AMIGA WORLD

Official AmigaVision Handbook

The only authoritative guide to AmigaVision and multimedia

Features in-depth, detailed primers on programming, technical authoring, & design

Plus: interface and database design, tips and techniques

Information on versions 1.31, 1.53, and 1.7

Heavily illustrated—with over 150 screen shots

by Louis R. Wallace

Foreword by John Campbell, Sr. Manager, Software Applications, Commodore International

"This book puts the power of AmigaVision in your hands, and I... highly recommend it." —John Campbell, Commodore

IDG BOOKS

AMIGA WORLD

Official AmigaVision Handbook

By Louis R. Wallace
Foreword by John Campbell
Senior Manager, Software Applications
Commodore International

IDG BOOKS
IDG Books Worldwide, Inc.
San Mateo, California 94402

AmigaWorld Official AmigaVision Handbook

Published by
IDG Books Worldwide, Inc.
155 Bovet Road, Suite 730
San Mateo, CA 94402
(415) 358-1250

Library of Congress Catalog Card No.: 91-070291

ISBN 1-878058-15-0

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Editor-in-Chief: Michael E. McCarthy
Associate Editor: Jeremy Judson
Production Manager: Lana Olson
Edited by Barbara Gefvert, Senior Editor, AmigaWorld
Technical edit by Cathy Godfrey, Authoring System Support Specialist, Commodore International
Interior design by Bill Hartman
Production by Hartman Publishing

Distributed in the United States by IDG Books Worldwide.
Distributed in Canada by Macmillan of Canada, a Division of Canada Publishing Corporation.
For information on translations and availability in other countries, contact IDG Books Worldwide.
For sales inquiries and special prices for bulk quantities, write to the address above or call IDG Books Worldwide at (415) 358-1250.

Dedication

I dedicate this book to my wife Sharon and my children, Terra, James, David, and Lisa.

Acknowledgements

Like any author, I would like to thank a great many people in a variety of places who helped me make this book possible...

I would like to thank everyone at IMSATT Corporation for creating a product like AmigaVision. A lot of computer software has been written over the years, but only a very few programs have been as impressive to use and as elegant to behold.

John Campbell at Commodore deserves special thanks for having the vision to see the need for AmigaVision, and the perseverance to champion its development.

Cathy Godfrey of Commodore gets many thanks for always being there with an answer when I needed it, for listening patiently while I asked for enhancements, and sending updates when they became available.

Michael McCarthy and Jeremy Judson of IDG Books receive thanks for all their efforts in helping me keep going with this project.

Bob Ryan at *Byte* I thank for offering equal parts of encouragement, sympathy, friendship, and Chinese beer whenever I needed them.

All the staff at *AmigaWorld* Magazine earn thanks for making it a place I enjoy working at, even when I was tired from being up most of the night before.

Barbara Gefvert of AmigaWorld deserves my thanks; her skill and patience while editing this book has been tremendously appreciated.

Special thanks to my children, Terra, James, David, and Lisa, who put up with Dad spending even more time than normal in the study, and who never fail to make my eyes shine and my heart swell.

And finally, all my thanks and love go to my wife Sharon, who has always been there for me when I needed her and without whom I would never have found my way.

(The publisher would also like to thank Bill Murphy, without whom this book would not have been possible.)

About the Author

Louis R. Wallace is currently Senior Editor, Technology for *AmigaWorld* Magazine. Prior positions include Technical Manager and Senior Technical Editor of *RUN* Magazine as well as being a Research Chemist and Computer Specialist for the Veterans Administration. He has been writing in computer magazines as a staff member or contributing editor for about eight years, and over that period of time he has written hundreds of articles and reviews, appearing in such magazines as *AmigaWorld, RUN, inCider, PCResource, CD-ROM Review & Lab Reports, PC Games, Family Computing, Commodore,* and the *Gazette* (Italy).

Lou resides in Peterborough, New Hampshire with his wife, four children, and two dogs. When not working, Lou enjoys such outdoor activities as hunting, archery, and gardening and is well known for his enthusiasm for horror films.

About IDG Books Worldwide

Welcome to the world of IDG Books Worldwide.

International Data Group (IDG) is the world's leading publisher of computer periodicals, with more than 140 weekly and monthly newpapers and magazines reaching 20 million readers in more than 40 countries. If you use personal computers, IDG Books is committed to publishing quality books that meet your needs. We rely on our extensive network of publications — including such leading periodicals as *ComputerWorld, InfoWorld, MacWorld, PC World, Portable Computing, Publish, Network World, AmigaWorld,* and *GamePro* — to help us make informed and timely decisions in creating useful computer books that meet your needs.

Every IDG book strives to bring extra value and skill-building instruction to the reader. Our books are written by experts, with the backing of IDG periodicals, and with careful thought devoted to issues such as readibility, organization, use of icons, and illustrations to provide a quailty learning experience for you. Our editorial staff is a careful mix of high-tech journalists and experienced book people. Our heavy use of personal computers at every step in production means we can deliver books in the most timely manner.

We are delivering books of high quality at competitive prices, on topics customers want. At IDG, we believe in quality, and we have been delivering quality for 25 years. You'll find no better book on a subject than an IDG book.

Jonathan Sacks
President
IDG Books Worldwide

This book was printed on recycled paper, and can be recycled

Table of Contents

Foreword

By John F. Campbell
Senior Manager of Software Applications
Commodore

AmigaVision was developed to put the power of the Amiga into the hands of the masses. Without needing to know C programming, the typical user can now control the impressive capabilities of the Amiga. Yet AmigaVision also retains the power of a standard programming language so its uses would not be limited to simple linear presentations. This book is intended to help you make the most of all the functionality of AmigaVision.

When Commodore first introduced the Amiga 1000 in 1985, it was clear that the Amiga technology had several distinct advantages: multitasking (now becoming more common in most operating systems); strong graphic and animation capabilities; and video compatibility with NTSC and PAL standards and standards for mixing creative media (pictures, musical scores, digitized sounds, animation, text). The result was more power and flexibility for less money.

With all these differentiating talents, it soon became apparent that the Amiga could go far beyond the standard uses for a computer — business productivity, education, and games. With all its special capabilities, it could be used for multimedia applications that would have far more impact than standard computer-based applications.

It was equally clear that creating multimedia applications for this advanced machine was not for the inexperienced. Only Amiga C programmers could assemble such applications, and even for them it was a somewhat arduous task. Valuable time would be wasted trying to create applications which could have a powerful influence in many different markets. How could the problem be solved?

It was decided that an "authoring system" would need to be constructed, one which was easy yet powerful. It had to be iconic for ease of use, yet able to include a lot of specific information. It had to give a graphical representation, but not result in a strict flow chart that only a computer science graduate could love.

With these seemingly contradictory requirements in mind, we scanned the globe looking for a developer who had an authoring system, understood the Amiga, and shared our vision of both the Amiga and the multimedia marketplace. After a six-month search, we selected IMSATT Corporation which had done a highly successful script-based authoring system and now wanted to get involved with the Amiga because it could "do things we simply cannot do on other platforms."

After several false starts, our team developed an iconic, outline-oriented, open-minded authoring system which supported the standard file formats. It was iconic rather than script based so that it would be easy for users to create and change a simple presentation. It was outline-oriented because we found that when people needed to give a presentation or teach a topic, they normally thought in outline form. It was simple enough for everyday use in the business and education markets, yet it was open-minded so that if there were any capabilities that were not adequately taken care of in AmigaVision, the experienced user could add routines to provide those functions.

AmigaVision is the glue which brings together the creative materials that are generally produced with products specifically designed for that media. In this way we are supporting both the developer community and the end users who have already amassed their own software library and creative materials.

During the development process, we had the help of over 100 beta testers ranging from computer scientists at major universities to company presidents. One of those early testers was Lou Wallace, who helped us make sure that AmigaVision had the capabilities the marketplace would demand. Mr. Wallace has since written many AmigaVision applications and magazine articles, and has appeared on television and at trade shows demonstrating the uses and capabilities of AmigaVision. I've known Mr. Wallace for six years, and with his great depth of Amiga knowledge, I can think of no one better to author this book.

This book puts the power of AmigaVision in your hands, and I therefore highly recommend it. As the Amiga continues its leadership position in multimedia, AmigaVision will continue to evolve so that you can get the most out of your Amiga.

Introduction

Welcome to *AmigaWorld Official AmigaVision Handbook.* This book follows the publication of the *AmigaWorld Official AmigaDOS Companion,* which is now in its third edition. Both books are designed to enhance your Amiga computing experience and hopefully, they will become permanent additions to your Amiga library.

This book is the result of my experience programming with and writing about the AmigaVision authoring language. I have taken a strongly visual approach to illustrate the concepts given throughout this book. It promises to be the most thorough book on AmigaVision to be found anywhere.

My background as a programmer and software author is pretty traditional. As a science major in college I learned BASIC, FORTRAN, C, and assembly languages. While I was sometimes lucky enough to get access to a video terminal, much of the time I was forced to use an antiquated device known as a keypunch, or card punch. It was a painful process which I will never forget nor wish to repeat. Because entering my programs in this manner was not an easy or casual manner, I quickly learned to spend more time planning and designing my programs so I could spend less time typing it on the dreaded keypunch machine. This approach seemed to work, and is one I have continued to use over the years on a multitude of machines and a variety of languages.

In late 1989 *AmigaWorld* Magazine began working on an issue about multimedia, and as Senior Editor, one of my tasks was to write about multimedia authoring systems. Commodore was reportedly working on something new, some sort of "silly" icon-based computer language and I requested a beta copy of the software so I could try it out and write a little bit about it. That language was, of course, AmigaVision.

Since my programming background was rather traditional I really didn't expect much from AmigaVision. I was quite surprised, however, to find that AmigaVision wasn't silly at all, but instead was a remarkably effective multimedia application language that allowed me to create programs that would have taken much longer using a conventional language approach.

I was also pleasantly surprised with AmigaVisions user interface. Instead of the awkward and cumbersome interface I expected, I found a very professional and flexible interface that was easy to use and understand.

