate through symbolism than language.

Before using words in an icon, think about how wide a distribution you envision for your product.

Icon Types

There are five types of icons on the Workbench: disk icons, drawer icons, trashcan icons, tool icons and project icons. Of these, only tool icons and project icons bear discussion here.

Tool Icons

A tool icon represents an executable file such as an application. A double-click on a tool icon will run the application. The look of tool icons varies from application to application.

Project Icons

A project icon represents a data file. Typically, a double-click on this icon will cause the application that created the data to run and automatically load this data.

The look of a project icon is usually determined by the application that created it and by the type of data it represents. However there is also a default project icon included in the system.

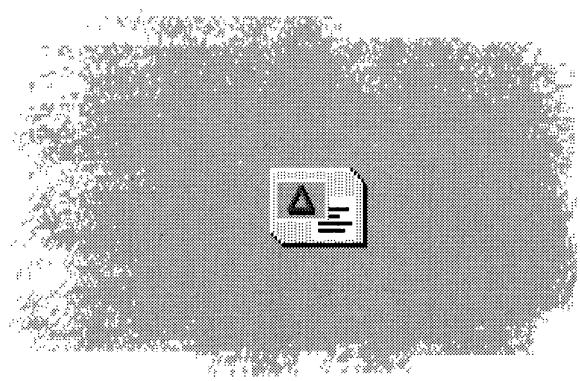


Fig. 7.3: The default project icon

Create Icons?

When your application creates a data file, it should, by default, create a .info file to go with it. Many users are accustomed to accessing their projects via an icon.

Create the icon after the user successfully saves the project. This will prevent the possibility of project-less icons in the system.

Your application, however, should allow the user the option of saving files without creating icons. The recommended way of doing this is to have an item in the Settings menu called Create Icons?. This item should be enabled by default. See Chapter Six for more information.

Positioning

New project icons should never be saved in a specific position. Rather they should be positioned algorithmically by Workbench (see the Workbench flag NO_ICON_POSITION). In the same vein, if the imagery of an existing icon is changed, (e.g., the application creates a reduced version of the project for use as the icon imagery) the icon should be saved without a specific position.

ARGUMENT PASSING

To make the common operation of starting up programs more powerful, the Amiga provides ways of passing along more specific information to the application. This process of providing additional information to a program that's about to run is known as argument passing.

When using the Shell, the system is straightforward. The user runs an application by typing its name along with additional information such as the name of a file to operate on and any command options. The entire command line is then passed to the program as an argument.

The Workbench also provides means of passing arguments – but it is tied to icons instead of lines of type.

Tool Types and Default Tool

Applications started from Workbench receive startup information from the system in the form of a WBStartup message. The WBStartup message can be used to get various Workbench arguments including those found in the Tool Types and Default Tool fields of the icon the user clicked on.

Tool Types

If you click once on any tool icon and choose "Information..." from the Workbench's Icon menu, you should see a requester with a field for Tool Types.

Arguments placed in this field take the form KEYWORD=value. KEYWORD is an argument name specified by your application, and value is a number or string that should be assigned to that argument.

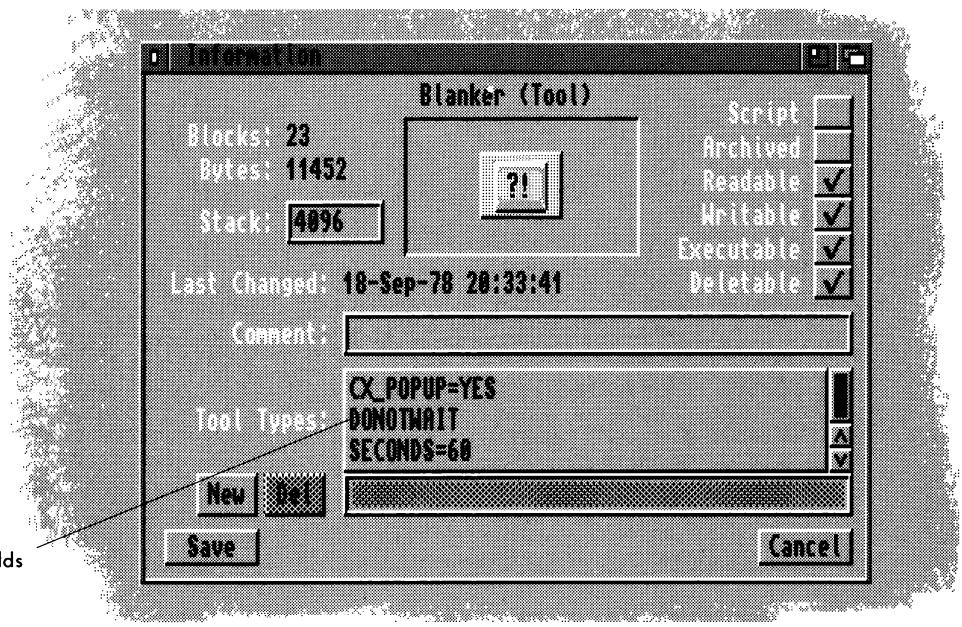


Fig. 7.4: The Information requester for a tool icon

For instance, the Tool Types field of a text display program might be set to LINES=20 to indicate how many lines to display at once.

Normally, the Tool Types information is filled in when the .info file is created; subsequent operations only read this information.

Default Tool

The Information requester for a project icon is different than that of a tool icon – it has both a Default Tool field and a Tool Types field. The Default Tool field tells the system which application is used to edit the project. When a user double-clicks on a project icon, the application specified in the Default Tool field gets run and the data is passed to that application via the WBStartup message.

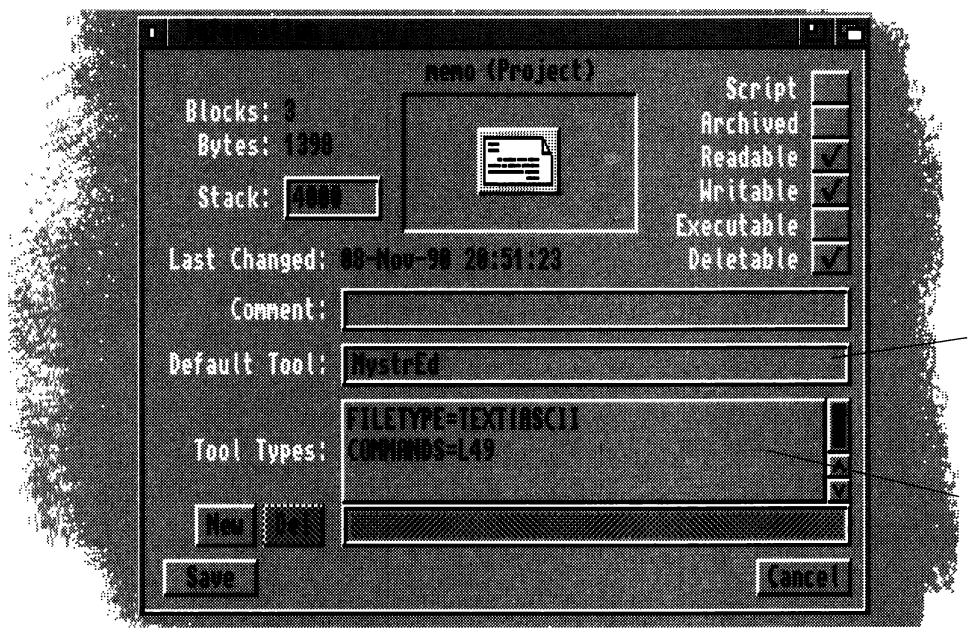


Fig. 7.5: An Information requester for a project icon

The Default Tool field

Tool Types fields

Altering Icons

If your application needs to alter an icon for some reason, change only the things you intend to change and preserve the rest. If your application needs to change the Tool Types field, for instance, leave the imagery, position and Default Tool field alone.

More important, change only the Tool Types entries relevant to your application – do not rewrite Tool Types from scratch for an existing project. If the icon

has a Tool Types entry that your application does not recognize, that entry should be preserved.

Tool Types and Networks

You should support project-specific Tool Types arguments. With the advent of networked Amigas sharing the same applications, this becomes increasingly important. Tool Types arguments written to specific projects will allow networked users to override Tool Types arguments written to the application's tool icon – arguments that have probably been set for the least common denominator.

Standard Tool Types Arguments

Some Tool Types arguments are already used by the system. If they apply to your application, support them; if not, take care that your Tool Types arguments don't conflict with these. Listed below are some of the more common Tool Types arguments found in the system:

Take care that your Tool Types arguments don't conflict with those listed here.

WINDOW=CON:<window spec>	
DONOTWAIT	*Don't wait for return; used by wbstartup*
TOOLPRI=<priority>	*Used to set your program's priority*
STARTPRI=<priority (-127 to 128)>	
PUBSCREEN=<name>	*The name of the public screen to open on*
STARTUP=<name>	*ARexx script to run at startup time*
PORTNAME=<name>	*Name to assign to your application's ARexx port; overrides default naming system*
SETTINGS=<name>	*Allows a user to specify a settings file*
UNIT=<number>	*Device or unit number*
DEVICE=<parallel/serial>	*The name of the device to use*
FILE=<file pathname>	
WAIT=<number (of seconds)>	
PREFS=<prefstype>	
CX_POPUP=<yes/no>	

CX_POPKEY= <CX key specifier >

CX_PRIORITY = <CX priority level>

<CX fkey spec> = <CX string spec>

The Apps

Other facilities also exist in Workbench that allow an application to get more information while the application is already running. These are known as AppWindow, AppIcon and AppMenu.

For example, a text editor's windows may function as AppWindows. A user would be able to drag the icon of a file into the window and that file would be loaded automatically.

AppWindows, AppIcons and AppMenus are aimed at the user. By using techniques that are totally graphic-oriented (Tool Types arguments still eventually come down to a line of type), the Apps bring the power of argument passing more into line with the "point-and-click" metaphor.

AppWindows

An AppWindow is a special kind of Workbench window that allows the user to drag icons into it. It's basically a graphical alternative to a file requester.

Applications that set up an AppWindow will receive a message from Workbench whenever the user moves an icon into that AppWindow. The message contains the name of the file or directory that the icon represents.

For instance, Workbench's IconEdit is an AppWindow with three different areas which users can drag icons into – each with a different purpose. When the user drags an icon into the large box, the icon is loaded as a project. When an icon is dragged into the box labeled "Normal", its image is used for the normal (not selected) image. Likewise for an icon that is dragged into the box labeled "Selected" – its image becomes the activated image for the icon.

An AppWindow will often use icon drop box gadgets to indicate the active area where the user may drop an icon. Note: IconEdit did not follow this con-

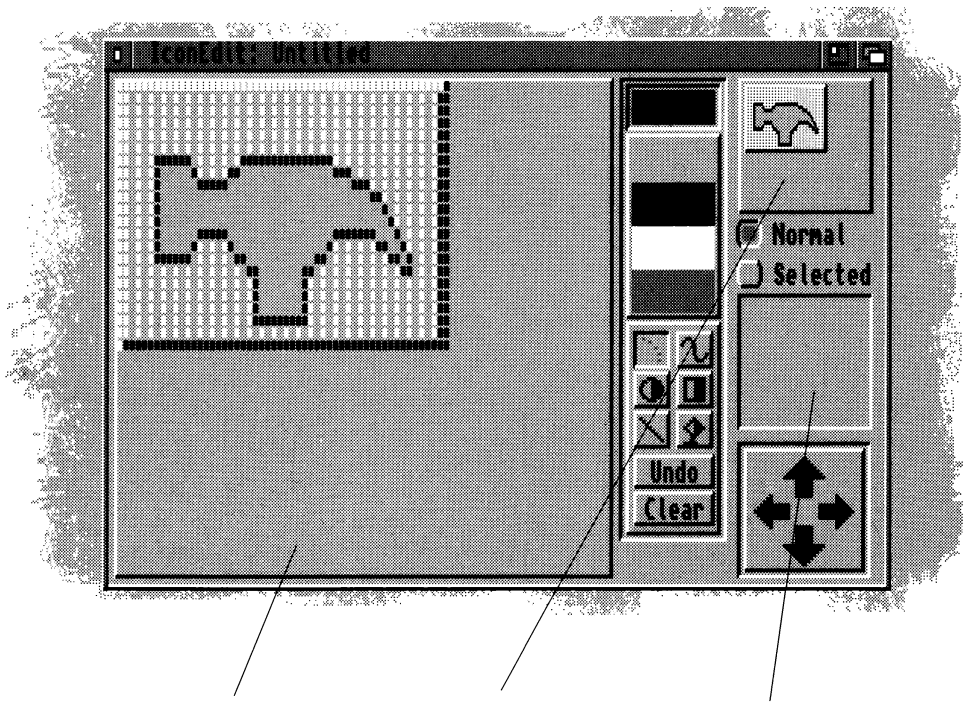


Fig. 7.6: IconEdit is an example of an AppWindow. It has three different drop areas which do three different things.

vention because it was more important to indicate the function of each area by its imagery (e.g., the "normal" area can be clicked on, etc.).

The window should activate when an icon is dragged into it.

AppWindows only work when your application is running on the Workbench screen. This makes sense because you need to be able to drag icons from Workbench to the AppWindow and draggable objects can't be dragged across screens. If the user opts to run your application on a screen other than Workbench, set your AppWindows so they will revert to AppMenus (see below).

As a general rule, AppWindows are appropriate when your application needs to have a window anyway.

AppIcons

An AppIcon is similar to an AppWindow – it allows the user to pass a graphical argument to a running application. The only difference is that AppWindows use a window to accept the argument and AppIcons use an icon.

The image for an AppIcon should give some indication what operations it supports; i.e., whether it represents an iconized application or supports dropped projects. Avoid vague imagery.

AppIcons are useful for programs that operate in the background with little other user interaction – a print spooler is a good example.

Double-clicking on an AppIcon should normally open a window offering information and controls. For example, a print spooler could open a status window in which the user could rename, abort or re-order the things he sent to be printed.

AppMenus

An AppMenu allows your application to add a custom menu item to the Tools menu on Workbench. An application that sets up an AppMenu item will receive a message from Workbench whenever the user picks that item from the Workbench menu.

Preferences

Applications can also get arguments from Preferences. Via Preferences, the user can graphically set up the system defaults to suit her taste and needs.

Through Preferences, the user can control such things as screen fonts, colors and other global information that your application should respect.

You can create a preference editor to handle the defaults used by your application. See Chapter Twelve for more information. ▲

chapter eight
THE SHELL

If my doctor told me I had only six minutes to live,
I wouldn't brood. I'd type a little faster.
- Isaac Asimov,
author of over 400 books

One of the best things about the Amiga is its versatility. While the rest of the personal computing community continues to argue about the best type of interface, Amiga users have both – a graphic interface with Workbench and a text-based interface with the Shell.

The Shell (also known as the Command Line Interface or CLI) preserves the best features of operating computers the "old-fashioned way," that is, by typing commands at a console. Although that is in some ways more difficult than clicking on graphics with a mouse, it provides a finer level of control, less overhead and greater power than is possible through a GUI.

Virtually anything that can be done through Workbench can also be done through the Shell. On Workbench, for instance, a user can move into a subdirectory by double-clicking on its drawer icon. The equivalent of this in the Shell is to type

```
cd <subdirectory name>.
```

The command *dir* will then list the contents of the directory.