Even after I had finished using AmigaVision for what I originally set out to accomplish, I still found myself continuing to use it for one task after another. By the time AmigaVision was shipping, I had already developed a number of fairly impressive applications that combined all the elements of multimedia — text, sound, graphics, animation, and multimedia — in a manner I would have found impossible with any other authoring system. Because of its power and ease of use, AmigaVision has become an integral part of my programming library!

The manual supplied with AmigaVision, while better then most software manuals, does not delve very deeply into the basics of programming, and, because there have been several upgrades to the original release of AmigaVision, it is lacking adequate descriptions of many of the new features. Because of this, I decided to write the AmigaVision Handbook, with the hope I could offer more detailed information, point out undocumented features and a variety of useful topics not included in the original manual, as well as give beginning authors some help in learning to program with it.

Who Should Use This Book?

AmigaWorld Official AmigaVision Handbook is designed to be the definitive guide for the AmigaVision author or author-to-be. Whether you've been thinking of buying AmigaVision, or you have been programming with it for years, you'll find this book an essential source of information for creating impressive multimedia applications that combine text, sound, graphics, and animation.

The AmigaVision **Beginner** can take comfort in the fact that this book thoroughly covers getting started, setup, and how to configure AmigaVision to get the most from your hardware resources. The beginner will also be introduced to the elements that make up the term "multimedia," the art of authoring, as well as the various components that make up AmigaVision — in the first five chapters.

The **Intermediate User** will more fully develop his or her programming skills after reading the easy-to-follow command reference sections that discuss in detail such things as interrupts and database functionality.

The **Advanced User** can fine-tune his or her programming skills with the many hints, examples, and undocumented features found only in this book. The last section of the book is devoted to editing tools and programming techniques — a must-read for the serious programmer. Also offered in the appendices are the state-of-the-art hardware and software products you can purchases to work with your system, as well as a discussion on the recent AmigaVision upgrade v1.70.

How This Book is Organized

This book is divided into four easy-to-follow sections:

I: The Basics of AmigaVision
II: Command Reference
III: Using AmigaVision Editors, Tools, and Programming Techniques
IV: Appendixes

Section I details how to get started with AmigaVision, the second section thoroughly covers the various command icons necessary to create an application, and the third section discusses editing tools and programming techniques. The advantage of this structure is that you can stop at any given section and make sure you have grasped the information before going on to the next, more challenging section.

Section One — The Basics of AmigaVision

Chapter 1, "Introduction to AmigaVision," introduces you to the AmigaVision software and describes how to get started, set up, and configure your system.

Chapter 2, "What is Multimedia," gives you a working definition of the term "multimedia" and why it is an important concept of the AmigaVision software. You will learn about the seperate data structures that make up a multimeda application: text, numeric data, graphics, animation, video, sound effects, speech, and music.

Chapter 3, "The Art of Authoring," demonstrates the key steps involved in program creation including: defining the program, outlining the program, and implementing the program.

Chapter 4, "Menus, Windows, Requesters, and Icons," takes a look at the development environment of AmigaVision and the different elements involved in it. In this chapter you will learn to access windows, requesters, and icons via pull-down menus.

Chapter 5, "Editing Your Program," completes the "basics" section by explaining how to edit your applications using such features as Move, Copy, and Search to manipulate icons within a program.

Section Two — Command Reference

Chapter 6, "The Control Command Menu," begins the "Command Reference" section with the Control command functions. You learn to determine the direction of program flow by jumping from one position in the program to another, or executing sections of code under conditional program control.

Chapter 7, "The Interrupt Command Menu," shows you how to use the Interrupt icons which allow you to interrupt a program to execute whole sections of code at a specified time.

Chapter 8, "The Database Command Menu," introduces the topic of databases. You learn to use the database command icons to manipulate already-existing database applications.

Chapter 9, "The Wait Command Menu," shows you how to achieve the highest degree of interactivity possible in AmigaVision using the five Wait command icons. This interactivity includes pausing program execution for a defined amount of time, until a certain defined condition occurs, or until a user presses a key on the keyboard or clicks with the mouse.

Chapter 10, "The Audio Visual Command Menu," shows you how to use the nine Audio Visual commands and incorporate all the elements of multimedia into your applications.

Chapter 11, "The System Command Menu," teaches you how to organize your applications using the seven System command icons including: the Module, Subroutine, Quit, Return, Execute, Timer, and Resource icons.

Section Three — Using AmigaVision Editing Tools and Programming Techniques

Chapter 12, "The Videodisc Controller," shows you how to incorporate full-motion video into your programs using the Video icon and Videodisc Controller.

Chapter 13, "The Expression Editor," teaches you how to create, define, and evaluate variables and expressions so you can manipulate data in your program.

Chapter 14, "The Database Editor," shows you how to use the database editor using the full complement of specialized commands for creating custom database applications.

Chapter 15, "The Object Editor," teaches you how to create your program's interface and its associated features — graphic objects, text information, and input fields.

Chapter 16, "Programming Basics," demonstrates how to develop serious applications with AmigaVision by familiarizing you with some of the basic programming concepts.

Section Four — Appendixes

Appendix A features the useful development tools and accessories available for AmigaVision programmers. Software packages include: graphics and animation, image processing, fonts, sound, music, text editors, and word processors. Hardware products include: video digitizers, audio digitizers, touch screens, and genlocks.

Appendix B provides all the current information and analysis of/on the recent upgrade v1.70 of the 1.53G AmigaVision release. Included in this upgraded version are the addition of important new features to the Execute icon as well as some new functions for use in the Expression editor. In addition to the new features, there are also some minor bugs in the earlier versions that have been corrected.

Conventions Used in This Book

Certain conventions are used in this book to help you learn AmigaVision. For each convention, an example is given to illustrate how each is used throughout the book.

🅰N (new Flow window):

> A solid box with an A inside of it is used to indicate that the Right Amiga Key is to be used. The Amiga key is located to the right of the space bar on the Amiga keyboard. When placed next to another letter, this means the Right Amiga Key can be pressed simultaneously with the corresponding letter (in this case, the letter N) to execute a command. Similarly, a solid letter A indicates the Left Amiga Key and can be used in the same way.

```
CD sys:
MAKEDIR AmigaVision:
```

All information that appears on screen is set off in a different font than the rest of the text. Listings of information that do not appear on screen are indented in the body text.

You can see that *x* is assigned a zero:

Any variable used as a place holder is printed in italics in the body text.

The Help requester appears when the Help button is clicked:

The names of all icons, requesters, menus, commands, buttons, gadgets, and windows are printed with the initial letter capitalized.

A wild card is indicated by an asterisk (*):

References to non-letter keys on the Amiga keyboard are spelled out and followed by their symbol in parentheses.

Another Title to Enhance Your Amiga Knowledge

Summary

When I first started using AmigaVision, I did not fully realize the potential of this icon-based multimedia application language. Don't be fooled by its ease of use — this is a serious programming language. AmigaVision has allowed me to create programs that would have taken much longer using a conventional language, and I am confident that AmigaVision will become an integral part of your programming library as it has mine.

I invite you to let me know what you think of this book and share any problems or triumphs you experience. Send me a letter with a self-addressed stamped envelope care of IDG Books, 155 Bovet Road, Suite 730, San Mateo, CA 94402. Now, let's get to it!

Section One
The Basics of AmigaVision

— 1 —

Introduction to AmigaVision

Until very recently personal computers were limited to processing text, simple graphics, and elementary sounds. Even the most sophisticated applications could manipulate only words and numbers, and the most advanced user interface available was limited to two colors: black and white. Then, in July 1985, a new type of computer appeared — a computer with high-speed color graphics, animation, video compatibility, sophisticated sound, synthesized speech, a multitasking operating system, and a modern, graphical user interface. This stunning new computer was the Commodore Amiga.

In the years that have followed the Amiga's introduction we have seen an incredible evolution at work in the computer industry. In the areas of graphics, animation, and video the Amiga reigns supreme. More software for these types of applications are available for the Amiga than for any other personal computer, and overall, this software is unsurpassed in performance and features. The computer itself and its operating system have also kept up with the times, so much so that the most advanced Amiga models can stand shoulder to shoulder with workstations costing many times more. Still, entry-level Amiga models offer significant value to the home and educational user.

Just as the computer's interface has continued to develop in terms of user friendliness, power, and performance, so too have the applications software advanced, giving us ever more features that are easier to use. As a result, more and more of us have been able to express ourselves creatively, by painting pictures, composing music, creating animations, and even manipulating video images.

Let's say you have just finished a wonderful 3-D animation. After watching it a few times, you begin to think about how much better it would be if you could just add a few sound effects and some background music. Perhaps you have created several animations, all meant to demonstrate some educational concept, but you need to be able to choose the appropriate animation for a given event. Of course, some on-screen documentation would help to clarify the finer points of each animation. To top it off, you might want to call up some portion of a laser disc to add to the effectiveness of the presentation.

Once you have begun thinking in terms of mixing graphics, audio, animation, and so on, you have crossed (perhaps unknowingly) into the world of multimedia. At this point, your progress begins to slow down, because actually combining these different information formats into a smooth, interactive presentation has traditionally been a complex process, and until recently only a very few media experts were capable of carrying it out.

Suppose that you planned to create such a presentation using one of the traditional computer languages like BASIC or C. Just what does that involve? Well, if you don't already know the language, you will have to learn how to program it effectively on the Amiga, which is no small task. Once you know the language, you will need to write some program code to display graphics. Next, you'll need an animation-player routine. Then you must write routines for handling sound effects, music, text, laser-disc segments, and user input. Finally, you need to write the main program to carry out the one specific presentation you have in mind.

Now, this isn't an impossible task, but even for an experienced programmer it is a serious, time-consuming project. And unless you write each subroutine with enough flexibility to accommodate every type of graphic, sound, or animation, the next time you want to create another multimedia application, you will be forced to spend even more time altering and debugging these routines. When you consider the effort required, it's no wonder so few people are willing and able to carry such tasks through to completion.

Since its inception, Commodore's Amiga has been in the forefront of the multimedia revolution. Although it has some of the best multimedia-creation tools available today, it has been obvious for some time that it takes more than good paint, sound, and animation programs to create multimedia applications. What is needed is a sophisticated multimedia-authoring system designed to bring together the various components of multimedia: something that is powerful yet easy enough to use that anyone can take advantage of it. With this criteria in mind, Commodore Business Machines and IMSATT Corporation developed not only one of the most advanced computer languages in the world, but also the most user-friendly. That language is AmigaVision.