Your application should support all three interfaces built into the Amiga: the GUI, the Shell and ARexx.

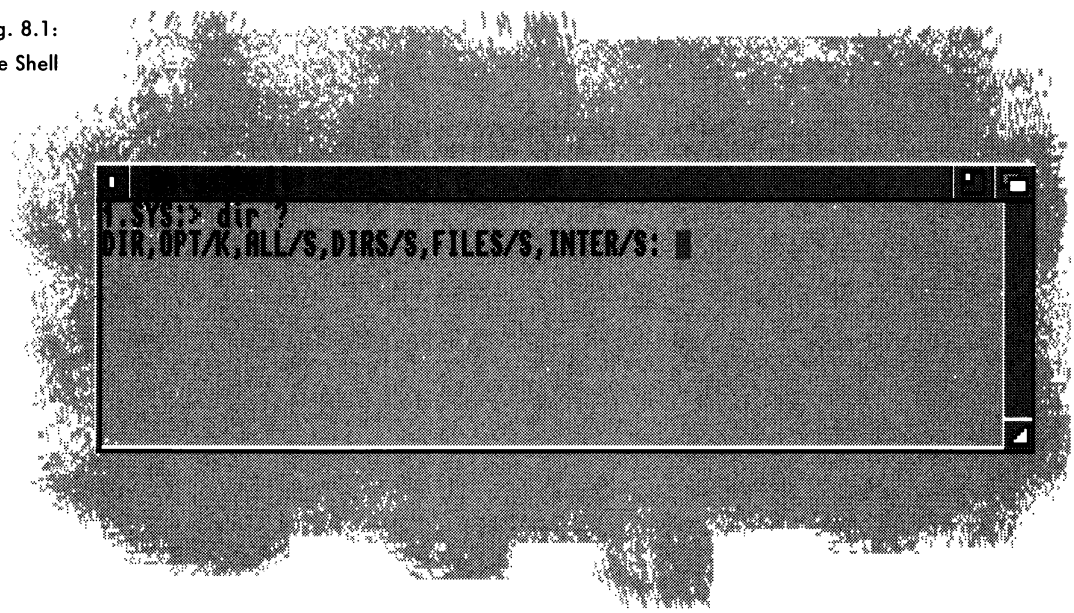
The same example can be used to illustrate the greater power of the Shell. In addition to *dir*, which gives a brief listing of files and subdirectories, the user can also use *list* which gives a more in-depth listing including the size of each file, information about the file and when it was last updated. Or you could list only a subset of files by using wildcards in the command. For example,

```
list #?.info
```

would list only those files ending with ".info".

In addition to being able to do things with finer control than Workbench, the Shell can also do some things that Workbench cannot do at all. For instance, from the Shell you can create a script – a pre-recorded set of commands that helps to automate repetitive tasks.

Fig. 8.1: The Shell



PARSING COMMANDS

You should use the command template method for parsing the command line.

Standard Form

In general, Shell commands take the form:

```
COMMAND [redirection-argument] [<argument1>, <argument2>, ...]
```

where COMMAND is the name of an executable file, [redirection-argument] is a "<", ">", and/or ">>" symbol followed by an AmigaDOS device name, and [<argument1>, <argument2>,...] is a list of arguments that will be passed to the executable file.

Here's a sample Shell command:

```
play myanim loops 20
```

The command is *play* (an executable program), there is no redirection and three arguments are passed to the play program: *myanim* (the name of the data file), *loops* (a command option) and *20* (another command option).

Built-in Parsing

The Amiga, like Unix and many other systems, passes command line arguments to your application. The system automatically provides the list of arguments and counts them for the application.

The system also needs to parse these arguments. Parsing refers to the job of examining the arguments to find out what they mean so that the appropriate operation can be performed.

Prior to Release 2 of the Amiga operating system, each application designer

had to manage his own parsing. This resulted in a variety of command styles and a more complicated Shell environment for the user. So the designers of Release 2 built a standard way of processing Shell arguments into the system.

When your application is started from the Shell, any arguments should be parsed using the command template method – the same method used to parse all the system-supplied Shell commands. This argument parsing method should also be used by your ARexx or scripting commands (see Chapter Nine for more information).

Using AmigaDOS routines to handle command line parsing can shorten both your code and your development time.

By far, the greatest benefit of this method is that it allows you to use AmigaDOS routines to handle the chore of parsing the command line. The DOS routines will handle errors and give help messages so both your code and your development time can be shorter. Using standard argument parsing also makes the Shell interface more consistent and more comfortable for the user.

The Command Template

In order for your application to use standard argument parsing, a command template must be constructed to describe the arguments that the command understands.

In the command template, each argument is specified by a keyword (a preset argument that the program understands) followed by a modifier that describes the properties of the argument. Modifiers take the form "/X" where X is one of the characters from the table below. Each keyword can have none or many of the modifiers.

Neither the keywords nor the modifiers are case-dependent. The format shown here corresponds to the way the system displays command templates in the Shell.

,(comma) - No arguments

The comma indicates a null argument. It also separates arguments in the command template.

= - Equivalents

The equal sign indicates equivalent abbreviations for keywords, e.g., PS=PUBSCREEN/K.

/A - Always required

The argument must be supplied in order for the command to be accepted.

Use the standard command template described here in your Shell and ARexx commands.

/F - Final argument

If this is specified, the entire rest of the line is taken together as a single argument, even if other keywords appear in it.

/K - Keyword required

This means the actual keyword must be typed on the command line along with its argument in order for the argument to be processed (often the keyword is optional). The argument will not be interpreted unless the keyword appears.

For example, if the template is *NAME/K*, then unless *Name=<string>* or *Name <string>* appears in the command line, the command will be interpreted as having no Name argument at all.

/M - Multiple argument

There can be a number of instances of this argument. For example, the AmigaDOS Join command lets you merge together any number of files into a single file. The template is:

```
FILE/M,AS=TO/K/A
```

This allows commands such as:

```
Join file1 file2 file3 as bigfile
```

When the /M modifier is the first argument in a template, any number of filenames may be specified by the user.

If a command line has any leftover arguments, they will be interpreted as belonging to the /M argument. So only specify one /M per template. For example, in the command *Join one two as bigfile three*, the word *three* is an extra argument. In this case, one, two and three will be merged together to form bigfile; the extra argument is tacked onto the /M arguments.

The /M argument also interacts with the /A argument. If there aren't enough arguments given to fill all the /As, then part of the previous /M argument will be used to fill in the /As.

/N - Number

This argument is a decimal number. If an invalid number is specified, an error message will be returned to the user. Unless the /N modifier is specified, all arguments are assumed to be strings.

/S - Switch keyword

This modifier indicates a switch keyword argument. If the keyword is given, the switch is "on." If the keyword isn't given, the switch is "off."

/T - Toggle keyword

This is similar to /S but, when specified, it causes the switch value to toggle from "on" to "off" or vice-versa.

Displaying the Command Template

When a user types a command name in the Shell followed by a space and a question mark, the command template should be shown to him. This acts as a sort of help message that gives the syntax of the command. For instance, in the above example of *play myanim loops 20* the command is *play.* When the user types *play ?* in the Shell he should see this:

```
ANIM/A,LOOPS/K/N
```

This means the command takes two arguments: the Anim argument and the Loops argument. The Anim argument consists of the optional keyword *ANIM* followed by an animation filename. The /A modifier means this argument must always be given.

In the Loops part of the argument, the /K means that the keyword *LOOPS* is required for the argument to be processed correctly, and the /N means that the *LOOPS* keyword should be followed by a decimal number. Notice that since the Loops argument does not have a /A modifier, it could be left out altogether.

Other possible commands a user could come up with include:

```
play myanimfile loops 20
play myanimfile
play anim myanimfile
```

These commands, however, would be illegal:

`play myanimfile 20` (the required keyword LOOPS is missing)

`play anim` (no filename is given)

`play anim myanimfile loops` (the numeric value for LOOPS is missing).

STANDARD ARGUMENTS

The following Shell arguments are standard within the system. Don't use these names for application-specific arguments with purposes different than those given below.

FILES/M	*List of files to work with as projects*
PUBSCREEN/K	*Name of the public screen to open on*
PORTNAME/K	*Name to assign to the ARexx port*
STARTUP/K	*ARexx script to run at startup time*
NOGUI/S	*Indicates that no GUI is desired*
SETTINGS/K	*Name of the preferences file to load at startup*

If you need a keyword for your application, try to find an applicable one that is already in use – and use it the same way. If you need to make a new keyword, don't make it the same as a common command name; e.g., "list."

PATTERN MATCHING

Pattern matching is a powerful feature that enables more efficient operations.

When most people think of pattern matching, they think of using wildcards, but pattern matching is actually much more. It's a powerful feature that enables more efficient operations, especially on systems like the Amiga that have a hierarchical filing system with few size limits.

Pattern matching is part of the appeal of the Shell. It would be difficult to use icons to perform an operation on 50 files at once – not to mention the problem of fitting all the icons on a 12-inch monitor. Some complex operations are just easier to do with a text interface.

standard available tokens	
?	Matches a single character. For instance *a?* matches any string with two characters that starts with the letter *a*.
#	Matches a subsequent expression 0 or more times. For instance, the pattern *#?* will match any string.
(ab | cd)	Matches any one of the items separated by |.
~	Negates the following expression. For instance the pattern *~x?* will match any two letter string except those starting with *x*. Another corollary of this is the pattern *~(x?)* which will match anything except two-letter strings beginning with x; i.e., xaa or ab but not xa.
[abc]	Character class; matches any one of the characters in the brackets.
[a-z]	Character range (must be within character classes). You can also use [a-e,x-z] style.
%	Matches 0 characters always. For example, the pattern *(foo | whiskey | %)bar* matches foobar, whiskeybar and bar.
*	Synonym for #?. Not available by default on Release 2, but it is an option that can be turned on.

Expression in this table means either a single token or character (such as ? or x), or an alternation (such as (ab \| cd \| ef)), or a character class (such as [a-zA-Z]).

The Amiga's Wildcard

On the Amiga, the wildcard is *#?* – as opposed to * on other platforms. For example, the command

```
delete #?.info
```

would delete all the files in the current directory that end in .info.

Within Applications

In addition to its usefulness on the filing system, pattern matching is also useful within applications. For example, the Find function within a text editor should be able to find strings according to a pattern.

Your application should support pattern matching from the Shell and from functions within the application itself, through AmigaDOS system routines.

EMBEDDED VERSION IDs

Follow the standard method for indicating the version of your software.

The Shell command VERSION provides support for version identification, but you have to follow a standard template.

A text sequence that indicates the current version of your application should be embedded within your code. (It should also be put in your ARexx scripts and your configuration files.) The format of this text sequence is:

```
$VER: <name> <version>.<revision> (<d>.<m>.<y>)
```

<name> *The name of your application*
<version> *Major version number*
<revision> *Minor revision number*

<d>	*Day created*
<m>	*Numeric month created*
<y>	*Year created*

If you follow this template, the user can find out what version of your software he is using with the Shell command VERSION. ▲

chapter nine

AREXX™

The "space" of all possible programs is so huge
that no one can have a sense of what is possible . . .
Proximity to a concept, and a gentle shove, are often
all that is needed for a major discovery –
and that is the reason for the drive
towards languages of ever higher levels.
- Douglas R. Hofstadter

Previous chapters covered aspects of the GUI and the Shell. The third built-in Amiga interface is ARexx.

Like the Shell, ARexx is a text-based facility for giving commands to the Amiga, but while the Shell operates externally to programs (copy, delete, etc.), ARexx can operate internally with programs.

ARexx has two main uses: as a scripting language and for Inter-Process Communications (IPC). With the latter, ARexx acts as the standard hub for programs to communicate with each other by sending command and data messages.

SCRIPTING AND IPC

The ability to handle macros, or scripts, is a powerful feature for any application. Whenever a user has a repetitive, well-defined job to do with application software, scripts allow him to automate.

For example, many communications programs allow the user to set up a macro that will dial the phone number of a host system, log into it, check for messages and download those messages for later reading. The macro allows the user to do automatically what is usually done interactively.

ARexx provides Amiga users with a standard macro language.

Benefits of ARexx

Unfortunately, if every application vendor creates his own macro language, the user will have to learn several macro languages instead of just one. Also, each developer would have to spend time creating a new macro language.

With ARexx in place as a system-level macro language, there exists a consistent syntax and user interface for all applications. Part of the reason it can do this is its versatility. Although ARexx is an interpreted language with its own set of keywords, these keywords can be extended to include new words specific to your application.

ARexx is based on REXX, variants of which are used across many platforms.

ARexx has another role, too. It is the system module that allows application software from different vendors to inter-operate, share data and communicate with one another in a multitasking environment. For instance, with ARexx a communications package could be set up to download data from a financial bulletin board and pass the data to a separate spreadsheet application for further analysis.

When told about ARexx, most people imagine its use by power users – but the benefits of ARexx aren't strictly limited to power users. In a corporate environment, an information manager could take off-the-shelf software, throw in a mix of macros, batch commands and IPC, and come up with a system that truly meets the needs of her office. Once set up, the new, customized programs could

be used even by novices.

Value-added resellers can use ARexx to design a custom front end that melds applications from different vendors. The result is a highly-specialized tool and an instant niche market for the application vendor.

ARexx is like an open door on your application. Supporting it leaves the user free to combine your application with others – as a result, buyers of your program could end up using it in ways you never imagined.

STANDARDS FOR AREXX

In order for your program to work with ARexx, it must have an ARexx interface: a software structure that allows your application to send and receive messages from ARexx.

If your application supports scripts, it should do so via ARexx. Even if your program doesn't use macros, you are urged to add an ARexx interface.

Having said that, the purpose of the rest of this chapter is simple: to give a listing of standard ARexx commands so the same commands do the same thing from program to program.

The fast-growing popularity of ARexx combined with its flexibility create, unfortunately, the likelihood of command name and function conflicts. It's frustrating to the user when the OPEN command works differently in his paint program than in his typesetting program.

Your program should support at least a minimal subset of ARexx commands. It increases the power of your program, and as a powerful and seamless standard, it's good for the platform as a whole. Applications that don't support ARexx decrease the overall power and lure of the Amiga. Average users who decide to learn ARexx will almost certainly experience frustration when they run into the brick wall of the program that doesn't support ARexx.

Once you learn how, incorporating ARexx support into your application

requires about as much programming effort as supporting menus – without having to deal with the graphics. This chapter discusses which ARexx commands to support and the style in which they should be supported. For more details on the ARexx language itself refer to the Release 2 Users Manual *Using the System Software.*

General Description and Guidelines

ARexx is an interpreted programming language. As in BASIC, ARexx scripts can be written with any text editor and run without having to use a compiler. If an error occurs, ARexx outputs error messages indicating which line or lines are in error so the user can make changes and try again.

ARexx is unique in that an ARexx script can contain commands that are specific to ARexx as well as commands that are specific to an application. ARexx provides commands typical of most programming languages (variable manipulation, arithmetic, conditional loops, string manipulation, etc.).

Follow these general guidelines when creating your ARexx commands.

Your application should provide, through ARexx, a subset, or even a superset, of the commands available through the GUI. This refers to menu or action gadget commands such as New, Save, Save As... and Quit.

By combining ARexx commands and application-specific commands, the user can create simple or complex scripts to automate common tasks, create new functions, and integrate application software.