AmigaVision

AmigaVision is an icon-based, object-oriented computer language. That's an impressive statement, but exactly how does AmigaVision differ from traditional computer languages? Well, until now programming has involved entering many lines of often cryptic commands into a text editor, and following each command with a variable number of obscure parameters. To do this, you must either memorize the parameter list or keep your language reference book nearby so you can look up commands to see what parameters are required. If you are using BASIC or some other interpreted language, your code can execute directly once you finish entering the program. If not, you must first compile and link your program with some supplied binary code. Regardless of the language involved, programming has traditionally been performed this way. AmigaVision, however, takes an entirely different approach.

Just as in more conventional languages, AmigaVision has specific commands that require a variety of information in order for the computer to process or execute them. AmigaVision's commands are not words, however; they are icons or symbols that pictorially represent the functions they perform. For example, the icon that represents speech synthesis is designed to look like a human mouth, while the symbol for the command that controls a laser-disc player looks like a laser disc (see Figure 1-1). This visual symbolism makes it easy for the beginner to immediately understand what action is being performed. These symbols help experienced programmers or software authors, too, by quickly communicating what the code is intended to do. This is helpful when you haven't looked at a program for a long time or when you are stuck trying to maintain or update a program written by someone else.

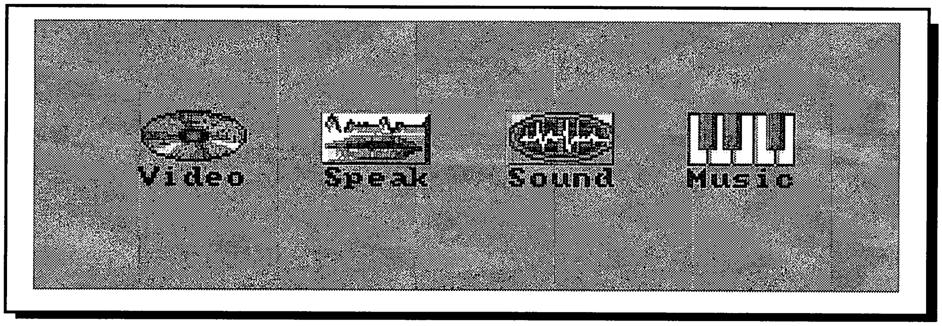


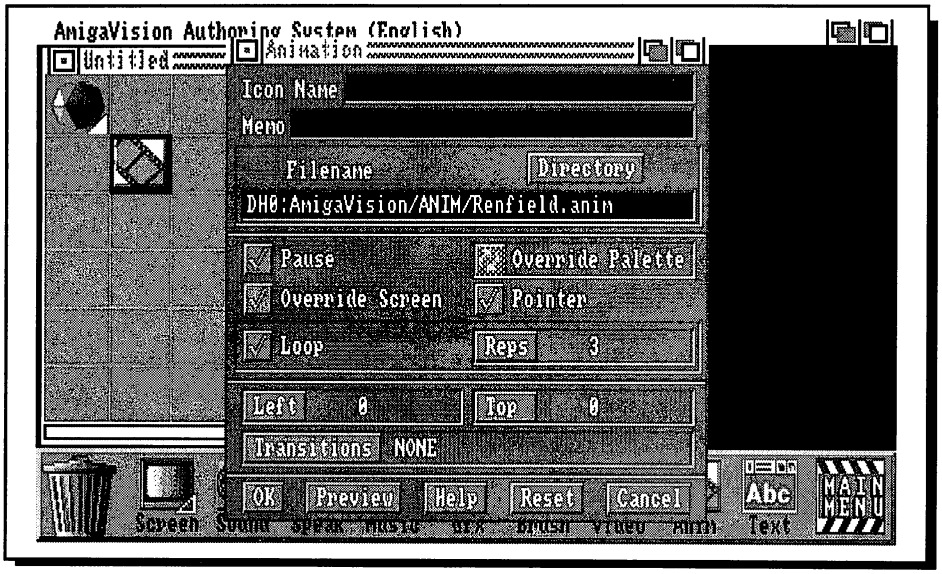
Figure 1-1
Sample AmigaVision Icon commands

Because the AmigaVision commands are graphical, it should come as no surprise that the program listing itself is also graphical. In fact, an AmigaVision program resembles nothing so much as the classical flowchart that every computer-science student is taught to design. The basic reason for teaching and encouraging program flowcharting is to help students determine exactly what they need to do before beginning the grueling task of writing the code. Once the logic is worked out the programming itself becomes easier.

The ability to visualize a problem does indeed make it easier to understand, and this concept is what makes programming with AmigaVision so easy. Regardless of your experience level, as you become more familiar with programming AmigaVision, you will find that its graphic interface not only makes it easier to bind together the elements of multimedia applications, but also facilitates the understanding of the logic behind your programs.

Like text-based languages, AmigaVision requires you to define parameters for the command icons before it can execute the commands. With AmigaVision, however, the parameter list is built into the icons themselves. All you — the programmer — need do is double-click on the icon, and a requester appears with buttons and gadgets that let you easily define the specifics of the given command (see Figure 1-2). This, then, is the object-oriented aspect of AmigaVision. Many of the commands, such as the Screen, Brush, Music, Anim, and even the Video icons, represent objects as well as actions. Simply stated, the icon represents an action or function, but when you define it via the requester, in many cases it also becomes a specific object — a data structure to be manipulated or used.

In the classic sense, a program is a sequence of instructions that the computer follows and executes in a linear manner. Generally, the machine carries out each instruction completely before executing the instruction immediately following it. This means that program flow moves from the top down, although branching to other sections of the program is allowed. AmigaVision flowcharts work in this manner, executing from top to bottom, but there are some differences. For example, some commands allow you to initiate their actions, then continue on to the next command without waiting for the first to finish its task. Such asynchronous command types are found in the AV (AudioVisual) menu, and examples of them are the Anim and Sound icons. The ability to have more than one command or process operating concurrently is extremely powerful, and gives the programmer more flexibility in the types of applications that can be created. This is especially important when combining different data elements into an interactive multimedia application.



Icon Name
Memo
Filename
DH0:AmigaVision/ANIM/Renfield.anim
✓ Pause
Override Palette
✓ Override Screen
✓ Pointer
✓ Loop
Reps 3
Left 0
Top 0
Transitions NONE

Figure 1-2
Requesters are used to define commands

Don't let AmigaVision's simplicity fool you into thinking it is merely a tool for creating slideshows. The AmigaVision Authoring System is a complete computer language, and has many features in common with other languages. It also has some significant differences and enhancements over other programming methods. Throughout this book we will look at both the similarities and the differences.

Getting Started

In order to effectively use AmigaVision, there are a number of hardware and software requirements you must meet. These requirements are covered in your *AmigaVision Users Guide,* but I want to expand on them here. According to the Guide, the minimum hardware requirements are:

- An Amiga computer with one floppy drive
- One megabyte of RAM

Commodore recommends, however, that to effectively author applications with AmigaVision your system should consist of:

- An Amiga computer with one floppy drive
- Three megabytes of RAM
- A 20MB hard drive

I also want to add some enhancements to those recommendations. If you are serious about developing software with AmigaVision, your system should consist of the following:

- An Amiga computer with two floppy drives
- Three or more megabytes of RAM
- A 40MB (or larger) hard drive
- A Super (one-meg) Agnus chip
- Versions 1.3 and 2.0 of the Amiga operating system

You will notice that in the last set of recommendations I mention Super Agnus but not Super Denise. That's because there are immediate benefits to having the full megabyte of chip RAM Super Agnus gives you, while Super Denise adds nothing to the current version of AmigaVision. When working with graphics and sound, as you will with AmigaVision, having as much chip RAM as possible is very desirable. Of course, if you want both chips (combined they are called the Enhanced Chip Set, or ECS), feel free to add them. A lot of other software does support the ECS directly, and having them is useful, especially if you want to run the 2.0 operating system. If you are not sure which versions of the chips your computer has, see your local dealer or service center. (Owners of the A3000 series need not worry about the ECS, as their computers come with the new chips, and in fact have a special version of the Agnus chip that gives them two megabytes of chip RAM.) You should also let the dealer handle chip installation; installing these chips is not something the average user should do!

I also recommend having at least a 40MB hard disk because graphics, sound, music, and animation eat up a lot of hard disk space very quickly. If you are working with an Amiga 500 equipped with the 20MB A590 hard disk, you may find it impractical to replace the drive. Just keep in mind that you will want to have room on the drive for more than the files used by AmigaVision. Word processors, paint programs, animation packages, databases, and spreadsheets need disk space too.

If you are still using version 1.1 or 1.2 of the Amiga's operating system, you should know that AmigaVision will not work with these older operating systems.

To use AmigaVision, you must upgrade to at least the 1.3 version, and I highly recommend getting the new 2.0 version as well. That way you can make sure your application works under both 1.3 and 2.0 (better safe than sorry!). If you have been using one of the older operating systems, see your dealer about upgrading with one of the Enhancer Packages from Commodore Amiga.

Depending on the applications you plan to create with AmigaVision there are a number of accessory hardware devices you should consider acquiring. These include:

- A genlock device
- A laser-disc player
- A touch-screen monitor

If you are interested in adding any of these to your Amiga system, please see the appendices for specific hardware suggestions that are known to be compatible with AmigaVision.

Setup

The procedure for installing AmigaVision onto a hard disk is well documented in the *AmigaVision Users Guide*, but I want to offer a couple of hints that will make authoring easier. First, I suggest you install AmigaVision into a hard-disk directory called AmigaVision. By having it in its own directory (instead of the root, or main, directory of the disk) you can create special directories for storing the components of an AmigaVision application without cluttering the main directory of the disk. The first thing to do is create a drawer called AmigaVision on the disk on which you want to install AmigaVision. This can be done either by copying the EMPTY drawer and renaming it to AmigaVision using Workbench, or from the CLI with the MAKEDIR command (which, in this case, is easier). Here is an example of how you might proceed from the CLI or SHELL if your main system disk is called DH0:.

```
CD sys:
MAKEDIR AmigaVision
CD AmigaVision
MAKEDIR applications
MAKEDIR ilbm
MAKEDIR anim
MAKEDIR smus
MAKEDIR 8svx
MAKEDIR dbms
MAKEDIR text
CD smus
MAKEDIR scores
MAKEDIR instruments
CD /
CD ilbm
MAKEDIR brush
CD /
MAKEDIR av_text
CD sys:devs
MAKEDIR players
CD sys:
COPY empty.info to ram:
RENAME ram:empty.info ram:AmigaVision.info
COPY ram:AmigaVision.info DH0:
DELETE ram:AmigaVision.info
```

These commands create a highly organized AmigaVision directory structure, and prepare a place in your Sys:Devs directory to put the laser-disc device drivers. With the above directory structure, your AmigaVision directory contains specific places for picture (ILBM), animation (ANIM), sound (8SVX), music (SMUS), database (DBMS), and text files (TEXT). In addition, the SMUS directory has subdirectories for SMUS scores and for the instruments needed for the scores, and the ILBM directory has a subdirectory for brushes. Finally, you have a dedicated directory for the applications and courses you will be writing.

Once this is done you should copy all the files from your AmigaVision disks into the appropriate directories using your favorite file utility. (I prefer DiskMaster from Progressive Peripherals & Software, but other good utilities are available both commercially and in the public domain.)

Don't forget to copy the AV_Help files to the DH0:AmigaVision/AV_Help directory and the laser-disc drivers found in the Devs/Players directory on your AmigaVision boot floppy disk over to your new Devs/Players directory. You will also need to copy the Player.Device and player-units files from the Devs: directory on the AmigaVision boot disk into your system Devs: directory.

Next, add a couple of path assignments to your Startup-sequence in the S: directory of your boot disk. (We will assume for demonstration purposes that your boot disk is called DH0:.)

```
ASSIGN AmigaVision: DH0:AmigaVision
ASSIGN instruments: AmigaVision:smus/instruments
```

By assigning AmigaVision as a device, you will be able to run AmigaVision applications from anywhere in your system. Just use AmigaVision: as the main pathname for your data files. The final assignment is necessary because some SMUS scores insist on looking for a volume called Instruments for the instruments they need.

Configuring AmigaVision

AmigaVision offers a number of start-up parameters that, if properly defined, can make better use of your hardware resources. These environmental parameters affect how much memory the AmigaVision editor uses, which, on systems with limited memory, have a direct effect on the complexity of the application you can run. The five environmental variables, or flags, are explained below.

Lace

Lace is a shortened name for interlaced mode. AmigaVision uses an eight-color, 640-by-200 (48K) medium-resolution display screen for its editor. If the Lace option is enabled the screen display changes to an eight-color 640-by-400 (96K) interlaced display. This mode has three immediate drawbacks. First, it uses twice as much memory as the default display. If your system is short of memory, this display is probably not something you want to use. Secondly, the icons appear half as tall and somewhat flattened in this mode. Finally, unless you have a multiscanning monitor and either an A3000 or a flicker-reduction card such as MicroWay's flickerFixer or Commodore's 2032 (both of which are currently available for A2000-series machines), the display will exhibit the trademark screen flickering associated with interlaced mode.

The positive side to the Lace mode is that it gives you twice as many lines of icons on the display as normal, which is very useful when you need to see more of the overall logic flow of your program.

I use Lace mode a great deal, even though I do not have a flickerFixer and multiscanning monitor. Being able to see more of the program is a real advantage for me, and AmigaVision's gray-scale interface minimizes the effects of the flickering.

Back

Back, short for Backup, instructs AmigaVision to automatically make a copy of each application file you save from the AmigaVision Editor, and to add the suffix

".bak" to its file name just before the suffix AmigaVision automatically adds to all the files it generates (.AVf). If, for example, you save a file called Slideshow while the automatic-backup feature is enabled, you will find two files on the disk, one called Slideshow.AVf and another called Slideshow.bak.AVf.

The purpose of this option is to keep a copy of the application version that you had before you saved it last. The importance of a backup becomes very clear when you accidentally delete a file, the file becomes corrupted, or if the changes you made to it last turn out to be all wrong. The .bak file is your insurance policy!

As with any insurance policy, there is a price to be paid. In this case you pay with disk space. The automatic backup doubles the amount of disk storage used, which can quickly become critical if you are working from floppy disks. Hard-disk users are not immune to these effects either, especially when the application is quite large. The choice is yours, but in general, enabling the automatic-backup option is a good idea.

Dbuf

Dbuf stands for Double Buffering. If your application includes animation or scrolls text in a window, you probably want to enable double buffering — a display tool that keeps two copies of your screen in memory in two separate buffers. While you look at one frame on screen, the computer prepares the succeeding one, drawing the next animation image or writing text, from the other buffer. With this option enabled, animation and text scrolling look very smooth because the viewer never sees the rendering being performed, only the result. Without double buffering, animation can appear to be jumpy or to contain color flashes. This effect occurs because even the Amiga's custom chips cannot always update the screen faster than your eye can detect the change.

Deciding when and if to use double buffering is easy. If you have enough memory, use it. If you don't have enough then you won't be able to do it anyway. In that case you should start saving your pennies for more RAM, or think of ways to alter your application so that it doesn't need such large animations.

LMEM

LMEM stands for Low Memory. Normally, when you present your application, the AmigaVision flow editor remains in memory. Enabling the LMEM flag will remove the flow editor from memory when your presentation begins and free up valuable RAM resources for your application. For systems with minimal memory it is highly advantageous.

If you have plenty of memory, however, it is best not to use the LMEM. That's because when the system finishes presenting the application, AmigaVision must reload the flow editor from disk. Programmers with fast hard disks may not notice much delay, but those working on floppy-based systems will quickly find it a nuisance. Again, the solution is more RAM!

Edit

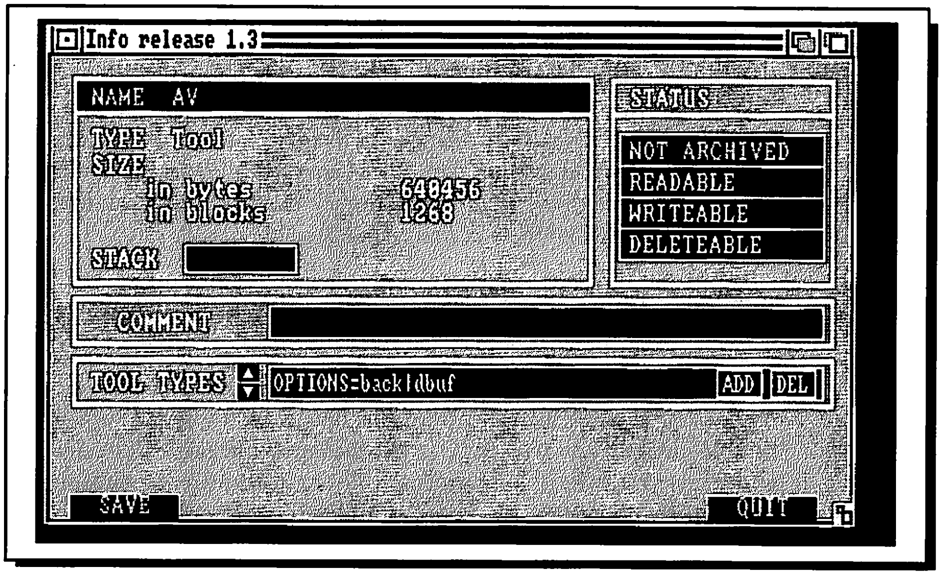
The Edit flag determines what happens when you double-click on an AmigaVision .AVf file from the Workbench. Without this option, once you double-click on an icon, the corresponding application is presented immediately, and when you exit it, you automatically return to the Workbench. With the Edit flag enabled, however, when you double-click on an applications icon, the program loads into the flow editor and does not execute. In general, it is a good idea to have this flag enabled unless the file is to be merely an executable file, as a finished application is when ready for distribution.

Opting for Options

There are two ways of setting these environmental flags. The first method involves setting AmigaVision's Tooltypes in the AV.info file via Workbench. To do this, click once on the AV icon to highlight it. If you are using Workbench 1.3, your next step is to select Info from the Workbench menu at the top left of the screen using the mouse. If, on the other hand, you are using Workbench 2.0, you should select Information from the Icon menu. Either action opens an information window that describes the AV file and contains, in the lower section, a field called Tooltypes. To set AmigaVision's configuration the way you want, simply enter "Options=" followed by the names of the flags you wish, in the Tooltypes field. (See Figures 1-3 and 1-4).

By entering a flag in this field you enable it (turn it on), and by leaving it out, you disable it. You can enable multiple flags simply by separating the commands with the vertical bar (|). To enable interlaced mode and automatic backups, for example, enter the following:

```
Options=lace|back
```



OPTIONS=back|dbuf

Figure 1-3
Configuring AmigaVision using the Workbench 1.3 Info requester

To use double buffering and automatic backups, enter just those two options:

```
Options=dbuf|back
```

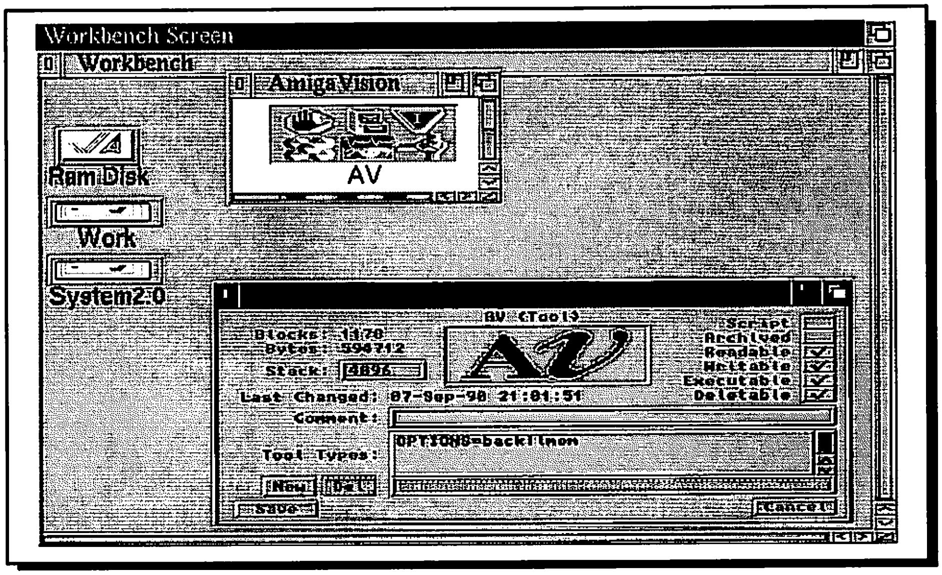
And to enable everything, type in the commands for all five flags:

```
Options=edit|lace|back|dbuf|lmem
```

Once you have entered the options you want, use the mouse to click on the Save gadget. From now on these are your defaults. If you want to change them later, just repeat the above process, adding or deleting flags as you wish.