For the sake of consistency, your application's ARexx commands should be similar in syntax to the AmigaDOS commands. They take the form:

```
COMMAND [<argument1>, <argument2>...]
```

where COMMAND is a keyword that your application recognizes followed by a list of arguments. In many cases the arguments will be optional and your application should be set to take some default action.

As a general rule, your command set should:

- accept arguments in the standard AmigaDOS style (see Chapter Eight for details);
- honor the AmigaDOS style of pattern matching (e.g., the #? wildcard) when appropriate (see Chapter Eight for details);
- not be case-sensitive, and keywords should not contain spaces;
- recognize and accept quoted arguments;
- be verbose rather than terse (the user can make more sense of a keyword called OPEN versus O);
- accept abbreviations for commonly used commands.

Errors

ARexx scripts can be written by the everyday user or the professional programmer. Scripts must be able to handle error conditions and take some kind of action when an error occurs (i.e., notify the user, prompt the user, terminate, etc.). Therefore, it is important that your application return error codes when a command cannot be performed.

The standard convention in ARexx is for application-specific commands to return error codes of zero (0) for no error, five (5) for warnings such as "Aborted by user," ten (10) for errors such as "<file> wrong type," and twenty (20) for failures such as "Couldn't open the clipboard."

Return codes are placed in ARexx's special variable called "RC" which can be checked for and manipulated in an ARexx script. For example, the user might try to save a file using the "SAVEAS" command. If the file could not be saved for any reason, it would be appropriate to return an error code of ten (10) so that the script could be terminated early, or some other action could be taken.

You may choose to support a more comprehensive set of error codes. This is acceptable but your application should still return zero (0) for no error and use return codes of less than ten (10) for warnings.

Returning Data

VAR = return the value.

STEM = place the value in the specified variable.

Some commands may return information back to ARexx. For example, a text editor or word processor might have a command that returns the string of text under the cursor. The string of text would be placed in a special ARexx variable called RESULT; the user, however, should be allowed to override this with a VAR or STEM switch in the command.

In order for result strings to be returned to ARexx, the script must use ARexx's OPTIONS RESULTS command. This command tells the application that it is OK to return result strings. Data placed in RESULT can then be copied and manipulated within an ARexx script.

Here's an example: a text editor supports a command called GETLINE that returns the line of text under the cursor. The following is a sample ARexx script that uses the GETLINE command to get the line of text under the cursor and place it in a variable called "line." Note that comments are surrounded by pairs of "/*" and "*/".

```
/* Example ARexx script */

OPTIONS RESULTS

'GETLINE'    /* Command telling the application to return the line of
                text under the cursor.
             */
line=RESULT  /* set the variable called "line" equal to ARexx's
                special variable called "RESULT"
             */
```

If the VAR (simple variable) or STEM (complex variable) switches are used, information should be placed into the named variable. For instance:

```
MYCOMMAND STEM comresult
```

This would place the return result in the variable named *comresult*. The VAR and STEM switches make it easier for applications from different vendors to operate on the same data under ARexx.

Text strings that contain records with spaces should be returned with quotes around the record. A space should separate multiple records. For example, if you were trying to return the following list of records:

Letter to John Doe
Notes on Standards
Addresses of the Stars

the information would be returned as:

```
"Letter to John Doe" "Notes on Standards" "Addresses of the Stars"
```

When the command returns multiple records, it should allow the user to specify a stem variable to place the contents in. For example, a command that would return the names of the loaded documents in a text editor would normally return information in the RESULT field, but if the user chooses to place the information in a stem variable, he could specify:

```
GETATTRS DOCUMENTS STEM DocList.
```

which would return:

```
DocList.count = 3
DocList.0 = Letter to John Doe
DocList.1 = Notes on Standards
DocList.2 = Addresses of the Stars
```

In the above example, DocList.count contains the number of records in the array. Also note that the elements do not contain quotes around the information.

ARexx Port Naming

Part of the job of supporting ARexx is adding an ARexx port to your application – this is a software structure through which ARexx communicates with your program. Each ARexx port must have a unique name and must be in upper-case. In ARexx, a port name is derived from:

```
<basename>.<slot #>
```

In the above line, <basename> is the same name your application's executable uses (see Chapter Two), and <slot #> is the next available ARexx port slot for that application.

This ARexx port name should be displayed in the window brought up by the About menu item (along with the version of your application, etc.). The user should also have the ability to rename the port. Being able to specify the ARexx port name in the project icon's Tool Types field, or from the command line by using the PORTNAME keyword, is a good thing.

Unique port names should be given to each project started within your application and for each instance of your application. For example, a fictional word processing package named GonzoWord with a basename of GWord should have an ARexx port of GWORD.1 when the first document is opened. A second document running concurrently should be given the port name of GWORD.2. If the user opened the GonzoWord program again without choosing QUIT in the first instance, that third document should have a port name of GWORD.3.

Command Shell

Your application should allow the user to open a console window or command shell within your application's environment. There are two possible ways of handling this: you can provide a window that looks like a basic Shell window or one that looks like Workbench's Execute Command window.

Commands entered into this window would allow direct control of your application.

As with any other type of window, your application should allow the user to snapshot the placement and size of the command shell.

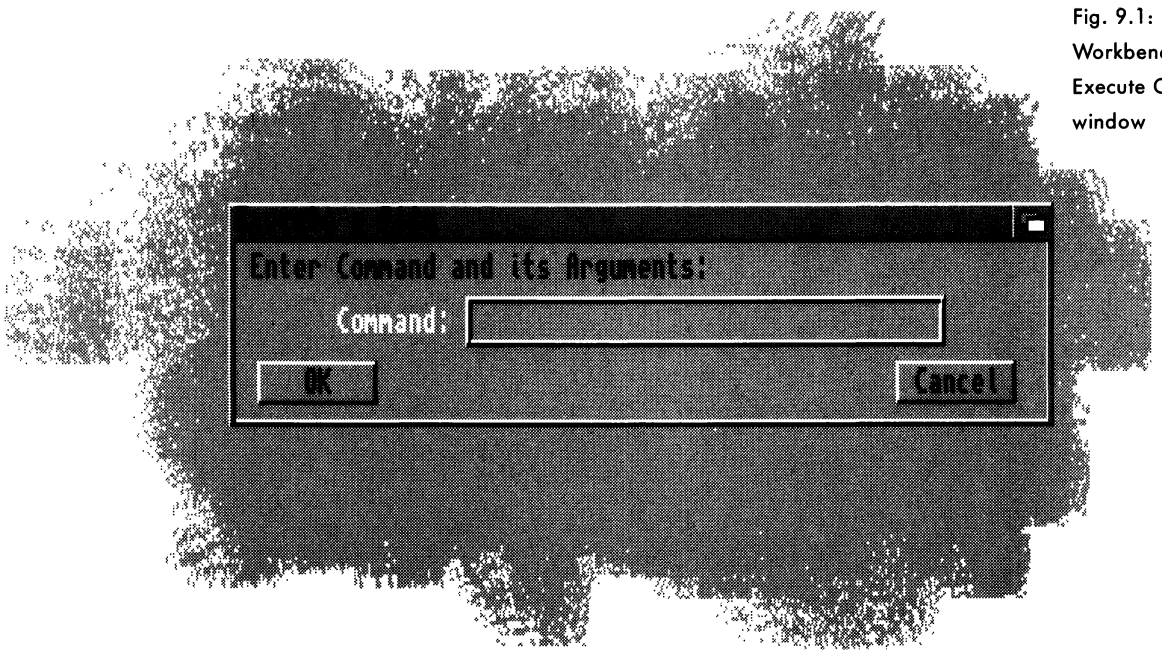
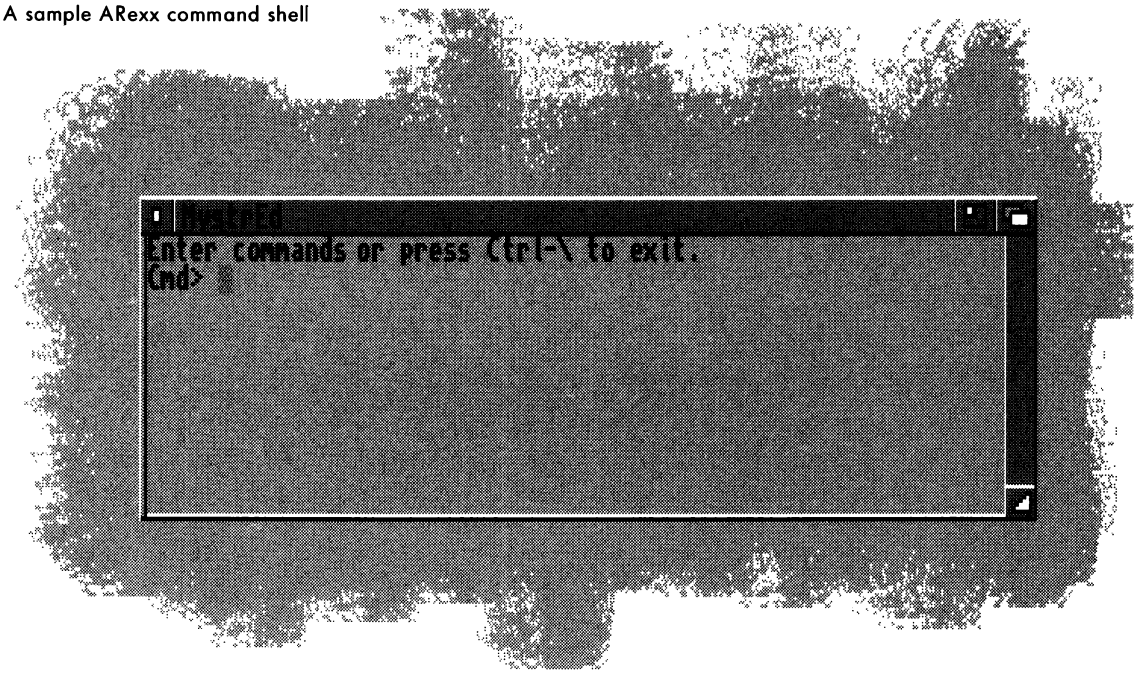


Fig. 9.1: Workbench's Execute Command window

Fig. 9.2: A sample ARexx command shell



STANDARD AREXX COMMANDS

The following commands are the minimum commands that your application should support. These commands are reserved across all applications – don't use these commands for any other functions.

As listed here, the command is named in bold on the first line. Any options for that command are indented beginning on the second line. The definition for the command is on the following line, and that is followed by indented and bulleted definitions of the options. If the options are self-explanatory, no definition is given.

The commands are presented here in the same style as Shell commands discussed in Chapter Eight. You should also use this style when you implement ARexx.

Project-Related Commands

NEW

PORTNAME/K

NEW creates a new project and work area. This command should return the ARexx port name assigned to the project.

- PORTNAME is used to assign a specific port name to the project.

Your application should support at least these 15 ARexx commands.

CLEAR

FORCE/S

This command clears the current project and its work area.

- FORCE suppresses the Modified Project requester.

OPEN

FILENAME/K,FORCE/S

This command opens the specified project into the current work area. If no FILENAME is given, prompt the user for a filename via a file requester.

- FORCE suppresses the Modified Project requester.

comma = no arguments
/A = always required
/F = final argument
/K = keyword required
/M = multiple arguments
/N = number
/S = switch keyword
/T = toggle keyword

SAVE

,

This command saves the current project to the current filename. If the project is unnamed, bring up the file requester so that the user can specify a filename.

SAVEAS

NAME/K

This command saves the current project to the specified filename.

- NAME specifies what the new project will be called. If no name is provided, it should bring up the file requester so that the user can specify the filename.

CLOSE

FORCE/S

This command closes the current project and window.

- FORCE suppresses the Modified Project requester.

PRINT

PROMPT/S

PRINT will print the specified object using the current settings.

- PROMPT provides a requester for use in setting print parameters.

QUIT

FORCE/S

QUIT stops the program. If the project was modified, the user should be prompted to save the work.

- FORCE suppresses the Modified Project requester.

Block-Related Commands

CUT

,

CUT removes the currently selected block of information from the project and places it in the clipboard.

COPY

,

COPY places a duplicate of the currently selected block of information into the clipboard.

PASTE

,

PASTE puts the contents of the clipboard into the project – at the currently active point.

ERASE

FORCE/S

ERASE removes the currently selected block of information from the project.

- FORCE suppresses the "Are you sure?" requester.

Other Standard Commands

HELP

COMMAND,PROMPT/S

HELP provides access to information about your application. Information about things such as the supported functions and parameters required for functions should be readily available. When triggered from a non-graphical user interface, HELP should present the user with a text list of all the commands that the application supports.

- COMMAND presents the user with the list of options available for that command.
- PROMPT activates a graphical help system.

FAULT

/N

FAULT gives the user the text message assigned to the given error number. The text should be sent to the RESULT field. (See the "Returning Data" section on page 118.)

RX

CONSOLE/S,ASYNC/S,COMMAND/F

RX allows the user to start an ARexx macro.

- CONSOLE indicates that a console (for default I/O) is needed.
- ASYNC indicates that the command should be run asynchronously.
- COMMAND sends whatever is typed on the rest of the line (usually a command) to ARexx for execution.

OTHER COMMON AREXX COMMANDS

These commands aren't applicable for every program, but please use them if your program provides this sort of functionality.

These other ARexx commands are presented to avoid conflicts among those applications that do support ARexx commands with similar functions or names.

Cursor Positioning Commands

GOTOLINE

/N/A

This command moves the cursor to a specified line.

GOTOCOLUMN

/N/A

This command moves the cursor to a specified column.

CURSOR

UP/S,DOWN/S,LEFT/S,RIGHT/S

CURSOR moves the cursor up, down, left or right a single line or column position.

LINE

/N/A

LINE accepts positive or negative arguments to move the cursor up or down relative to its current position.

COLUMN

/N/A

COLUMN accepts positive or negative arguments to move the cursor left or right relative to its current position.

NEXT

WORD/S,SENTENCE/S,PARAGRAPH/S,PAGE/S

NEXT moves the cursor to the next word, sentence, paragraph or page.

PREVIOUS

WORD/S,SENTENCE/S,PARAGRAPH/S,PAGE/S

PREVIOUS moves the cursor to the previous word, sentence, paragraph or page.

SETBOOKMARK

/N

This command is used to remember a place in text. If the program supports multiple bookmarks, a number can be used to differentiate them.

GOTOBOOKMARK

/N

Move cursor to a bookmark. If the program supports multiple bookmarks, a number can be used to differentiate them.

POSITION

SOF/S,EOF/S,SOL/S,EOL/S,SOW/S,EOW/S,SOV/S,EOV/S

POSITION moves the cursor to the position specified by the argument.

- SOF moves it to the beginning of the file.
- EOF moves it to the end of the file.
- SOL moves it to the beginning of the current line.
- EOL moves it to the end of the current line.
- SOW moves it to the start of the current word.
- EOW moves it to the end of the current word.
- SOV moves it to the top of the current view.
- EOV moves it to the bottom of the current view.

Find and Replace Commands

FIND

TEXT/F

FIND searches for text that matches the specified string. If no string is specified on the command line, a requester should be presented allowing the user to enter a search string.

FINDCHANGE

ALL/S,PROMPT/S,FIND/K,CHANGE/K

- PROMPT brings up a requester.
- FIND searches for the string specified after the keyword.
- CHANGE replaces the specified FIND string with the string specified after the CHANGE keyword.