Setting these parameters from the Shell or CLI is just about as easy. The only difference is that you must enter them each time you start AmigaVision. Here, the environmental flags are represented by the following single-letter commands:

b	=	Back
d	=	Dbuf
i	=	Lace
m	=	Lmem
a	=	Load and present the application.



Blocks: 1170
Bytes: 594712
Stack: 4096
Last Changed: 07-Sep-90 21:01:51
Comment:
Tool Types: OPTIONS=backlinon

Script [ ]
Archived [ ]
Readable [✓]
Writable [✓]
Executable [✓]
Deletable [✓]

Figure 1-4
Configuring AmigaVision using the Workbench 2.0 Information requester

With one exception, entering these commands in the CLI or Shell sets the corresponding option. The exception is the "a" command. Although you would expect it to enable the edit option, in fact it does just the opposite, telling AmigaVision to load and present the application. The "a" command is the equivalent of having the Edit flag *disabled.* This option is different than the others in one other respect, too: while you must leave spaces between the other commands and the file names when you enter them in the CLI, no spaces are allowed between the "a" and the file name.

Here are a couple of examples for starting AmigaVision from the Shell or CLI, with and without flags. Both examples assume you are in the AmigaVision Applications directory and have your applications in that directory.

`AV aStates.AVf`	Load and present the course States.AVf
`AV d m aStates.AVf`	Load and present States.AVf using Dbuf and Lmem options.

– 2 –

What is Multimedia?

Multimedia is the most prominent computer buzzword of today. IBM, Apple, and Commodore are all busy promoting it in their television and newspaper ads, while industry pundits argue the merits of the various platforms. With all this discussion of multimedia, it is interesting to note that there is no single definition of multimedia that is accepted industry-wide. In fact, if you were to ask a dozen individuals who use multimedia technology in their businesses to define the term, odds are that you would get 12 very different answers.

Because AmigaVision is defined as a multimedia authoring language, and because AmigaVision is the subject of this book, the lack of an accepted definition of the term multimedia would seem to pose a problem. As we will be encountering the term in various ways throughout the book, it is probably wise to establish not only a working definition of the term multimedia, but also to discuss the various components of multimedia that AmigaVision allows.

In my opinion, of the various computer companies that are developing multimedia platforms, Commodore has come the closest to accurately defining a multimedia system. Here is the 'official' definition of multimedia from Commodore:

> "(Multimedia is) a method of designing and integrating computer technologies on a single platform that enables the end user to input, create, manipulate, and output text, graphics, audio, and video utilizing a single-user interface."

Not surprisingly, this definition is very closely associated with the Amiga. The simple truth is that Amiga is the only personal computer that can perform these tasks without forcing its users to buy a wide array of expensive add-on devices.

While the term multimedia is fairly new, the Amiga has made extensive use of the various components of multimedia since its release. Besides its advanced graphics and sound abilities, the Amiga was the first personal computer to offer serious animation features in hardware and software by way of its blitter objects (BOBs). It was also the first to provide reusable hardware sprites, which are still not all that common in computers of any size. Manufacturers have found it easier to develop video devices such as genlocks and frame grabbers for the Amiga than for other machines because the Amiga's internal hardware clocks are timed to closely match the NTSC signals used by television. Most importantly, the Amiga's operating system was designed to allow multiple tasks or programs to run concurrently, a critical need for effective multimedia applications.

For the Amiga, the concepts behind multimedia were already well established under the earlier banner of "desktop video." That term presents problems, however, because it focuses attention solely on video, and while video is an important consideration, it is only one of many forms of data you can incorporate into your applications.

Because the term desktop video is so narrow in scope, multimedia is a far better term for describing applications that combine multiple forms of information. It's better also because most people get a more accurate idea of what is meant when a program is described as a multimedia application rather than a desktop-video application.

Joseph's Coat of Many Colors

To gather information from the complex environment we live in, we rely on our five senses, the most important of which are sight and sound. Although computers may some day be able to store, process, and recall information relating to touch, smell, or taste, for now our eyes and ears are the only vehicles through which computers can present information to us.

While computer multimedia applications convey information via only the two main senses, today's technology allows the presentation of sight and sound information in many ways. When you combine several of these data-relay forms into a computer application, you are creating multimedia applications.

For the AmigaVision programmer, the separate data structures that can make up a multimedia application are text, numeric data, graphics, animation, video, sound effects, speech, and music.

Text

The simplest method that computers have of conveying information is through written words. All computers can manipulate text and present it on the screen in a variety of ways. The Amiga, however, can handle any number of fonts in a wide variety of styles and colors, and AmigaVision gives you great control over the text you display in your applications.

Additionally, AmigaVision contains a database-file manager, compatible with the popular MS-DOS program dBASE (by Ashton-Tate), which you can use to systematically store and retrieve information in the form of text strings, numbers, calendar dates, and Boolean (logical) variables.

Numeric Data

The AmigaVision authoring system is a full-featured computer language. As such it includes, in addition to standard text-manipulation capabilities, an impressive array of numeric functions and support for both integer and floating-point mathematics. These features allow additional processing of information that you can then present to the user in more visually exciting or even auditory ways.

Pictures

AmigaVision fully supports the Amiga's IFF-ILBM graphics standard. This means that your AmigaVision applications can use and display images created in virtually any Amiga graphics program. AmigaVision supports all resolutions and modes, including the 64-color Extra_Halfbrite and 4096-color HAM (Hold And Modify) modes. The software also allows full video overscan, so you can use images up to 736-by-480 pixels. Besides full-screen images, smaller IFF graphics images (such as Electronic Art's DeluxePaint brushes) are supported.

Animation

If one picture is worth a thousand words, an animated image is worth several times more. AmigaVision supports the standard IFF animation file format (ANIM5) used by most Amiga animation packages, including DeluxePaint III

(Electronic Arts), Photon Paint 2.0 (MicroIllusions), Sculpt Animate 4D (Byte by Byte), Turbo Silver (Impulse), and a host of others.

Video

Perhaps nothing puts the stamp of multimedia on an application better than the inclusion of video sequences. AmigaVision has superb support for a variety of industrial laser-disc players, and in combination with a genlock device, the software gives you unparalleled power in mixing computer information with full-motion video sequences on the Amiga's standard RGB screen. In addition, the laser-disc player control extends to the stereo audio tracks on the laser disc. You can selectively enable or disable each channel to retrieve music, narrative, or both directly from a laser disc.

Digitized Sound

AmigaVision allows the inclusion of digitized sounds — both mono and stereo — in the IFF-8SVX file format. You can digitize 8SVX-format sound effects, voices, and music using any of several sound-sampler devices, or purchase them on disk as "clip sounds," in much the same manner as desktop publishers purchase clip art. In addition, a large number of digital sounds are available in the public domain. You can find these in users'-group libraries, commercially available PD (public-domain) software collections, or on electronic networks and bulletin-board systems (BBSs).

Speech

Because the Amiga's standard equipment includes speech-synthesis capabilities, AmigaVision allows the inclusion of synthetic speech into its applications. While Amiga-generated speech certainly is not as good as a digitized voice, it is nevertheless a useful option in many situations. The Amigas' speech narrator is very easy to use, and has the ability to directly convert text into speech without any preprocessing. The narrator can handle anything from a single letter, word, or phrase to an entire text file.

Music

AmigaVision can play IFF-SMUS-format music files, which you can use in conjunction with digitized instrument files to produce very complex and beautiful music scores. It should be noted that AmigaVision will play only those SMUS files (and instruments) in the format generated by Electronic Art's Deluxe Music Construction Set (DMCS).

User Interaction: A Multimedia Plus

There is yet another element you can add to your multimedia application to greatly enhance both its substance and usefulness. That element is user interaction. AmigaVision allows the user to interact with your program using the mouse, the keyboard, or even a touch-screen monitor.

By allowing the user to participate and even control the flow and execution of the program, you bring an extra dimension to your application. Instead of presenting information in a linear manner to a passive user, interactivity in a multimedia application puts the user in the driver's seat. Thus, the user can decide what to see and hear and in what order, just as he or she does in the real world. As the author of the application, an important part of your job is to see that any paths the user can choose have a purpose. Unlimited flexibility has its place, but in most cases it is not productive to let the user wander aimlessly through mountains of information — regardless of how interesting the data is.

A properly designed interactive multimedia application, then, allows the user to make decisions on what areas to pursue, yet keeps the options relevant to the situation. This is not to say that a good application could not offer, as an option, unlimited browsing through the data. If you want the end user to ultimately accomplish some task or learn a specific lesson, however, a controlled environment should also be available.

In the strictest sense, a multimedia application could be defined as a program that seamlessly integrates two or more of these key data types into a coherent application. According to this definition, however, the simple integration of text and numeric data is multimedia, and that is surely not what we mean when we use the term. What if we mix text and graphics, and throw in some simple sound effects... is that multimedia? Perhaps in the most elementary sense it is, but in that case just about all computers are already multimedia.

In order to be considered multimedia by today's standards, an application should contain *a substantial mixture of different data types.* It does not have to include full-motion video, although one that does can easily claim to be a real multimedia application. A program that contains text along with animation, sound, music, or speech can justly claim to be a multimedia application.

With AmigaVision, you can create programs, courses, and presentations in true multimedia style or using just one or two data-presentation methods. Your programs can be highly interactive or consist solely of simple presentations or slideshows.

Now that we have a working definition of multimedia and have identified the components of multimedia applications, it is time to begin learning how to use AmigaVision to create multimedia programs yourself.

– 3 –
The Art of Authoring

If you were to give 50 different programmers the same programming assignment but left the implementation of the program up to them, chances are very good that you would end up with 50 different approaches to solving the same problem.

The reason for this is that programming is much more an intuitive art than an exact science. A programmer builds software, crafting each routine in the manner learned through his or her experience. It is this personal knowledge and insight into the problem at hand that makes people better at programming than even the most sophisticated machines. If you doubt that statement, consider the following.

If an expert programmer writes an application using some high-level language such as C, Pascal, Fortran, or BASIC, and runs it through a compiler, the result will likely be more compact code that is capable of processing faster than the original code executed as an interpreted language. (A compiler is a utility that converts high-level source code to some form of machine language.) As anyone who has used a compiler — especially a BASIC compiler — can tell you, the increase in speed is usually quite dramatic.

If, however, the same programmer writes the same program in assembly (machine) language, the result will always be a more efficient, more compact, and usually a much faster program than the machine-generated compiled version. The reason is that even the best optimizing compilers running on today's most powerful computers are not as creative as their human counterparts, even at such computer-related tasks as programming.

Even though programming is not a science, there are specific steps involved in the creation of any program. If you follow these basic procedures, you will find your time at the keyboard to be more productive and satisfying.

AmigaVision Authoring

Regardless of whether AmigaVision is your first programming language or whether you are an experienced programmer, you will discover that the key to creating an effective AmigaVision application is to follow three principal steps: define the application, outline it, and implement it.

Defining the Program

Before you begin a programming task, you must first thoroughly understand what it is you intend to create. You need to define, fully and completely, just what you wish your program to do. The more forethought you give to the problem at hand, the easier it will be to design the program, the better the design will be, and the faster and easier it will be to actually create the program.

For example, suppose you wanted to create the old standby, the computerized checkbook. Using either pen and paper or a text editor (after all, you do own a computer), write down the various things you want your computer checkbook program to do.

```
AmigaVision Checkbook Functions
Enter check data
    Amount
    Date
    Who
    Check number
    Sales tax included?
    Purpose
Enter deposits
    Amount
    Date
    Source
Save data to check database
Recall data from check database
Balance checkbook
    Balance = balance - checks
    Don't forget to subtract any checking account fees!
    Add any interest earned
Balance inquiry
    Current balance
    On any date
```

```
Search By
   Check number
   Amount
   Date
   Person or business
Special Features
   Graphic interface (use a digitized check as an input form?)
   Formatted printed report by range of check numbers, dates or
   amount (to screen or printer)
   Calculate sales tax over a given time period
```

As you can see, by writing out what the program is to do, we have managed to define exactly what we want to accomplish. This list gives you, as a software author, a strong guide to the needs of the program. Again, the first rule is to *define the program in as much detail as possible before you start writing it!*

Outlining the Program

Once you know exactly what it is you want to accomplish, you need to determine how the program should go about doing it. I have always found that it helps a great deal to outline how the program will work using either a graphic flow chart or a simple text outline. Here is a simplified outline of our checkbook.

```
AmigaVision Checkbook Program Outline
Display main menu
User selects option
   Add checks
   Add deposit
   Make inquiry
If response is
   Add checks - call check-entry subroutine
Else if response is
   Add deposit - call deposit-entry subroutine
Else if response is
   Make inquiry - call report-data subroutine
Goto main menu
Check entry subroutine
   Display check data input screen
   Have user input check information
   Calculate new balance
   Store new data in database
   User selects option
      Finished entering checks
      Enter another check
```

```
        If response is
           Enter another check - goto check-entry subroutine
        Else If Response is
           Finished entering checks - return from subroutine
     Deposit-entry subroutine
        Display deposit-data input screen
        Have user input deposit information
        Calculate new deposit
        Store new data in database
        User selects option
           Finished entering deposits
           Enter another deposit
        If response is
           Finished entering deposits - return from subroutine
        Else if response is
           Enter another deposit - goto deposit-entry subroutine
     Report-data subroutine
        Display report-options screen
        User selects option
           Balance inquiry
           Deposit inquiry
           Check inquiry
           Finished data inquiry
        If response is
           Balance inquiry - call display-balance subroutine
        Else if response is
           Deposit inquiry - call display-deposit subroutine
        Else if response is
           Check inquiry - call check-search subroutine
        Else if response is
           Finished data inquiry - return from subroutine
```

This outline does not include all the routines that our report-generation option requires, but it should give you a good idea of how to outline an application. With practice you will find that you can quickly work out the overall outline and program flow, and the end result is much like the result of planning your summer-vacation route well in advance — you will get to where you want to go faster and with less problems, yet will be free to make detours and changes if you find it necessary or desirable.

Implementing the Program

If you followed the previous two procedures, you know how to define the program you want to create and then, using that definition, to outline the logic flow necessary for the program to work. Now all that's left to do is to implement the program, which means using AmigaVision and its command icons to create the checkbook application. We won't actually do that here because we are still at the beginning of the book and have a lot to learn about AmigaVision.

While we have not yet done any programming in AmigaVision, we have already learned some important lessons in how to program. Perhaps you can use the above definition and outline as the basis for a program of your own. One mistake beginning programmers often make is to omit those first two critical steps and jump right in at the implementation or programming stage. The result is usually poorly designed programs that take an unnecessarily long time to create and are often full of errors (also called program bugs). If you get into the habit of carefully defining and planning your programs you will find your workload is much lighter; that even complex programs are easier to implement. Perhaps most importantly, however, you will find the task of authoring software to be more enjoyable and rewarding.

Besides the actual programming involved in creating your application, the implementation process involves generating the text, images, sounds, animations, and even video that will make up your presentation. Deciding when to create or acquire these components depends on a number of things. For example, if there are aspects of the program you may not be able to adequately foresee in your definition and outline, it is probably not a good idea to create all the screens ahead of time. Using AmigaVision's Object editor, you can prototype your displays using simple geometric shapes instead of fancy graphics images. Then, after you implement the program code, you can go back and draw, paint, or otherwise generate the final images. On the other hand, if parts of your program depend on the existence of particular components, it may be necessary to have all or some of the elements on hand before you begin writing the program. You will have to make that decision based on the program's requirements.

Now that you are primed to become an AmigaVision program author, it is time to begin in earnest our study of the AmigaVision system.

– 4 –

Menus, Windows, Requesters, and Icons

Programming in AmigaVision involves working with a number of different elements. Besides the icon commands, there are windows, menus, and a large variety of requesters that you need to be familiar with. In this chapter we will take a good long look at these environmental aspects of programming in AmigaVision. You will find that the better you understand the developmental environment, the more productive your program-creation sessions will be. With that in mind, let's begin our study of AmigaVision by taking a serious look at its pull-down menus, windows, requesters, icons, and icon relationships.

To start AmigaVision from the Workbench, just position the mouse over its icon and quickly click the left mouse button twice. After a few seconds, you will be presented with the main AmigaVision menu and an empty Flow window (see Figure 4-1). If you wish to start from the CLI or Shell, use the AmigaDOS CD command to change to the AmigaVision directory and type AV.

The Flow window is where you will begin writing or authoring your AmigaVision applications. Directly below it is the Main Icon menu, a line of icons that graphically represent groups of functions or commands that AmigaVision offers. With one exception, when you click on any icon in this row, a new set of icons appears. As you may have guessed, that one exception is the Trashcan icon. The Trashcan icon appears in every icon menu and is used to discard icons you no longer need or want.

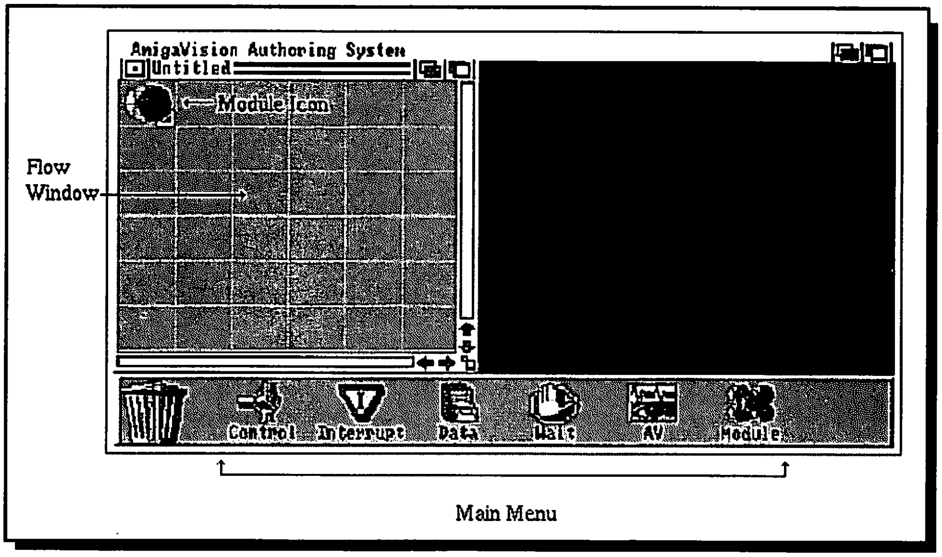


Figure 4-1
The initial AmigaVision editing display, with Flow window

Pull-down Menus

Across the very top of the AmigaVision display the title "AmigaVision Authoring System" appears. In version 1.53G and up, the language your AmigaVision software was written in appears next to the Authoring System title (not pictured above). Whenever you hold down your right mouse button, this title bar changes to reveal the names of the various menus that are available: Project, Edit, Tools, and Configuration (see Figure 4-2).

When you move your pointer to the top of the screen and position it over one of these menu headings, a menu pops down to display a number of options. These pull-down menus let you access a variety of tools, including applications-editing tools, the Object editor, the Database editor, and the Videodisc controller. In addition to using the mouse and menus to make selections, you can access many of these tools from the keyboard, an option you will find very convenient once you become proficient with AmigaVision. All available keyboard shortcuts are referenced throughout this book following the menu name at the beginning of each section. Unless otherwise noted, I will use 🅰 to refer to the Right Amiga key — the key located just to the right of the space bar at the bottom of the keyboard. (This key sports a capital 🅰 in an outline font, as opposed to the key to the left of the space bar, which is labeled with a solid capital A). To use these keyboard shortcuts you must hold down the 🅰 key and press the indicated key.