FINDNEXT

,

This moves the cursor to the next occurance of the current search string without displaying a requester.

Other Text-Related Commands

TEXT

TEXT/F

This command can be used in ARexx macros, allowing text to be entered via a command and honoring word wrap, insertion or any other current modes.

UPPERCASE

LOWERCASE

SWAPCASE

CHAR/S,WORD/S,LINE/S,SENTENCE/S,PARAGRAPH/S

These three commands can be used if your program has the ability to swap the case of a letter or let-

ters. The default switch should be WORD.

FONT

NAME/S,SIZE/N,ITALIC/S,BOLD/S,PLAIN/S,UNDERLINED/S

Use this command if your application supports font selection. A combination of the text style arguments should be allowed following the SIZE argument.

Since ARexx is a standard language on the Amiga, and since it is used across applications, the command names given here are considered to be reserved for the function listed with the name.

UNDO

,

This command reverts back to the last state of the project. If your program supports multiple levels of undo, a number should be recognized as an argument. The default should be one level of undo.

REDO

,

For applications with multiple levels of undo, redo will allow the user to undo the UNDO command. See Chapter Six for more information.

Window-Related Commands

MOVEWINDOW

LEFTEDGE/N,TOPEDGE/N

This command should be used to change the position of a window. An argument of -1 means no change or don't care. Your application should either do the best job it can to move the window to the specified position, or return an error code.

SIZEWINDOW

WIDTH/N,HEIGHT/N

This command should be used to change the size of a window. An argument of -1 means no change or don't care. Your application should either do the best job it can to size the window or return an error code.

CHANGEWINDOW

LEFTEDGE/N,TOPEDGE/N,WIDTH/N,HEIGHT/N

This command is similar to MOVEWINDOW and SIZEWINDOW but the application should move and size the window as one operation. This is important to the user because moving a window to a new size and position can require as many as three commands, depending on the original position and size as well as the target position and size.

WINDOWTOFRONT

,

This command moves a window to the front.

WINDOWTOBACK

,

This command moves a window behind all others.

ACTIVATEWINDOW

,

This command activates a window.

ZOOMWINDOW

,

If your program supports zoom gadgets on your windows, use this command to make a window small.

UNZOOMWINDOW

,

If your program supports zoom gadgets on your windows, use this command to unshrink it.

Telecommunications Commands

BAUD

/N

This command sets the baud rate.

PARITY

EVEN/S,ODD/S,NONE/S

Use this command to set the parity.

STOPBITS

1/S,0/S

Use this to set the stopbits.

DUPLEX

FULL/S,HALF/S

Use this to set the duplex mode.

PROTOCOL

/K

Use this command to set the protocol.

SENDFILE

NAME/K

Use this to start a file send. If the filename is omitted, the user should be presented with a file requester.

CAPTURE

NAME/K

Use this to open a capture buffer. If the filename is omitted, the user should be presented with a file requester.

DIAL

NUM/F

Use this command to dial a phone number.

REDIAL

,

Use this to redial the last phone number.

SEND

,

Use this to send a text string.

WAIT

,

Use this to wait for a text string.

TIMEOUT

/N

Use this to set the timeout value (in seconds) for waits.

Miscellaneous Commands

TEXTPEN

BACKGROUNDPEN

/N

Text-oriented applications that support the ability to set pen and paper colors can use these two commands. Valid values should be bounded by the screen's maximum number of colors rather than by existing Amiga hardware.

LOCKGUI

,

This inhibits the graphical user interface of the application.

UNLOCKGUI

,

This undoes LOCKGUI, allowing GUI events to resume.

GETATTR

OBJECT/A,NAME,FIELD,STEM/K,VAR/K

GETATTR obtains information about the attributes of an object.

- OBJECT specifies the object type to obtain information on.
- NAME specifies the name of the object.
- FIELD specifies which field should be checked for information.
- STEM (keyword) specifies that the information is to be placed in the following stem variable rather than the RESULT field.
- VAR (keyword) specifies that the information is to be returned in a simple variable.

In the above OBJECT argument, the user could be seeking information on the attributes of any of the following items:

- APPLICATION
- PROJECT <name>
- WINDOW <name>
- PROJECTS
- WINDOWS

If the destination variable is a stem variable, the following fields are recommended:

APPLICATION would consist of the following nodes (an aspect of APPLICATION that the user can seek sub-information about):

- .VERSION — Contains the current version information for the application.
- .SCREEN — Contains the name of the screen that the application resides in (if the screen is public).

PROJECTS.COUNT would contain the number of projects; PROJECTS.0 through PROJECTS.n would contain the handle for the project.

PROJECT would consist of the following nodes. Other variables could be added based on the type of application. For example, a text editor could add a variable called .LINES that would contain the number of lines in the project's file.

- .AREXX — Contains the ARexx port assigned to the project.
- .FILENAME — Contains the complete filename of the project.
- .PATH — Contains the path portion of the filename.
- .FILE — Contains the file portion of the filename.
- .CHANGES — Contains the number of changes made to the project (since last save).
- .PRIORITY — Priority at which the project's process is running. Allows the user to set the priority. For example, a 3-D modeling application could be working on several different projects at one time. Using this field, the user would be able to give more time to the most important project.

WINDOWS.COUNT would contain the number of windows; WINDOWS.0 through WINDOWS.n would contain the handle for the window.

WINDOW would consist of the following nodes:

- .LEFT — Left edge of the window.
- .TOP — Top edge of the window.
- .WIDTH — Width of the window.
- .HEIGHT — Height of the window.
- .TITLE — Window title.
- .MIN.WIDTH — Minimum width of the window.
- .MIN.HEIGHT — Minimum height of the window.
- .MAX.WIDTH — Maximum width of the window.
- .MAX.HEIGHT — Maximum height of the window.
- .SCREEN — Name of the screen in which the window resides.

SETATTR

OBJECT/A,NAME,FIELD,STEM/K,VAR/K

SETATTR manipulates the aspects of an object.

- OBJECT specifies the object type to manipulate.
- NAME specifies the name of the object.
- FIELD specifies which field should be modified.
- STEM specifies that the information is to be extracted from the following stem variable.
- VAR specifies that the information is to be extracted from the following simple variable.

WINDOW

NAMES/M,OPEN/S,CLOSE/S,SNAPSHOT/S,ACTIVATE/S,MIN/S,
MAX/S,FRONT/S,BACK/S

WINDOW allows the user to manipulate several aspects of the window.

- NAMES specifies a list of windows to manipulate. This command should support wildcard expansion.
- OPEN is used to indicate that the window(s) should be opened.
- CLOSE is used to indicate that the window(s) should be closed.
- SNAPSHOT records the current position and size of the named window(s).
- ACTIVATE will make the window the input focal point.
- MIN sets the named window(s) to the minimum size.
- MAX sets the named window(s) to the maximum size.
- FRONT places the named window(s) in front of other windows.
- BACK places the named window(s) behind other windows.

CMDSHELL

OPEN/S,CLOSE/S

CMDSHELL opens a command shell so the user can interact directly with the application at a command level. CMDSHELL allows the user quick access to infrequently used functions or macros that are not bound to a key, menu or button.

ACTIVATE

,

ACTIVATE starts the GUI of an application. This involves "de-iconifying" it, bringing its screen to the front, unzooming the main window (i.e., expanding it to its previously set size) and making that window active.

DEACTIVATE

,

This command shuts down the GUI of an application. DEACTIVATE restores the GUI to the state it was in before receiving the ACTIVATE command.

DISABLE

NAMES/M

This command disables a command or list of commands. DISABLE should also disable any user interface items to which the command is bound. For example, if a command is bound to a gadget, then that gadget should be disabled as well. This command should support pattern expansion.

ENABLE

NAMES/M

ENABLE makes a command or a list of commands available to the user. ENABLE should also enable the GUI items to which they are bound (see DISABLE). This command should support pattern expansion.

LEARN

FILE/K,STOP/S

LEARN allows the application to build an ARexx macro consisting of the actions the user performs.

- FILE specifies the name of the file in which the user will save the commands.
- STOP tells the application to stop learning.

ALIAS

NAME/A,COMMAND/F

ALIAS assigns a user-definable name to a function and its options.

- NAME specifies what the new command will be called.
- COMMAND represents the rest of the line where the command and whatever options are to be bound to that command are entered.

SELECT

NAME/A,FROM/K,NEXT/S,PREVIOUS/S,TOP/S,BOTTOM/S

SELECT is used to select an object from a list of similar objects. The main use for this command is to allow the user to select the current project from a list of projects. Returns the name of the current project.

- FROM specifies the list from which the user can select. The default should be the main project list.
- NEXT specifies that the next object on the list should be selected.
- PREVIOUS specifies that the previous object on the list should be selected.
- TOP specifies that the first object on the list should be selected.
- BOTTOM specifies that the last object on the list should be selected.

ADVANCED AREXX COMMANDS

Increasingly, programmers are supporting ARexx commands that allow users to add their own ARexx scripts or even to modify the program's interface. If you would like your program to support any of the following functions, please use these standard commands:

BUTTON

NAME/A,LABEL/K,WINDOW/K,PROMPT/S,CMD/F

Some applications could set aside a number of action gadgets (buttons) to which the user can assign

ARexx scripts. The BUTTON command would allow the user to modify these special gadgets.

- NAME specifies the name of the button to be edited.
- CMD specifies the command name assigned to the button.
- LABEL specifies the text label assigned to the button.
- WINDOW specifies the window in which the button belongs. This should default to the active or last active Intuition window for that application.
- PROMPT activates a window in which the user can graphically edit the button.

Some of these advanced commands allow the user to modify your program's interface.

KEYBOARD

KEY/A,WINDOW/K,GLOBAL/S,HOTKEY/S,PROMPT/S,CMD/F

KEYBOARD allows commands to be tied to keystrokes.

- KEY specifies which keystroke to edit.
- CMD specifies what command will be assigned to that keystroke.
- WINDOW specifies which window the hotkey will perform in. It should default to the active or last active Intuition window for that application.
- GLOBAL indicates that the keystroke will operate when any of the application's windows are active.
- HOTKEY indicates that the keystroke will operate regardless of what window or application is active.
- PROMPT gives the user a window in which to edit the keystroke.

MENU

NAME/A,LABEL/K,MENU/K,WINDOW/K,PROMPT/K,CMD/F

MENU provides a means to add, edit or delete a menu item. MENU also allows access to the function that the menu item triggers.

- NAME specifies which menu item the user will edit.
- CMD specifies what command will be assigned to the menu item.
- LABEL assigns a text label to the menu item.
- MENU specifies which menu strip the new or edited item will be added to.
- WINDOW specifies which window the new or edited menu belongs in. This should default to the active or last active Intuition window for that application.
- PROMPT (switch) activates a window in which the user can graphically edit the menu item.

NOP

,

The NOP command is a special do nothing command (short for No OPeration) that is often extremely useful. For example, if your program supports custom keyboard definitions, it is often necessary to disable some keys when writing custom macros.

Programmable Requesters

When a user writes an ARexx macro, it is often very useful to be able to bring up a file requester, or other type of requester as part of the macro. If your program supports programmatic access to its requesters, the following commands should be used:

REQUESTFILE

TITLE/K,PATH/K,FILE/K,PATTERN/K

This command brings up a file requester. The user can specify the title, path and filename. If any of these arguments are omitted, they should be filled in with some defaults (e.g., a default title and empty strings for path and/or filename).

The path and filename should be returned in RESULT. A warning should be returned if the user cancels the requester.

REQUESTSTRING

PROMPT/K,DEFAULT/K

This command brings up a string entry requester. The user can specify a prompt string and a default string to be placed in the string editing gadget. Like the REQUESTFILE command, defaults should be provided if arguments are omitted.

The string entered should be returned in RESULT. A warning should be returned if the user cancels the requester.

REQUESTNUMBER

PROMPT/K,DEFAULT/K

This command brings up a number entry requester. The user can specify a prompt string and a default number to be placed in the string editing gadget. Like the REQUESTFILE command, defaults should be provided if arguments are omitted.

The number entered should be returned in RESULT. A warning should be returned if the user cancels the requester.

REQUESTRESPONSE

TITLE/K,PROMPT/K

This command brings up a query requester. The user can specify a prompt string and possibly text for the OK and CANCEL gadgets if your program wants to support these additional options.

A warning should be returned if the user selects cancel.

REQUESTNOTIFY

PROMPT/S

This command brings up a notification requester that can only be satisfied with an OK type of response. ▲

chapter ten
THE KEYBOARD

In the fight between you and the world,
back the world.
- Frank Zappa

Your application should make gadget and menu items selectable with the keyboard as well as with the mouse. Often these keyboard equivalents will provide shortcuts and a smoother workflow that is appreciated by the user. Also, allowing the choice conforms to the Amiga credo: "Power to the User."

The set of conventions you should follow to implement this style rule are presented in this section.

KEYBOARD CONVENTIONS

The arrangement of the Amiga's keyboard varies from model to model, but in general there are always three sets of keys:

- the standard keyboard
- special keys
- modifier keys

The Standard Keyboard

The standard keys consist of the familiar alphanumeric keys found on any standard typewriter. (In English, this is referred to as the QWERTY keyboard, but in German, since the standard keys are arranged differently, it's known as the QWERTZ keyboard. It's called other names in other languages.)

Since the Amiga computer is sold in many different countries with different national keyboards, the arrangement of the standard keys can vary. To handle this, the Amiga uses a special facility known as a keymap that allows the user to change the way the keyboard input is mapped so it will correspond to his country's keyboard layout and characters. For example, to access all the standard German ASCII characters the user can type "Setmap d" in the Shell. Then German characters such as ü and ß will have the same key designations as they do on a standard German keyboard.

Use the keymap facility to handle key assignments on the standard keyboard.

Special Keys

The special keys include the ten function keys on the top row of the Amiga's keyboard, the Help key, the cursor (arrow) keys, the Del key, the backspace key and the Esc key.

Modifier Keys

The modifier keys are the Ctrl, Shift, Alt and Amiga keys found on either side of the space bar. Modifier keys don't do anything by themselves; they alter the meaning of a key that is pressed at the same time. They are also used to modify the meaning of a mouse selection. For example, holding down the Shift key while clicking with the mouse is used for multiple object selection.

Dead keys

A "dead" key is a key, or combination, that does nothing immediately but modifies the output of the next key. For example, on an American keyboard, Alt-H will superimpose a caret (^) symbol over the next appropriate character.

Although relatively unimportant on the American keyboard, dead keys are important in many languages. Keep in mind that you need to work these in manually if are doing your own raw key mapping.

SYSTEM KEYBOARD SHORTCUTS

To provide mouse functions from the keyboard, only three features are needed: a way to press the left mouse button, a way to press the right mouse button, and a way to move the mouse pointer. These are provided by the system. They are listed here so you are aware of them and can avoid using these key combinations for any other purpose.

keyboard shortcut	equivalent mouse activity
Either Amiga + Left-Alt	Same as clicking left mouse button
Either Amiga + Right-Alt	Same as clicking right mouse button
Either Amiga + Cursor Key	Moves mouse pointer
Either Amiga + Shift + Cursor Key	Moves mouse pointer in larger steps

Five additional Amiga system functions have keyboard shortcuts:

default keyboard shortcut	equivalent system function
Left-Amiga N	Workbench screen to front
Left-Amiga M	Front screen to back
Left-Amiga B	Requester OK (leftmost gadget)
Left-Amiga V	Requester cancel (rightmost gadget)
Left-Amiga + selection button	Drags screen whether the pointer is on the title bar or not

The first four functions listed above always use Left-Amiga in combination with another key. The second key in the combination can be changed by the user with the IControl Preferences Editor in the Prefs drawer of Workbench. For the screen dragging shortcut, the user can change the modifier or combination of modifiers (Ctrl, Amiga, Shift, Alt) that need to be combined with the selection button.