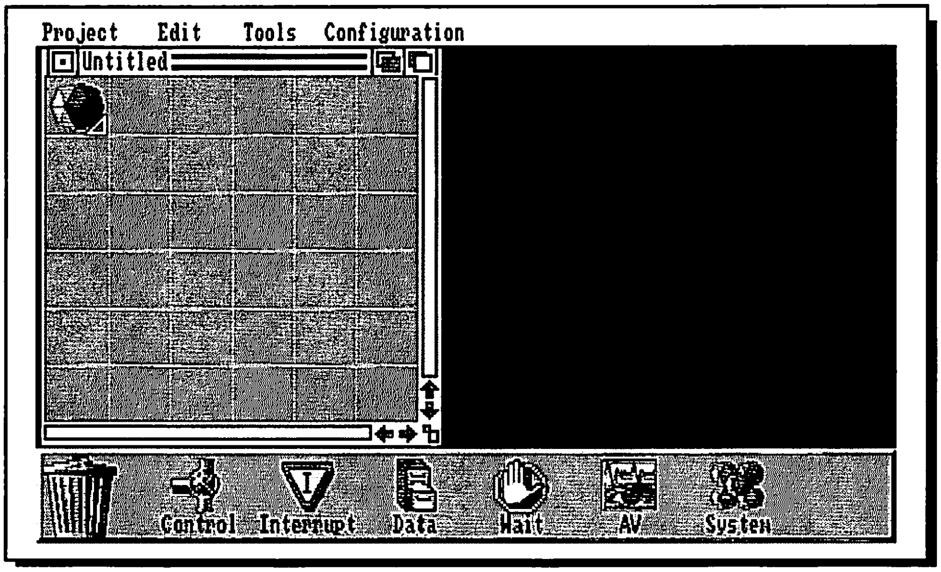


Figure 4-2
The title bar changes to reveal the names of various menus

Project Menu

Through the Project menu, you can load and save applications to and from the Flow window, create new or additional Flow or Content windows, present your application, and define your AmigaVision default configuration (Figure 4-3).

New

The first option on the Project menu is New. When you move your pointer over the word New, a submenu appears to the right of the Project menu (Figure 4-4). This menu allows you to select either a new Flow window (N) or a new Content window (Y).

You can have multiple Flow and Content windows open simultaneously; the total number is limited only by the amount of RAM available in your system. (Chip RAM is just as important as expansion RAM. You can have multiple megabytes of free memory in your system, but quickly run out of chip RAM when opening multiple windows.) Each new window appears over and just below the top window. In a non-interlaced display you can have 11 windows visible, overlapping, and stacked together like a deck of cards.

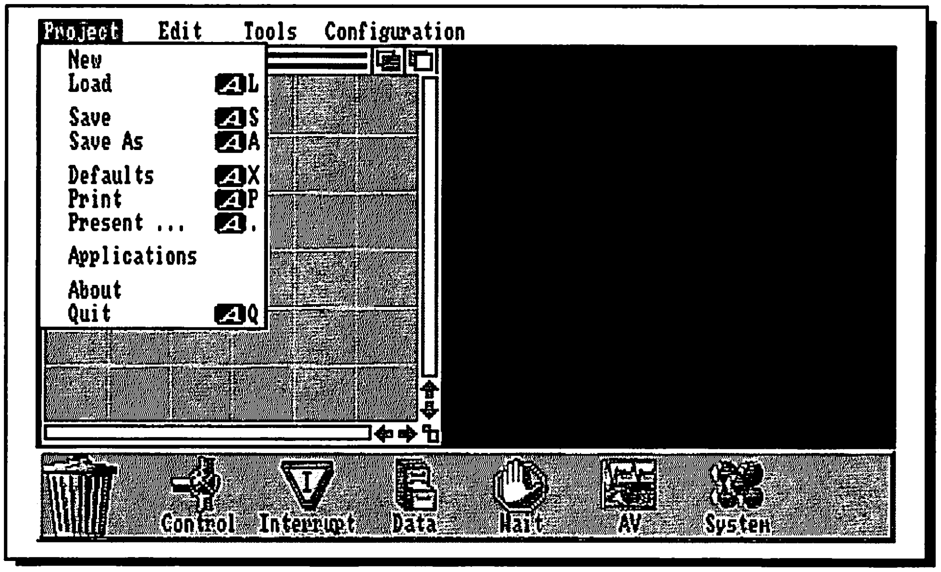


Figure 4-3
What you get when you pull down the Project menu

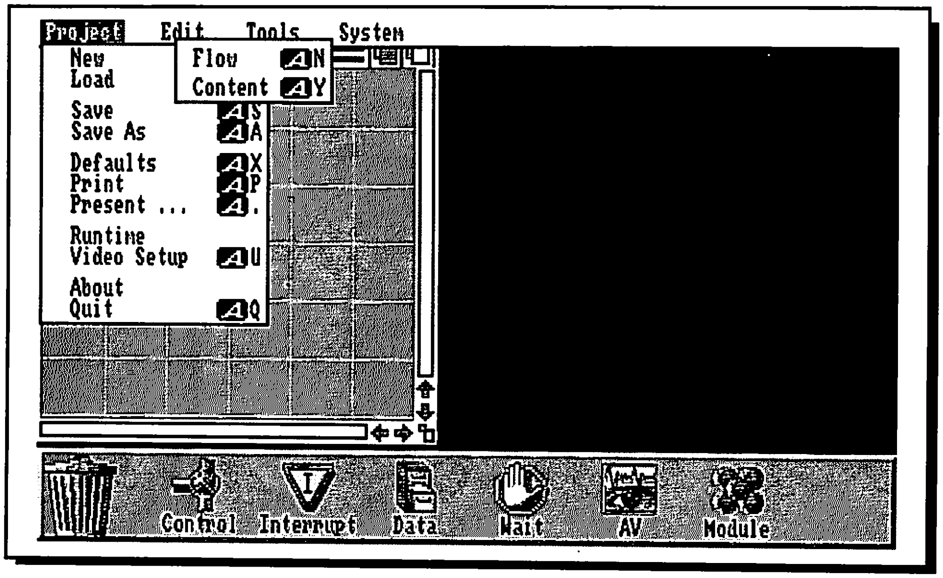
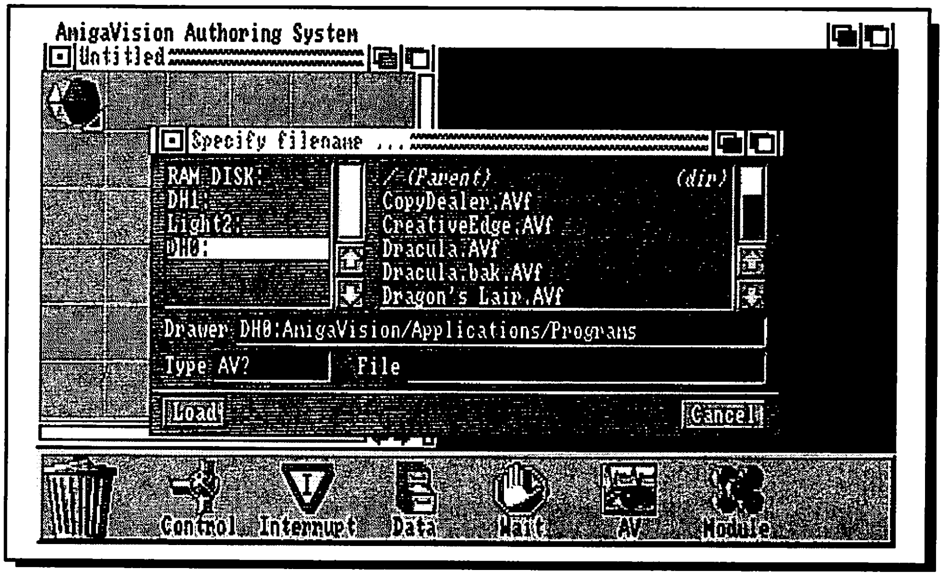


Figure 4-4
The New submenu lets you create a new Flow window or a new Content window

Load

The next option in the Project menu is Load (ⒶL). The Load option allows you to load existing AmigaVision applications from disk into the AmigaVision editor, where you can present or alter them.

Selecting Load brings up a special kind of menu called a file requester (Figure 4-5). In this case the file requester allows you to select the specific AmigaVision program you want to load into memory.



Drawer DH0:AmigaVision/Applications/Programs

Type AV?

File

Figure 4-5

The Load File requester lets you select an AmigaVision program to load

The Load File requester consists of two windows, three string gadgets, and two buttons. The window on the left displays the available disk devices — floppy disks, hard disks, and RAM disks. The window on the right shows the files and subdirectories found in the "current" directory, that is, the directory you have selected (only one directory can be current at a time). The first string gadget, labeled Drawer, shows your current drive and the directory path (the current directory and any parent directories it resides within). Below it and to the left is the Type text-string gadget, which specifies the type of file you are selecting. This field allows you to specify a group of characters that the program can use to find other files in the directory containing the same string. When you define a string in the

Type gadget, only files containing a trailing string of characters are displayed. This makes it easier to find the file you want, especially when there are many files. This field is usually context-sensitive, meaning it often appears properly defined for the type of file you are looking for. For example, in the case of Figure 4-5, the file requester appeared because you selected the Load option, and the suffix is .AV?. If you were to select a database, the field would contain the suffix .DBF. If you do not specify a string in the Type field, the program lists every file in the current directory.

To the right of the Type gadget is the File gadget. This contains the name of the file to be loaded. To indicate the file you want to load, just position the mouse pointer over that file's name and click once with the left mouse button. The file name will appear in the File gadget. You can also enter the file name from the keyboard by clicking once inside the File gadget box (which makes a cursor appear in the box), and typing it in.

If you want to access a drive other than the one that is currently selected, go to the left window where the available devices are shown, point to the gadget corresponding to the device you want to read, and click the left mouse button. After a brief delay while the program reads the directory into memory, the right window will list all the files and drawers in the root (main) directory of the device you selected. To enter a subdirectory (drawer), just click once on its name, and the right window will be updated with the contents of that directory. To step back up to the directory that contains the drawer you are in, simply click on the word Parent found at the top of the directory list.