The Left-Amiga key

Keep in mind that the Left-Amiga key is reserved at all times for system operations and should never be used as a qualifier for an application keyboard shortcut.

APPLICATION KEYBOARD SHORTCUTS

Your application should provide a way for the user to bind any application function or ARexx macro to a key or combination of keys.

Gadgets

Use a logical letter from the gadget label as the keyboard shortcut. For instance a gadget labeled "Smoothing" could use the S key as its keyboard control, while a "Left Offset" gadget may use the O key.

Place an underscore under the letter in the gadget label that activates the gadget. (Note: although the letter is capitalized on the label, the default action should react to the lower-case letter. Some gadgets have a different action for the shifted version of the letter. See the list on the next page.)

Three Rules

- All action should occur on the downpress of the key.
- The same visual feedback should be given for keyboard activation as is given for mouse activation.
- Avoid assigning the Enter key to a gadget.

feedback from keystroke-activated gadgets

Action button	On the downpress of the key, the gadget should appear to be pressed in. On release of the key, the gadget should come back out. (Note: at the time this manual was published, GadTools did not support this function.)
Check box	Toggle the state of the check mark.
Scrolling list	Unshifted would cycle forward through the choices. Shifted would cycle backwards through the choices.
Radio button	Unshifted would cycle forward through the choices. Shifted would cycle backwards through the choices.
Cycle gadget	Unshifted would cycle forward through the choices. Shifted would cycle backwards through the choices.
Selection gadget	Unshifted would cycle forward through the choices. Shifted would cycle backwards through the choices.
Scroll gadget	Unshifted would cycle forward through the choices. Shifted would cycle backwards through the choices.
Slider	Unshifted would increase the level by one unit. Shifted would decrease the level by one unit.
Text box	Activate the gadget for entry.
Numeric entry	Activate the gadget for entry.

Menus

Use a Right-Amiga combination as the default keyboard shortcut for a menu item. Here is a list of the defaults in the English language for an application that has standard menus. This should be localized to the language at hand for non-English applications.

Project Menu

Right-Amiga N	New
Right-Amiga O	Open...
Right-Amiga S	Save
Right-Amiga A	Save As...
Right-Amiga P	Print
Right-Amiga Q	Quit Program

Edit Menu

Right-Amiga X	Cut
Right-Amiga C	Copy
Right-Amiga V	Paste
Right-Amiga Z	Undo

USE OF THE SPECIAL KEYS

As stated previously, the special keys are the ten function keys, the Del key, the Help key, the cursor (arrow) keys and the Esc key.

Cursor Keys

Cursor keys are a convenient way to control the movement of the cursor inside an application. Here are some standard ways to use them:

Unmodified Cursor

Move a small amount in the specified direction. Often this is one unit such as a character in a word processor or a pixel in a paint package, but it could be several pixels in an application where navigation is more important than fine control.

Shift + Cursor

Move to the appropriate extreme of the window, or shift the view by one windowful if you're already at that extreme.

In applications such as a word processor where the two directions are not symmetrical, shifted cursor keys could take on different meanings. Shifted up and down cursors would page through windowfuls while the shifted left and right cursors would show more on the left and right if there is more to show. Of course it is often the case that the width of the document fits in the window, so the shifted left and right cursors could act as beginning- and end-of-line commands (or edge of window if the cursor isn't constrained to the form of the text).

Alt + Cursor

This is used for application-specific functions. It's usually semantic units such as words in a text processor or fields in a spreadsheet.

Ctrl + Cursor

Move to the appropriate extreme of the project (beginning, end, extreme left, extreme right).

In the word processor example, the up and down cursors keys would take the cursor to the beginning and end of the file, respectively. But again, if the file fits within the width of the window, the left and right cursor keys combined with Alt would act like their shifted cousins.

Function Keys

Function keys should generally be reserved for the user to define. If your application does make use of them, then it should at least allow the user to redefine or modify them.

Help Key

If your application has built-in help, use the Help key to trigger it from the keyboard. Avoid giving the user that helpless feeling he gets when he presses the Help key and nothing happens. ▲

chapter eleven
DATA SHARING

We say that there is something that is equal.
I do not mean a stick equal to a stick or a stone to a stone,
or anything of that kind,
but something else beyond all these, the Equal itself.
- Plato

This chapter is a little different from the other chapters in this book because it does not directly consider the question of how to make the best human–machine interface for Amiga applications. Instead, it covers the best ways for programs to share data with each other.

User interactions are not directly involved in data sharing, but the issue is crucial to the user. Without a data sharing standard, the user has to convert data to a different format every time it is used across applications. Worse still, a conversion tool may not be available. With a data sharing standard this problem is eliminated.

A data sharing standard is also important to get the best advantage from multitasking. When applications on the Amiga follow a data standard, a unique synergy is created – specialized applications from different vendors can be combined seamlessly for a custom environment.

For instance, you can create a background picture in a paint package, add three-dimensional text in a ray-tracing application, and add animation in an ani-

mation package, all without once running a conversion program. The applications can run together simultaneously as an integrated graphics package.

The Amiga's standard for data sharing, Interchange File Format (IFF), is a widely accepted and simple set of rules that was well-defined and adopted very early in the Amiga's history.

A PRACTICAL INTRODUCTION TO THE IFF STANDARD

This section provides a brief overview of the IFF standard. For more detail refer to the "Interchange File Format Specification" published by the Commodore Applications and Technical Support group.

Perhaps the greatest strength of IFF is that it was well-defined and adopted very early in the Amiga's history.

An IFF data structure has two levels. The first level is a wrapper or envelope structure which is, technically, the syntax. It's easy to describe: the IFF wrapper is just a header containing an identifier and the file size. The identifier is four letters and the file size is a number taking up 32 bits of space. The whole IFF wrapper weighs in at eight bytes of disk space. That's it.

There are only three possible identifiers for the wrapper part of IFF files: FORM, CAT (a space after the T) and LIST.

- FORM This is by far the most common type, equivalent to a simple file.
- CAT A concatenation, equivalent to multiple files put together.
- LIST Also a concatenation but with some properties shared between the separate files to avoid redundancy.

It is possible to build some fairly complex files using just these three types since a FORM may contain other FORMs within it; a LIST can contain other FORMs, CATs or LISTs and so forth. (To help with the possible complexity of IFF syntax, Release 2 of the operating system now contains a function library named iffparse.library.) In practice, however, the most common IFF file is the simple

FORM containing a single set of data. There may be any type of data within the FORM wrapper.

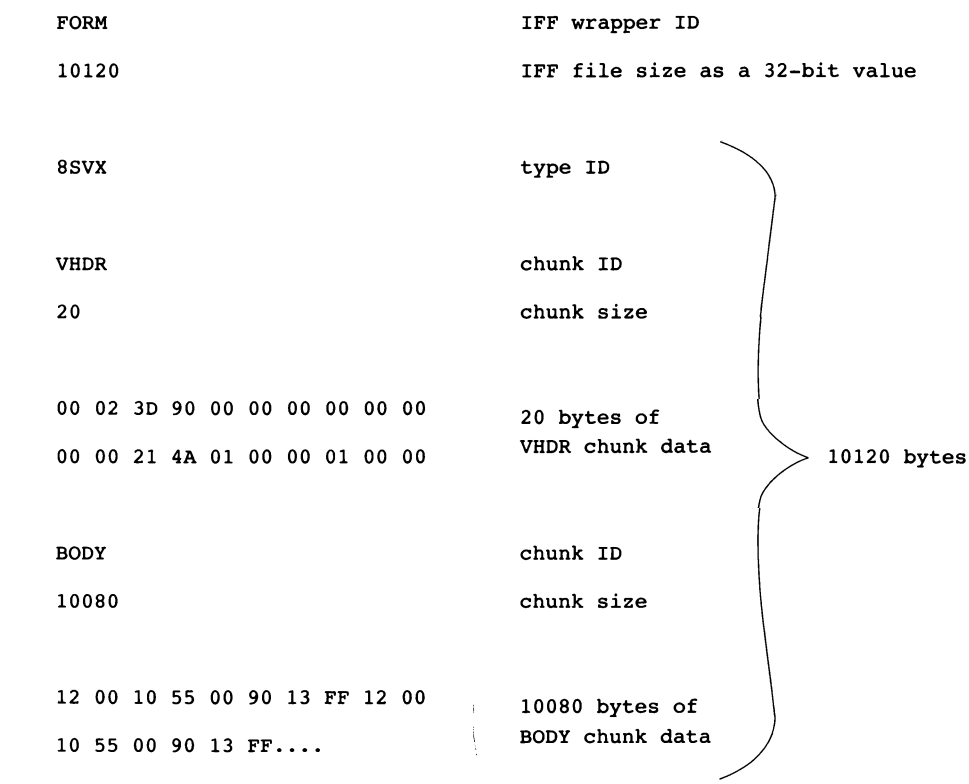
The second layer of an IFF file, underneath the wrapper, defines the particular file types. The file types consist of a four-letter identifier followed by a series of data chunks which contain the data natural to the type. Some examples are:

type ID	data stored in this type
ILBM	Graphics data in interleaved bitmap format. This is very widely used on the Amiga.
8SVX	Audio data in 8-bit sample bytes. The most widely used audio file format on the Amiga.
FTXT	Formatted text. Recommended for sharing text data.
ANIM	A common way to store interchangeable animation files.

Each IFF file type contains a number of data chunks specific to the type. These data chunks are the basic building blocks of IFF files and they come in a wide variety. They can be structured in any number of ways, but a chunk always starts with a four-letter ID followed by the chunk size which, as in the wrapper, is given as a 32-bit number.

For example the 8SVX file type consists of two chunks: a VHDR chunk which contains information like the sampling speed, and the BODY chunk which contains the sample bytes.

Here is a listing of a typical 8SVX file to give you a better idea of how IFF files are structured:



The IFF Philosophy

The goals of IFF are lofty but obviously within reach. IFF is intended to make data abstract enough to be stored independent of a particular program, compiler, device or machine architecture. With that achieved, IFF files could be shared not only between applications but also on and between platforms other than the Amiga.

The key to this is making the data abstract. The IFF standard accomplishes this by building the files through chunks each with an ID and size. The ID is a reference by name to a given implementation of an access procedure. The access procedures can evolve and change independently of the software which uses

them. And the size count enables applications to support object operations like "copy" and "skip to next" regardless of the object type or contents.

IFF is meant to be easily extensible. The CAT and LIST wrappers make it easy to combine existing types to form new composite types. And if all else fails the IFF specification provides rules for creating new file types for special storage needs that are not handled by the existing IFF types.

IFF and Your Application

If an IFF format exists for your program's genre, you only need to provide two features:

- a way for users to save files in that IFF format
- a way for your program to import files that have been saved in that IFF format.

You should support IFF two ways: provide a way for users to save files in IFF format; and provide a way for users to import IFF files.

In some cases the IFF format is also the best format for everyday file storage. Many applications have opted to use ILBM for those purposes. In other cases, the IFF format is not great as a normal storage format. In these cases, you can provide your own internal storage format, but you should also provide the means to save and import IFF.

Note, however, that revisions to your application could mean changes in your internal data format. If you have been using your internal data format as your storage format as well, any files saved in the old format may not be readable by the new format. If you don't use IFF as the default storage format, at least try to provide for backward compatibility in your own program.

If No IFF Format Exists

If no IFF format exists for your program's genre, it would be a good idea to develop an IFF format and use that.

If you need to create a new IFF format or expand upon an existing one, then you must register the change with Commodore. Commodore's Applications and Technical Support group serves as a central clearinghouse to keep track of new

developments in the IFF standard. A registry of IFF form and chunk types is available from Commodore on the IFF disk. This specification is required reading for all developers who want to conform to the Amiga's user interface standards.

THE CLIPBOARD

IFF provides a data sharing standard. In conjunction with that, the Amiga's clipboard device provides a place to store and retrieve that data. It does this by caching data to RAM and automatically spooling the data to disk if necessary.

The clipboard is the recommended storage device for cut-and-paste operations and it's the best place to store data which is meant to be quickly moved between two applications that are running simultaneously. It also provides a solid metaphor in keeping with the Workbench.

The clipboard allows the exchange of data dynamically between your application and another.

Use IFF

All data written to the clipboard must be written in the IFF format. For most applications the data will be one of two types: graphics or text. For graphic clips use the ILBM form of IFF. For text clips use the FTXT form of IFF.

When reading from the clipboard, never blindly read the data assuming, for instance, that the clip contains FTXT with a CHRS as the first and only chunk. Remember, the user may have been switching between many applications and may even have made a mistake in the cut-and-paste operation.

Storage Concerns

When storing data in the clipboard, it is acceptable to write different representations of the same data. For instance, in a music application when the user cuts a bar of music, the application can write the information in three ways. The

music itself could be saved in a SMUS form; the lyrics could be saved separately in an FTXT form; and a picture of the notation could be saved in an ILBM form. This type of multiple clip should be contained within a CAT "wrapper" form. Use CAT CLIP as the global form name.

Unit Selection

The clipboard device allows for the selection of units ranging from zero to 255. By default, your application should always use clipboard unit zero for interactive cut, copy and paste operations. Provide a means, however, for the user to specify a different unit number.

NOTIFICATION

Notification is a means by which an application can receive a message whenever a certain file of interest is modified. By establishing "hot links" between applications, notification allows a change made to a file with one program to be automatically reflected in another application that is using the same file.

Hot links allow a change made to a file with one program to be automatically reflected in another application which is using the same file.

An example would be a paint package in which the user could save a brush and, using notification, automatically have an image processor application modify the brush and return it in a separate file. The scenario would work like this:

1. PAINT PACKAGE
    - Writes brush to file named T:brush.

2. IMAGE PROCESSOR
    - Gets notification that T:brush has been updated.
    - Reads T:brush into work area.
    - Alters brush using an image processing feature.

- Writes brush out to T:brush.mod.

3. PAINT PACKAGE
   - Gets notification that T:brush.mod has been updated.
   - Reads T:brush.mod into brush area.

There is only one style standard to worry about with notification: keep RAM usage reasonable. Because RAM is a limited resource, applications that have very large amounts of data should not store directly in RAM:.

Notification may not work in a network situation because it is not implemented with some file systems. It does work on RAM:, FFS and OFS.

ENVIRONMENT VARIABLES

With general availability of networking cards for the Amiga, the use of environment variables is sure to increase. Environment variables are basically places where information can be stored; they are similar to logical assignments in that a variable text string is given a generic label. For example, in the Shell, the user can type:

```
echo $workbench
```

and be presented with the current version in use. Environment variables are set by the user with the SetEnv command of AmigaDOS.

Store your variables in a subdirectory such as ENV:<basename>/variables. Only user-defined and standard system variables should go in ENV:. ENV:Sys/ is reserved for system functions.

Here are some recommended environment variable names currently in use in the system. Note: they are not case-sensitive.

workbench	*The version of Workbench*
kickstart	*The version of Kickstart*
hostname	*Contains the machine name*
username	*Contains the login name of the current user of the machine*
editor	*Sets the preferred text editor*

The following are local variables provided by the Shell. These are most relevant to applications that use DOS scripts. Of course you should avoid using these names for your variables since they are set automatically by the Shell.

process	*The process number*
RC	*The primary return code of the last command run*
Result2	*The secondary return code of the last command run* ▲

chapter twelve
PREFERENCES

All men attribute to themselves
freedom of will.
- Immanuel Kant

On the Amiga, the term "Preferences" refers to the set of programs and data used to configure the machine's working environment.. They come in two basic varieties: system preferences (the basic ones in the operating system) and application preferences (any you choose to add).

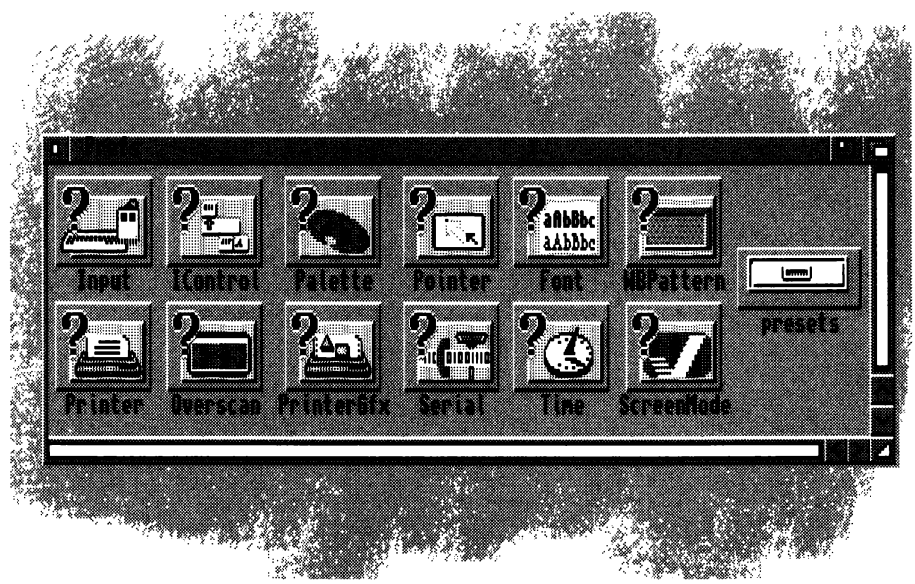


Fig. 12.1: Workbench's Prefs directory

A variety of editors in the Prefs drawer of Workbench allows the user to set various system defaults. These represent the system preferences. Some of these control settings for peripherals such as printers, while others control the look and feel of the system. The Palette editor, for instance, allows the user to set the default colors used by Workbench.

PREFS IN MODERATION

Every additional selection in your application means additional complexity for the user. Before writing any preference controls for your application, consider these two warnings:

Make sure that a preferences item adds real value in the eyes of the user.

Don't add a preferences item just because you couldn't reach a decision. It's very easy to settle a dispute about how something should work by adding yet another preferences item.

Don't add a preferences item just because it's technically feasible to do it. Make sure it adds real value in the eyes of the user. (Note: we speak from experience here. Trust us on this one.)

The 90% Rule

If 90 percent of your users prefer A, and all but the most obstinate of those who prefer B could live with A, don't bother making it a preferences item. Remember, it is far easier to add a control next version than it is to take one away.

TWO CHOICES

Two basic design decisions confront you when it comes to preferences.

Duplicate System Prefs?

First, if one of the system preference editors contains an item relevant to your program, you must decide whether to duplicate the control in your program so that the user can override the system setting locally. For example, if you are writing a terminal package, you can use the baud rate the user has set with the Serial preferences editor or you can provide a control that allows the user to set the baud rate inside your application.

Here's a guideline: if in the course of using your program, a user is likely to change an option, provide controls local to your application even if the control duplicates an item in a system preference editor. In that case, your controls should override the system preferences within the scope of your application.

Whether you add controls local to your application or not, your program should look at the defaults the user has set up in the Workbench preferences editors and use them as the starting values for the preferences controls in your application.

Your application should look at the defaults the user has set up in the Workbench Prefs editors and use these as the starting values in your application.

Menu or Editor?

The second design concern: you must decide where to put the controls for application-specific preferences. You can allow the user to access them through your application's Settings menu (see Chapter Six), or you can have an application preferences editor in your application's drawer; or you can provide both.

In general, try to use your application's Settings menu rather than a standalone editor to set preferences. Any menu item could lead to submenus or a series of requesters for more complex settings.

However, there are times when a preferences editor makes more sense than

the Settings menu. For instance, your application might be a compiler that is used only from the Shell, or your application may have environment controls that need to be adjusted only when the user's hardware setup changes. In these cases, an application preferences editor may be better than using the Settings menu.

System-Wide Preferences

If for some reason, your preferences will control options system-wide, the editor can go in SYS:Prefs. One example of this is a stand-alone PostScript printer driver.

THE SYSTEM PREFERENCE EDITORS

Here's a list of the system preference editors and the system defaults they control. You should know what the system controls provide before writing any controls of your own.

Font Specification of default screen, system and Workbench fonts. The file format is IFF FORM PREF (FONT chunk). The filenames are screenfont.prefs, sysfont.prefs, and wbfont.prefs, respectively.

IControl Intuition-specific control items, including verify timeout and command key definitions. The file format is IFF FORM PREF (ICTL chunk). The filename is icontrol.prefs.

Input Mouse and keyboard control items. The file format is IFF FORM PREF (INPT chunk). The filename is input.prefs.

Overscan Standard video and text overscan areas for the various modes

supported by the system. The file format is IFF FORM PREF (OSCN chunk). The filename is oscan.prefs

Palette Color selections for the Workbench screen. The file format is IFF FORM ILBM (the main chunk is CMAP). The filename is palette.ilbm.

Pointer Design of the mouse pointer image. The file format is IFF FORM ILBM. The filename is pointer.ilbm.

Printer Printer text preferences and printer driver selection. The file format is IFF FORM PREF (PTXT chunk). The filename is printer.prefs.

PrinterGfx Printer graphic preferences. The file format is IFF FORM PREF (PGFX chunk). The filename is printergfx.prefs.

Screen Mode Display information such as the Workbench display mode and the raster dimensions. The file format is IFF FORM PREF (SCRM chunk). The filename is screenmode.prefs.

Serial Serial port definitions including baud rate, handshaking and parity. The file format is IFF FORM PREF (SERL chunk). The filename is serial.prefs.

Time System and real-time clock date and time. No file is used. Data is stored into the system and the battery-backed clock/calendar.

WbPattern The backdrop patterns used in Workbench and its windows. The file format is custom. Names are wb.pat and win.pat respectively.

Storage of System Settings

Users can change system-wide settings for their current session by clicking the Use gadget in an editor, or they can change the permanent settings by clicking the Save gadget.

The system stores these settings files in four different locations. (Your application should follow these steps only if it installs a system-wide preferences editor in the Prefs drawer. Otherwise, you should follow the conventions outlined in the section beginning on the next page.)

Listed at right is how the system stores settings. Your application should follow the steps outlined on the next page in the "Accessing Settings Files" section.

- Defaults are imbedded in the system preferences editors themselves.
- If the user modifies a preference item and selects Save on the requester, the settings are stored in the preference archive ENVARC:sys/<filename> and ENV:sys/<filename> (see the list above for the filename).
- If the user selects Use on the requester (instead of Save), the current settings are stored only in ENV:sys/<filename>. The archived settings in ENVARC:sys remain unchanged. ENV: does not survive a reboot.
- Alternative settings called presets are stored in the Workbench: Prefs/Presets drawer or in other user-specified places.

Preference Presets

Preferences presets are alternate versions of the recognized preferences files. Their purpose is to support a variety of user configurations that can be "turned on" at the user's discretion.

For example, the default definition of the Amiga's mouse pointer is archived in ENVARC:sys/pointer.ilbm. However, the user may have an alternate pointer he favors when working with a paint program. An alternate image can be designed in the Pointer editor and saved in Workbench:Prefs/Presets.

Once it's saved as a Preset, the user can activate the alternate image at any time by double-clicking on its icon. (The icon's default tool is the Pointer preferences editor.) The preset can also be activated from the Shell or from the editor. The original version will still be available in the ENVARC:sys directory.

CONVENTIONS FOR YOUR APPLICATION'S PREFS

The general rules: whenever possible, your application should allow the user to save his preferred settings and should initialize on startup to the user's settings – whether they are the user's default settings or another settings file specified by the user in a startup argument.

Also, whenever possible, the settings controls should be contained within your application and accessed via the Settings menu. (See Chapter Six for information about the Settings menu.)

Format

There is no strict structure for preferences files. They may be ASCII, IFF, binary, or any form suggested by the preferences data therein. Many of the system files are made up of structured binary data and are saved with only a small amount of identifying header information. Others, such as pointer.ilbm, are actual IFF ILBM files.

Preferences files may be ASCII, IFF, binary or any form suggested by the data. Using an IFF form is recommended.

It is recommended that you use an IFF form for preferences files. If you wish to use a new or modified IFF form to contain your preference data, you should register the form with Commodore Applications and Technical Support.

Accessing Settings Files

Keeping the preferences operations simple yet viable for all types of users gets somewhat tricky for the application designer. To help with this, the following rules for accessing user preferences settings were devised.

Upon startup, your application should look for its application preferences settings in the following places, and in the following order. You should use built-in defaults only if no user-specified settings are found.

Note: keep in mind that allowing the user to save individual aspects of your program, rather than a general Save Settings, will change this scenario and the

next one about saving settings files. Instead of looking for an individual file containing the settings, your program would be searching for a subdirectory of preferences settings. That directory would contain files such as pointer.ILBM or beepsound.8SVX.

Follow these steps to access settings files.

1. Look first in a file specified in startup arguments. Users should be able to specify a filename and path in either the icon's Tool Types field or on the command line in the Shell. The keyword in both cases would be SETTINGS. The specified file would be a settings file or directory saved during a previous session with your program.

   This will allow users without hard disks to save their settings wherever convenient – perhaps on a separate disk (some boot disks are too full already). It will also allow networked users to specify their individual settings files in project Tool Types or on the command line.

2. If no SETTINGS argument is specified, look for the file <basename>.prefs in your application's directory. Accessing and storing the settings file in your application's directory instead of the SYS: directory may benefit users without a hard disk whose boot disks may be full. In most cases this will effectively be the default since most users will not specify a settings file in Tool Types or the command line.

3. If the settings aren't found in places 1 or 2, look for your application preferences settings file in ENV:<basename>/<basename>.prefs.

4. Use the built-in default settings.

Saving Settings Files

The general rule here is to save any new settings chosen by the user to the place from which the settings were loaded – but allow the user to specify another place.

However, this changes if the settings were loaded from your built-in defaults – which would be the case if this is the first time a user is using your program. In this case you should go through what are basically the same steps for loading a settings file.

Remember that if you allow individual aspects of your application to be saved, this changes from a search for a file to a search for a subdirectory and then a file or list of files.

Follow this order when your application is saving settings files.

1. As stated in the general rule above, save to the same path and file from which you loaded the settings.

2. If a file was not specified, save to the file called <basename>.prefs found in your application's directory. In most cases, this would be far enough to look and a good solution, but if your application is being used on a network, the network moderator would most likely set this file to be write-protected. She wouldn't want every user saving his or her settings on top of those optimized for the network.

3. If you are unable to write to the file in step two, save the settings in ENVARC:<basename>/<basename>.prefs and in ENV:<basename>/<basename>.prefs.

Application Preference Editors

The common way to handle user preferences is through a Settings menu item, but in more complex cases or when the settings will affect the overall system, you may want to use a stand-alone preferences editor.

Complex cases include those where a number of invocations of your program are running concurrently or sequentially, and your application needs to know the settings from previous invocations.

An example of a case where your program's settings affect the overall system is software for a multi-serial card.

The design guidelines for preferences editors given here will ensure that each one presents a familiar and consistent user interface. Be sure to include all of the following standard operations in your application preferences editors. See Chapter Two as well for general design guidelines.

The design guidelines given here for preference editors are intended to ensure that each has a familiar and consistent user interface.

Editor Startup

Every application preferences editor should have a set of reasonable default settings imbedded in the editor code. When the editor is started it should display the current internal settings unless a preset filename was given as an argument. If no preferences data is found, then revert to the default settings embedded in the code itself.

Editor Gadgets

Every application preferences editor should open a window with at least three gadgets: Save, Use and Cancel. The gadgets should be near the bottom of the window (Save on the left, Use in the middle, Cancel on the right). Here are the functions the three gadgets should perform:

Save Use the new preferences settings the user has chosen and save the data to the assigned file. By default, it should be stored to the file ENVARC:<basename>:basename.prefs and ENV:<basename>/<basename>.prefs. The editor should terminate. Note that only

system editors should store their data in the ENV:sys and ENVARC:sys directories.

Use Use the new preferences settings the user has chosen and save the data to ENV:<basename>/<basename>.prefs. The editor should terminate. Because no change is made to the copy in ENVARC:<basename>/<basename>.prefs, any changes the user makes this way will be lost if the system is reset.

Cancel Exit the editor. Any changes in preferences settings the user has chosen should be ignored.

Note: applications with a stand-alone preferences editor shared by default should read in their settings from ENV:<basename>/<basename>.prefs and use file notification to watch for changes to that file.

Editor Menus

Every application preferences editor should have at least three menus, a Project menu, an Edit menu and an Options menu. The Project menu should contain these menu items:

Open Load preferences data from a preset file. A requester should appear that allows the user to specify the file's name as well as its path. The command key is Right-Amiga-O.

Save As Save preferences data in a preset file. A requester should appear that allows the user to specify the filename and path. The new file can be used as a preset. The command key is Right-Amiga-A.

Quit	Exit the editor without saving. The command key is Right-Amiga-Q.

The Edit menu should contain these items:

Reset to Defaults	Reset the preferences editor to the default values embedded in the editor code.
Last Saved	Reset the editor with the data from the preferences archive data file in ENVARC:<basename>/<basename>.prefs.
Restore	Reset the editor to the settings that were in effect when it was started.
Undo	Undo the most recent change the user made to the preferences settings.

The Options menu should contain at least this one menu item:

Create Icons?	Controls whether or not the editor will save a project icon with each preset saved.

Arguments

Editors should also accept command line arguments when executed from the Shell. When started from the Shell, the editor will not always open its window. Instead, the editor should perform the action as detailed in the arguments.

For both Shell usage and Tool Types usage, the command template should be of the form:

```
FROM,EDIT/S,USE/S,SAVE/S,PUBSCREEN/K
```

- FROM tells which preset file to use. If this argument is left out, use the archived preferences settings in <programdirectory>:<basename>.prefs or revert to the defaults coded into the editor itself.
- EDIT opens the editor window and loads the preferences data in the file named in the From argument. This is the default switch. If no From is specified, the window should open with the defaults.
- USE performs a "Use" on the data found in the data file named in the From argument. Do not open the editor window.
- SAVE performs a "Save" on the data file named in the From argument. Do not open the editor window.
- PUBSCREEN opens the window on the named public screen.

Here's an example command line:

```
1> Serial FROM SYS:ENVARC/Prefs/Presets/myserial.pre USE
```

With this command, the serial device Preferences editor is run invisibly from the Shell. Without opening its window, the editor loads the serial preset file named myserial.pre, performs the Use function and then exits. If this command is placed in the Amiga's Startup-sequence, it will override normal default preferences with the preset named in the From argument.

ICONS

The user should also have ultimate control over what icons look like.

Under past releases of the operating system, most applications hardcoded their project icons into the program. Under Release 2, you should name your default project icons in a standard manner and store them where the user can access them.

These rules for icon naming and storage are designed so the user can control the look of your program's default icons.

Naming Conventions

Use the following naming convention:

```
def_<icon type>.info
```

The placeholder <icon type> indicates the type of file represented. For example, the types of icons in the system are: disk, drawer, tool, project and trashcan.

The default icons for IFF files should use the FORM name as the <icon type>. For instance, a default icon for an IFF picture would be def_ILBM.info since ILBM is the FORM name. See Chapter Eleven for more information on the IFF standard.

Storage and Access Conventions

Store your default icons in ENV:<basename> and in ENVARC:<basename>. If the user wants to change the default icon she can find it in that directory.

When your application needs an icon, direct it to look in ENV:<basename> first, then ENV:Sys for the system default icon for that type of file. If it fails to find that, it should revert to the system's default project icon.

Here's an example: when the user saves a text file in an application named MystrEd, the application will first look in ENV:mysed for an icon file named def_TXT.info. Finding none, it will then look in ENV:Sys for a file named the

same. None is found there, so it uses the system's default project icon.

If however, the MystrEd user creates a default TXT icon in IconEdit and stores it as ENV:mysed/def_TXT.info, that icon should be used as the default.

NOTIFICATION

Notification is an Amiga facility whereby an application may be informed whenever a change is made to a file of interest. Through notification it is possible for an application to find out whenever the user has made a change to any preferences data file. This allows the application to make an adjustment to reflect the new preferences settings. See Chapter Eleven for more information.

Notification can be used to let your program know when the user makes changes to a system-wide preference editor.

Since an application has to handle its own preferences, there is not much point in getting notification about changes to application preferences files. However, changes to system preference files may be relevant. For instance, a publishing program may want to be notified whenever the user changes the printer graphic setting from black-and-white to color.

Notification is a new facility in Release 2 of the Amiga operating system. Under previous versions of the operating system, an application could ask Intuition to send it a message whenever system preferences data changed. This method still works for Preferences items that existed under Release 1.3.

In either case, the important thing to understand is that your application is *not* expected to respond to every change made to system preferences. This is an extra feature you may add to your application if you believe it is warranted. ▲

appendix a
GLOSSARY

The purpose of this glossary is three-fold:

First, it is for you, the reader, if you are unsure as to the meaning of a term in this manual.

Second, the technical writer in your organization can use it as a reference/style guide when writing documentation. For that reason, some terms are included in this glossary that haven't been discussed within the main body of this book. The capitalization, spelling and definitions given here agree (when applicable) with Commodore's Release 2 Users Guide, *Using the System Software.* If you follow this style, your manual will better agree with the main user manual for Release 2 of the Amiga system software. For instance, the definition for serial may be obvious to you, but by consulting this guide you can see that Commodore's Release 2 user's manual did not capitalize it.

Third, it is for you, the developer, to let you know where certain functions are found in the system and what version of the system software contains that function. For instance, the concept of AppIcons first appeared with Release 2 of the system software and is supported in workbench.library. If you look at the glossary entry for AppIcon you'll see it suffixed with *Release 2 workbench.library.*

Note: The codes (v) and (n) refer to verb and noun, respectively. A word may have different spellings and/or meanings depending on whether it is being used as a verb or a noun.

a

absolute pathname
The explicit identification of a file or directory – one that includes the device or partition name and any directories that lead to the file.

action gadget
A box in a window that lets you choose an operation to be performed by selecting the box. Common action gadgets are Continue, Save and Cancel. Also called *buttons*.

active
Currently selected; such as "the active window."

AmigaDOS
The disk operating system (DOS) used by Amiga computers.

AppIcon
An icon on Workbench that allows the user to pass arguments to an application. For instance, if a text processor has an AppIcon on Workbench, the user could drag the icon for a text file onto the AppIcon and that file would be loaded into the application. *Release 2 workbench.library*

application gadget
Any of a number of programmed graphic images that appear within programs and can be manipulated with the mouse to perform a certain function. Under Release 2 of the operating system, standard application gadgets are available to the developer through GadTools. Previously, you would have to create application gadgets within your program.

AppMenu
An AppMenu allows the user to add a custom menu item to the Workbench Tools menu. *Release 2 workbench.library*

AppWindow
A window on Workbench that allows the user to pass arguments to an application. For example, if a text processor has an AppWindow, the user can drag the icon for a text file into the AppWindow and that file would be loaded into the application. *Release 2 workbench.library*

archive
1. (n) A backup copy of a file or files. 2. (v) To copy files to disk or tape for backup purposes.

ARexx
A text-based language that, along with Workbench and the Shell, serves as a built-in user interface for the Amiga. ARexx has two main uses. As a scripting language, it can operate internally with applications. It can also be used to operate two or more applications that may or may not be inherently compatible. In your documentation you should note that ARexx is a trademark of William S. Hawes.

argument
An additional item of information, such as a filename, value or option, included along with a command. This information determines the exact action of the command.

argument passing
Specifying parameters for a program or command to follow. On the Workbench this can be handled through the Tool Types and Default Tool fields of an icon, or through an AppWindow, AppIcon or AppMenu. More traditionally, arguments can be passed via a command line in the Shell.

assign
To link a directory name to a logical device name, with the ASSIGN command, so that programs which use that directory can look for one device name rather than having to search through several volumes for the directory. For instance, the RAM:T directory is commonly assigned to the device name T:.

attributes
A series of flags stored with every file. Attributes indicate file type and control the file operations (read, write, delete, etc.) permissible on the file. Also called *protection bits*.

autoscroll
To automatically move a screen when the pointer reaches the edges of the currently viewable area.

background process
A program that is started from the Shell with the RUN command. The program does not take over the Shell but is run in the "background."

back up
(v) To make a backup copy.

backup
(n) A copy of a file on disk or tape used to replace lost data.

basename
A specific and non-conflicting name for an application that should be used by the developer when naming accompanying files and ports such as the application's ARexx port and the file where application-specific preferences are stored. In most cases, the basename should be the same as the name of the application's executable.

bitmap
An image made up of pixels.

boolean
Having two possible states: on or off, true or false, yes or no.

Bridgeboard
An expansion board made by Commodore that allows the Amiga to emulate PC-compatible computers.

brush
An IFF graphics file; usually a section cut from a full-sized picture.

buffer
A temporary storage area.

C

check box
An application gadget used to let a user turn an option on or off. When a check mark appears in the box, the selection is considered to be "on." *Release 2 GadTools*

Chip RAM
The area of RAM accessible to the Amiga's custom chip set used for graphics and sound data.

choose
In terms of using the mouse, choose is what the user does with menu items. *Select* is what he does with icons.

clear
1. To change a bit or flag to 0, off or disabled state. Opposite of *set*. 2. To erase a screen or window display.

CLI (Command Line Interface)
A means of communicating with a computer by issuing commands from the keyboard. The program that allows this on the Amiga is called the Shell and, along with Workbench and ARexx, is one of the three built-in interfaces. Before the Shell was available, the program used was called the CLI.

click
To press and release a mouse button.

close
To remove a window from the screen.

close gadget
A system gadget used to close windows. It appears in the upper left corner of the window.

cold reboot
To reset the Amiga by turning the power off, waiting 20 seconds, then restoring power.

color correction
A printing option, selected through Prefs' PrinterGfx editor, that tries to better match the colors of a printout to the colors on the screen.

color selection gadget
A gadget from which you can choose one of several displayed colors. Also referred to as the palette gadget. *Release 2 GadTools*

command history
A feature of the Shell that allows you to recall previously entered command lines by using the cursor keys.

command template
A line of text showing how a command and its arguments should be used. When a user types "<command> ?" in the Shell, he should be shown the template for that command. *Release 2 dos.library ReadArgs()*

Commodities Exchange
A system that simplifies the process of writing programs which monitor the input handler system – these programs can respond to hot keys, take actions based on mouse action or inactivity, or even modify the input stream as it goes by.

condition flag
A variable that contains a return code value indicating the success or failure of command execution.

console window
A window used for the input and output of text.

context cue

A graphical hint to the user that where he is in the system or application has changed. When a user chooses New from a Project menu, for example, a new window opens. The new window is a context cue.

contiguous

Continuous; consisting of a series of adjacent items. Contiguous memory is a block of memory.

Control-key combination

A key combination that performs a special function, entered by holding down Ctrl while pressing another key on the keyboard. Some Control-key combinations are executed as soon as they are pressed, such as when Ctrl-C is used to abort the execution of an AmigaDOS command. Some produce a reversed character image and have no immediate effect.

cursor

A highlighted rectangle used in the Shell and some applications to indicate text position.

cursor keys

The four keys with directional arrows on them found below Del and Help on the keyboard.

custom icon

An application-specific gadget that can be moved and manipulated. Example: the note object in some music applications.

cycle gadget

An application gadget that allows the user to select one of several options. One option is displayed at a time and, as the gadget is selected, the other options become visible. The displayed option is the selected option. Similar to radio buttons but this takes up less space in a window or requester. *Release 2 GadTools*

d

dead key

A key, or key combination, that modifies the output of the next key to be pressed. For instance, on an American keyboard, Alt-H will superimpose a caret (^) symbol over the next applicable key to be pressed.

default

A value or action assumed if the user has not specified anything else.

Default Tool

An argument passing device used with project icons on the Workbench. When a project icon is double-clicked, the application specified in the Default Tool field of that icon's Information window will automatically load and run.

delete

To erase or discard a file, buffer or other stored item.

delimiter

A character that marks the beginning and end of a string.

depth gadget

A system gadget for moving a window or screen in front of or behind other windows or screens. It appears in the upper right corner of the window or screen.

deselect

This term was used in the Release 2 user documentation rather than *unselect*.

destination

The device, directory or file that is receiving information. For instance, in EDIT, the file that the revised text is being sent to is the destination file.

device

A physical mechanism, such as a printer or disk drive, or a software entity (logical device), such as CON: or NIL:, used as a source or destination for information. In the Release 2 user manual, hard disk and disk drive designations such as DF0: were capitalized.

device handler

Files that act as intermediate stages between AmigaDOS and physical devices, such as the Port-handler file in the L: directory which handles the interface for the PAR:, SER: and PRT: devices.

directory

A subdivision in a computer's filing system. Directories are represented on the Workbench as drawer icons.

disk

A medium for storage of computer data. This term was used in the Release 2 documentation rather than *diskette*.

display box
An application gadget, usually under a scroll gadget or next to a selection gadget, that displays the current selection but doesn't allow the user to edit it.

double-click
To press and release the mouse's selection button twice.

drag
To move an icon, window, gadget or screen across the display by pointing to the object, holding down the selection button and moving the mouse.

drag selection
The process of selecting several icons at once by holding down the selection button and using the mouse to draw a dash-rule box (marquee) around the icons to be selected. When the button is released, all the icons in the marquee will be selected.

drawer
A subdivision of a disk storage area. A drawer corresponds to an AmigaDOS directory.

drive name
The name assigned to a floppy disk drive or hard disk such as FH1: or DF0:. It's the same as the device name.

dump
A printout of the image displayed on the screen.

e

ECS (Enhanced Chip Set)
The upgraded versions of the Amiga's Agnus and Denise coprocessor chips. The ECS offered new display modes and expanded existing graphics capabilities. Many of the benefits of the ECS are available only in conjunction with Release 2 of the operating system.

ENV:
A directory where environment variables and user preferences are temporarily stored. Short for "environment."

ENVARC:
Similar to ENV: but it will survive a reboot. Short for "environment archive."

environment variable
A variable used by AmigaDOS to represent a string or a value.

error code
A number identifying an error that has occurred during execution of a command or program.

executable
The name for an application which the user types into the Shell in order to run the program. The executable should be one word – short, but long enough to prevent it from conflicting with the executables of other programs.

extended selection
The process of selecting several icons at once by holding down Shift while selecting each icon with the mouse.

FastFileSystem (FFS)
An enhanced Amiga file system usable with both floppy and hard disks. A volume is formatted as either FFS or OldFileSystem (OFS).

Fast RAM
General memory used by programs and data; as opposed to *Chip RAM*.

fatal error
Describes an error serious enough to halt the process that caused it.

field
1. The screen area behind the text under a Workbench icon. The color of the field can be changed with the Font editor. 2. An area in a requester where a text string can be inserted. A database, for instance, will often have multi-field requesters.

filename
Use one word in your documentation. Note: use brackets when information should be substituted. In this example, <filename> is just a placeholder for the actual filename.

flag
A status indicator variable with a limited number of possible states.

format
To prepare a disk for use with the Amiga. Use this term instead of *initialize* when referring to disks.

function keys
Keys at the top of the Amiga keyboard, labeled F1 to F10, that can be programmed to perform special tasks.

g

gadget
Any of various programmed graphic images that may appear in a window, requester or screen, that can be manipulated with the mouse to perform a certain function.

GadTools
A toolkit, available to developers under Release 2 of the operating system, that supplies pre-programmed, standard application gadgets for use within applications.

ghosting
Superimposing a pattern of dots in the shadow color over disabled menu items or gadgets. This gives the user a visual cue that the item is unavailable. Intuition will do this automatically to all standard menu items and gadgets that your program disables.

graphic user interface (GUI)
A visually oriented system allowing you to tell a computer what to do by manipulating graphic symbols rather than by typing in commands. Often, the GUI employs a metaphor for ease of understanding. The Workbench is the Amiga GUI.

h

hard disk
Use this term rather than *hard drive*.

history buffer
A section of memory that stores the most recent commands for a given Shell.

hold down
To continually press a mouse button until instructed to release it.

hot key
A key or key combination used by Commodities Exchange programs to open a hidden window.

i

icon
An image appearing on the screen to represent a disk, drawer, project or tool. Icons can be moved and selected with the mouse to allow you to work with the items they represent.

icon drop box gadget
The drop box glyph in an AppWindow. *Release 2 workbench.library*

IFF (Interchange File Format)
The standardized format in which the Amiga stores data for such things as text, graphics and sound. Such a standard is useful for data sharing between applications. *iffparse.library (can be used with Release 1.3 and Release 2)*

.info file
A file containing the image and position data for an icon.

Information window
The window presented on Workbench when a user selects (clicks once on) an icon and chooses "Information..." from the Icons menu.

Inter-Process Communications (IPC)
The means by which two or more applications can operate in conjunction with one another regardless of whether they are inherently compatible. On the Amiga, this can be accomplished with ARexx.

Internal command
Refers to an AmigaDOS command that is built into the Shell, rather than loaded from disk.

Intuition
The collective term for the GUI toolkit and function libraries that contain the elements necessary for you to build graphic interfaces for your Amiga applications.

K (Kilobyte)
1024 bytes. The abbreviation used in the Release 2 user's manual; e.g., "512K".

k

keymap
A file that determines the arrangement of characters on the keyboard and determines the meaning of each key. Different languages have different keymaps.

keystrokes
In the Release 2 user's manual, alphabetical keys are specified with an upper-case letter and the word *key* is not used. Example: *Press Q and then press Return.* Non-alphanumeric keys appear in the manual as they do on the keyboard (Esc, Del, Alt, Ctrl, Caps Lock, Help) with the exceptions of Backspace, Return, Shift and the cursor keys. Key combinations are separated by a hyphen and upper-case letters are specified by the Shift combination. Example: *Ctrl-O will move the cursor to the end of the line, but Ctrl-Shift-O will move the cursor to the end of the file.*

keyword
A word recognized by a command as identifying an argument or specifying an option. If the user needs to type the keyword on the command line along with its argument, "/K" should follow the keyword in the command template.

Kickstart
Software that is read from disk and used to boot the Amiga. Also refers to the portion of the OS that is in ROM.

l

library
A related set of functions and collections of data that can be shared by various programs. For instance, the Commodities.library in the LIBS: directory is used by all the Commodities Exchange programs.

link
A file that is a pointer to another file. When the original file is called, the linked file will be used.

m

macro
A single command that represents a sequence of commands. ARexx should be the macro language used by applications on the Amiga.

MB
The abbreviation for megabyte (1,048,576 bytes) used in the Release 2 System Software manual.

memory
The Amiga has both Chip (graphics) and Fast (normal) RAM, as well as 512K of ROM memory.

menu
A list of on-screen options, displayed by using the menu button, from which users can choose commands that control a program.

menu bar
The list of headings that appears across the top of the screen when the menu button is held down. When the menu button is not depressed, the visible bar is called the title bar.

menu button
The right mouse button.

menu item
An option that appears in a menu.

modified project requester
A requester that prompts the user to save the project before continuing. It should be presented when the user chooses an action (such as Quit Program) that would cause his currently unsaved work to be lost.

Moniterm Viking
A high-resolution monitor with a paper-white display especially useful with CAD or DTP applications. A similar monitor sold by Commodore is the A2024. When used with the Amiga, these monitors work in a special mode that tiles a number of high-resolution screens together to make one large, high-resolution display.

monospaced font
A font in which each character takes up an equal amount of space.

MountList
A text file in the DEVS: directory that contains information about devices that have been attached to or installed in the Amiga.

multiscan
A type of video monitor that can accept several different scan rates.

multitasking
The ability to perform more than one operation, or task, at a time.

mutually exclusive
Only one option can be chosen from a given group of options. For example, in a group of radio buttons, one and only one is always selected. When a user clicks on an unselected radio button, the one that had been highlighted becomes unselected.

notification
A means by which an application can receive a message whenever a specified file is modified. *Release 2 dos.library*

null string
An empty string. Null strings are commonly used in text editors to delete information. If the user replaces a word with a null string, the word is deleted.

o

offset
To shift or move over.

open
To make the selected object available for use. When the user opens a disk or drawer icon, its contents are displayed. When the user opens a project or tool icon, a program is started.

overscan area
The normally unused area surrounding a standard-size screen. The Overscan Preferences editor allows the user to expand her screen to fill this area. *Release 2 Intuition QueryOScan()*

overwrite
To write information to a file or disk, replacing any information that previously was stored there.

p

parallel
An interface port that transfers data one complete byte (8 bits) at a time, contrasted to a serial interface which sends a single bit at a time. The Amiga has an extended parallel port to which a printer is often connected.

parent
The window or directory from which another window, directory or file was generated.

parsing
When the system examines and interprets arguments so the appropriate operation can be performed. *Release 2 dos.library ReadArgs()*

partition
A distinct section of a hard disk.

path
The series of device, directory and subdirectory names that defines the location of a file.

pattern matching
An AmigaDOS feature that lets the user specify file and directory names by using wildcards and other tokens. *Release 2 dos.library MatchFirst() MatchNext() MatchPattern()*

peripheral
An external hardware device.

pitch
The number of characters printed in a horizontal inch.

pixels
The dots of light that make up the Amiga screen display. A pixel is the smallest unit of display information on a given screen.

pointer
An image on the screen that moves as the user moves the mouse. A default pointer, which can be redefined by the user, is included in the system. The pointer often changes to reflect processes and tools.

Preferences (Prefs)
A Workbench drawer containing editors that let the user configure and customize his Amiga environment.

press
The term used in Release 2 documentation when telling the user to hit a key. Example: Press F6 for your new macro.

priority
A variable determining the proportion of the Amiga's processing time that will be allotted to a given task. Each task has an independent priority. Task priority is set automatically but can be changed with the CHANGETASKPRI command.

process
A task that can communicate with AmigaDOS. Each process has a unique process number. Shell process numbers are usually displayed as part of the Shell prompt.

project
A file in which information created or used by a tool is stored.

prompt
A message or symbol, such as 1>, that indicates that text input to the computer is possible.

protection bits
Use *attributes* instead.

pseudo-icon
An icon that is displayed, when the Show All Files item is chosen from Workbench's Window menu, for an object that does not have a .info file.

public screen
A screen that can be used by any application. Workbench is a public screen. Release 2 allows an application to create its own custom public screen which can then be used by other applications. *Release 2 intuition.library*

pure
Describes a command or program that can be made resident.

q

qualifier
A key, such as Shift, Ctrl or Alt, that changes the Amiga's interpretation of a simultaneous or subsequent keystroke or mouse click.

r

radio buttons
A group of circular gadgets used to offer mutually exclusive choices. *Release 2 GadTools*

RAM (Random Access Memory)
Part of the Amiga's internal memory that can be used for data storage and is directly accessible by the CPU. Data in RAM is lost when the Amiga is rebooted or powered off.

Ram Disk
A section of RAM set aside to function as if it were a disk drive. Also known by its logical device name of RAM:.

RC
In ARexx, the field where numerical returns are placed. String values are placed in the RESULT field.

read
To retrieve stored information.

Read Only
If disk status is Read Only, the user can only look at the contents.

Read/Write
If disk status is Read/Write, the user can look at and alter the contents.

reboot

To reset the Amiga by pressing Ctrl, Left-Amiga and Right-Amiga simultaneously. Also called a *warm boot*.

redirect

To change the source or destination of a command's input or output from the default by using the special characters < or >.

relative pathname

The path to a file or directory that does not include the device or partition name that leads to the file.

Release 2

Use this term instead of *Version 2.0* or just *2.0*, unless in the future you are referring to a specific version of Release 2 of the operating system.

requester

A window that allows the user to control options, access files and confirm actions. Many requesters function just like other windows but others, called modal requesters, block input to the system until the user responds to it.

resident

Describes a command or program that has been copied into memory, with the RESIDENT command, for quicker execution. Only pure files can be made resident.

resolution

The number of pixels associated with a particular display mode. For example, a normal NTSC Hires screen has a resolution of 640 (horizontal) by 200 (vertical) pixels.

RESULT

In ARexx, the field where string values are returned. Numerical values go in the RC field.

return code

A numerical value, generated upon execution of a command, to indicate its level of success. The number is 0 if the command was successful and usually 5, 10 or 20 if there was a problem in executing the command. The return code value is assigned to the condition flag.

RGB (Red-Green-Blue)

A type of video signal in which the three primary color signals are sent separately. Standard Amiga output uses an RGB monitor.

ROM (Read Only Memory)
Permanent memory that is pre-programmed with system instructions and does not change. The contents of ROM are not affected by user commands or program operation.

root directory
The main directory on a volume. The root directory is at the top of the filing hierarchy and created when a volume is formatted. The root directory is specified by the volume name followed by a colon.

S

save
To write the current version of a file to disk. In many settings requesters the user is given the choices Save, Use and Cancel. Save, in this case, would cause the program to use those settings each time it opens.

screen
An area of the display that shares the same video attributes, such as resolution and palette.

script
A text file containing a series of commands that can be automatically executed to perform a complex or repetitive task. An example of a script is the Startup-sequence that is executed when the Amiga is booted.

scroll
To move through the viewing area of a window.

scroll arrows
Parts of a scroll gadget that can be used to move the viewing area continuously.

scroll bar
The highlighted area within the scroll box that can be dragged to display the hidden contents of a window. It changes in size to indicate the portion of the window that is currently visible.

scroll box
The shaded area within which the scroll bar can be dragged. You can click in the scroll box to move the scroll bar.

scroll gadget
A gadget that may appear in a window to let the user move through the viewing area of a window. A scroll gadget is made up of the scroll bar, scroll box and scroll arrows.
Release 2 GadTools

scrolling list
A gadget that allows the user to select an object from a variable list of objects. A scrolling list is made up of a scroll gadget, a view box and a text gadget. *Release 2 GadTools*

SCSI (Small Computer System Interface)
A standard interface protocol for connecting peripherals, usually mass storage devices, to computer equipment.

select
To choose an item to work with by pointing to it with the mouse, then pressing and releasing the selection button.

selection button
The left mouse button.

serial
An interface port that transfers data one single bit at a time, contrasted to a parallel interface which sends one complete byte (eight bits) at a time.

set
To change a bit or flag to its on or enabled state. Opposite of clear.

Shell
The command line interface used to send typed commands to the Amiga. One of the three interfaces built into the Amiga.

sizing gadget
A system gadget that allows the user to enlarge or shrink the size of the window.

slider gadget
A gadget that allows the user to select a value by dragging a rectangle up or down in a vertical bar. *Release 2 GadTools*

slider value
A number that appears next to a slider gadget to indicate the currently selected value.

snapshot
To save the positions of a window and/or the icons in it.

source
A device, directory or file that is supplying information.

stack
A special area of RAM reserved by a program for temporary storage.

Startup-sequence
An AmigaDOS script file, executed when the Amiga is booted, that helps set up the hardware and directory systems.

streaming tape
A high-capacity, mass-storage device that uses a magnetic tape cartridge to hold data; generally used to back up large hard disks.

string
A piece of text treated as a single unit.

subdirectory
A directory that is within another directory.

submenu
A secondary menu that appears when some menu items are highlighted. A menu item that produces a submenu should have the symbol » at the far right.

switch
A command keyword that turns an option on or off. If the keyword is typed onto the command line, the option is considered to be on.

syntax
The rules for the proper arrangement of commands, keywords and punctuation.

task
A software function spawned by a process.

text gadget
A rectangular box in which the user can type information such as a filename or command. *Release 2 GadTools*

3-D look
The technique used by Release 2 of the operating system which uses simulated light and shadow to create the illusion of depth and simultaneously give the user context cues. *Release 2 GadTools (provides 3-D gadgets)*

title bar
The top border of a screen or window that commonly displays the name of the screen or window. On a draggable window, this is sometimes referred to as the *drag bar.*

tool
A program that creates, uses or displays data.

Tool Types
A method for passing arguments used by the GUI. Tool Types is a field in the Information window of a project or tool icon where optional parameters can be entered.

Trashcan
A directory for storing files the user wants to delete.

type
In Release 2 documentation, this term was used instead of *enter* or *key in*. Example: *Type the macro name in the console.*

version
A number that identifies the edition of a program.

V

virtual screen
A screen that is larger than the actual display area of the monitor.

volume
A floppy disk or hard disk partition.

volume name
The name given to a disk or partition.

wait pointer
An image that appears in place of the normal pointer when an application is busy and cannot accept further input. The Workbench's image of a stopwatch was not accessible by developers at the time this manual was published.

W

wildcard
A symbol used in pattern matching to represent a range of possible values.

window
A rectangular screen area that can accept or display information.

Workbench
The Amiga's icon-based GUI.

write
To record data in memory or on a storage medium such as disk or tape.

write-enable
To allow information to be written onto a storage device.

write-protect
To prevent information from being written onto a storage device.

Z

zoom gadget
A gadget that may appear in the upper right corner of a window which allows the window to alternate between two sizes.

Zorro
The name for the expansion slot specification used by Amiga computers. Amiga 2000 and 3000 families contain Zorro II and Zorro III slots, respectively. ▲

appendix b

COMMODORE ADDRESSES

Commodore Business Machines Pty. Ltd.
67 Mars Road
Lane Cove
New South Wales 2066
Australia

Commodore Computer GmbH
Kinskygasse 40-44
A1232 Vienna
Austria

Commodore Computer N.V./S.A.
Europalaan 74
B-1940 St. Stevens-Woluwe
Belgium

Commodore Business Machines Ltd.
3470 Pharmacy Avenue
Agincourt, Ontario M1W 3G3
Canada

Commodore Data A/S
Jens Juuls Vej 42
DK 8260 Viby J/Aarhus
Denmark

Commodore France S.A.R.L.
150-152 Av. de Verdun
92130 Issy Les Moulineaux
Paris
France

Commodore Electronics Ltd.
Commodore Centre
2-12 Wing Kei Road
Kwai Chung, New Territories
Hong Kong

Commodore Italiana S.p.A.
Viale Fulvio Testi, 280
20126 Milano
Italy

Commodore B.V.
Kabelweg 88 1014 BC
Amsterdam
Netherlands

Commodore Business Machines (NZ) Ltd.
1-3 Parkhead Place
Albany
Auckland
New Zealand

Write to your local Commodore office for information about joining Commodore's developer program.

Commodore Computers Norge A/S
Postboks 109
Okem, N-0509
Oslo 5
Norway

Commodore Portuguesa Electronica, S.A.
Praça Infante D. Pedro, 13-b R/CH Do
Miraflores
1495–Lisbon
Portugal

Commodore S.A.
Principe de Vergara, 109
28002 Madrid
Spain

Commodore AB
Fagerstavaegen 7
Lunda Industriomrade
Spanga
Sweden

Commodore AG
Langenhagstrasse 1
CH-4147 Aesch
Switzerland

Commodore Business Machines U.K. Ltd.
The Switchback, Gardner Road
Maidenhead, Berks
United Kingdom

Commodore Business Machines, Inc.
1200 Wilson Drive
West Chester, PA 19380
United States

Commodore Büeromaschinen GmbH
Lyoner Strasse 38
Postfach 710126
6000 Frankfurt 71
Germany

INDEX

List of Illustrations

List of Tables

Amiga Programming

> $21.95 FPT USA
> $28.95 CANADA

AMIGA TECHNICAL REFERENCE SERIES

AMIGA® User Interface Style Guide

The Amiga computers are exciting high-performance microcomputers with superb graphics, sound, multiwindow and multitasking capabilities. Their technologically advanced hardware is designed around the Motorola 68000 microprocessor family and sophisticated custom chips. The Amiga's unique system software provides programmers with unparalleled power, flexibility, and convenience in designing and creating programs.

Written by the technical experts at Commodore-Amiga, Inc., the **Amiga User Interface Style Guide** provides an introduction to, and in-depth explanation of, the issues programmers must understand to create the best user interface for Amiga applications. The book includes:

- the design principles and metaphors underlying Intuition, the Amiga's graphical user interface
- guidelines for programs that use the Amiga's high-performance ARexx and Shell interfaces
- detailed specifications on how to arrange the elements of the Amiga's user interface to make applications consistent, powerful, and easy to use.

This new addition to the Amiga Technical Reference Series covers the entire Amiga line of computers, including the Amiga 3000, and the most current version of the system software—Release 2.

For the serious programmer who wants to take full advantage of the Amiga's impressive capabilities, the **Amiga User Interface Style Guide** is the definitive source of information on designing the front end to Amiga applications.

Cover design by Hannus Design Associates

52195>
9 780201 577570
ISBN 0-201-57757-7
57757

Addison-Wesley Publishing Company, Inc.