Both the Device and Directory windows are equipped with scroll gadgets, which appear as up and down arrows on the right side of the window. If the list of names is longer than the window can accommodate, you can scroll up or down through the list by clicking on these arrows.

Finally, the Load requester contains two buttons labeled Load and Cancel. Clicking on Cancel aborts the load process and returns you to whatever you were doing before. Clicking on Load causes the program to read the specified file into the computer's memory. You can also load a file from the window on the right by merely double-clicking (clicking the left mouse button twice in rapid succession) on the name of the file you wish to load.

The Load requester is the standard file requester that is used throughout AmigaVision for loading and defining the different elements you use in your applications. You can access it from the editors and many of the command icons.

Save/Save As

The two save options — Save and Save As — let you store your AmigaVision application on disk. Save As (ⒶA) brings up a file requester that is identical to the Load requester, except that in place of a Load button there is a Save button. You can enter a file name into the File string gadget by clicking on the gadget with the mouse and then typing in the name from the keyboard. Or, you can click on a file name in the right window, and that name will appear in the File string gadget.

When saving your file, the Type field defaults to the suffix .AVf (AmigaVision flow). Unless you change the contents of the Type field, whatever name you give your application will have .AVf added to it.

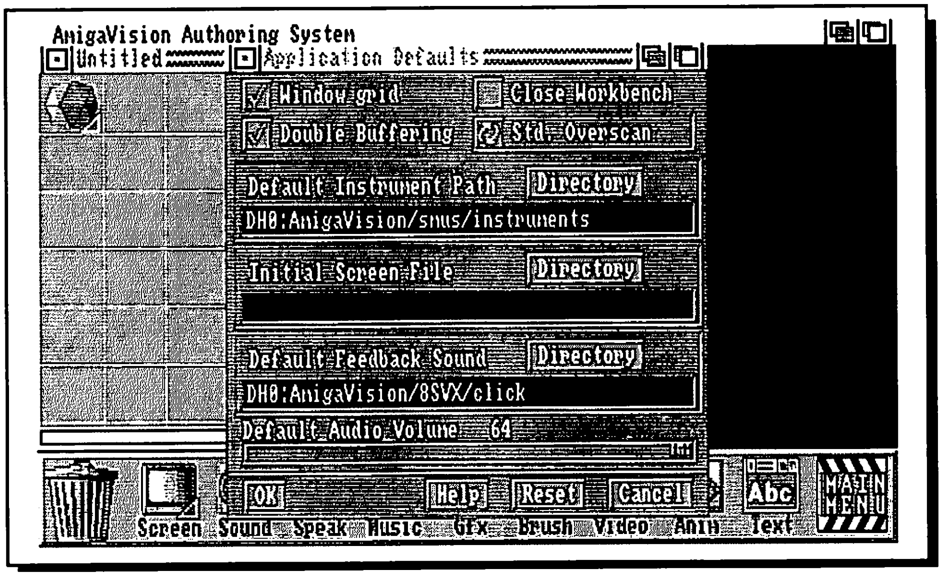
Selecting Save (ⒶS) from the Project menu saves the current version of your application, overwriting any file of the same name on disk. (If you have the backup option enabled, the previous version of the file is automatically saved with a suffix of .bak.) If your file has not been saved before and has the default name of Untitled, the Save requester will appear so you can give it a different name.

Defaults

The Defaults option (ⒶX) allows you to configure the AmigaVision development environment to your personal system (see Figure 4-6). With it you can specify a default path for musical instruments, select a digital feedback sound different from the standard beep, choose a default start-up screen (useful for initializing screen modes and color palettes), and set a volume level for sounds and music.

Each of the first three defaults (Instrument Path, Initial Screen, and Feedback Sound) have a Directory button above and to the right of the string gadgets. Clicking on a Directory button brings up the standard file requester, and you can specify a path or file name just by clicking on items listed in the windows. The Default Audio Volume is controlled by a sliding gadget. To use it, just position your pointer over the small knob in the gadget and move the mouse to the left or right while holding down the left mouse button. The volume level goes up or down depending on the knob's position. This volume-control gadget also appears in other areas of the program concerned with sound. Wherever you make changes to another sound gadget, that new volume level will take priority over the default level set here.

At the top of the Defaults requester are four gadgets. Three of these, Window Grid, Double Buffering, and Close Workbench, are simple toggle-style gadgets. Clicking on the button toggles the function on and off; the "on" setting is indicated by the presence of a check mark within the button.


AmigaVision Authoring System

Untitled

Application Defaults

[✓] Window grid

[ ] Close Workbench

[✓] Double Buffering

[✓] Std. Overscan

Default Instrument Path

DH0:AmigaVision/smus/instruments

Initial Screen File

Default Feedback Sound

DH0:AmigaVision/8SVX/click

Default Audio Volume 64

Figure 4-6
The Application Defaults requester lets you configure the defaults to your system

Using the Window Grid gadget, you can turn on or off the icon-size grid within the Flow and Content windows. It affects only the window that is currently active. When this option is not selected, the window appears as a solid color without any lines; when it is on (checked), a grid appears in the window. "On" is the default setting that is in place when a window is first created. In general, I advise keeping this option enabled. The grid makes it easier for you to position icons, and gives the Flow window a more structured overall appearance.

The Double Buffering option lets you choose whether the system is to use double buffering when displaying animations. This technique allows flicker-free animation by using two copies of the animation image. With double buffering, all screen updates occur on the hidden screen, and are made visible only when the image drawing is complete. Double-buffered animation is much smoother and cleaner looking, but it is very RAM intensive, requiring twice as much memory for the display screen.

When enabled, the Close Workbench option instructs AmigaVision to turn off the Workbench screen when you start running your application. This can free up memory (especially Chip RAM) for use in your applications. If there are any open windows or active programs on the Workbench screen (including a CLI or Shell), however, the Close Workbench command will have no effect.

The fourth gadget at the top of the requester allows you to cycle through various overscan-display options by clicking on it. Here you can determine the type of overscan display — standard or maximum — your specific application will support. The resolutions of the different forms of overscan are displayed in the table below.

Standard Overscan

352-by-240	-	Low resolution
352-by-480	-	Interlaced low resolution
704-by-240	-	High resolution
704-by-480	-	Interlaced high resolution

Maximum Overscan

368-by-240	-	Low resolution
368-by-480	-	Interlaced low resolution
736-by-240	-	High resolution
736-by-480	-	Interlaced high resolution

The default mode is standard overscan, although in this case overscan has no effect unless you display an overscan screen. If you have limited memory, you should use the standard mode unless something in your application demands the larger display.

At the bottom of the Defaults requester are four command gadgets, OK, Help, Reset, and Cancel. Clicking on OK indicates that you are satisfied with the settings for the default values; it removes the requester and establishes these settings as the defaults for the active window.

Selecting Help opens a text window containing context-sensitive information pertaining to the requester you are in when you choose it. The Help window will be discussed in depth later in this chapter.

Reset is useful when you make changes to the settings in the requester, but then decide that they are not what you really want. Clicking on Reset will set some, but not all, of the values back to where they were when the requester first appeared. Those affected by Reset are Window Grid, Close Workbench, Double Buffering, and Default Audio Volume. To return the others to their original values, you can either reset them manually or select Cancel.

Cancel discards all changes made to the requester, and removes the requester from the screen. Use it when you have changed your mind, or in those cases where you made changes to the default directories and paths that you really didn't want but cannot correct with the Reset gadget.

Print

Selecting Print (AP) allows you to send a listing of all or just part of your application to either a printer or a disk (see Figure 4-7). If you output to a printer you can print either in text mode or as a bitmapped (graphic) image.

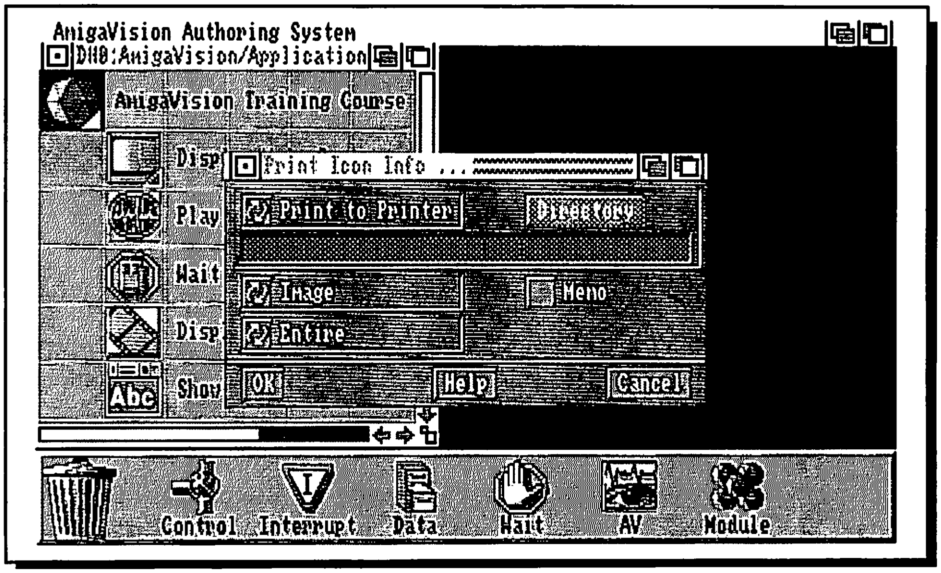


Figure 4-7
The Print requester lets you send things to the printer

However you decide to output your listing, keep in mind that in using Telescope, the output will not contain all the information contained inside the icon. For example, a defined screen icon contains information describing the particular type of screen and how it should be displayed, but this information will not appear in the printout or disk file.

You control the Print requester via a number of gadgets including Print to Printer/File, Image/Text, and Entire/Selected. If you select Print to File, the Directory command gadget and the file name gadgets become active, allowing you to specify the device and name to use for your file. If you specify Entire, the program prints the whole flow chart; if you choose Selected, you must also highlight an icon, which AmigaVision will then print. Using the Memo gadget, you can instruct AmigaVision to include the contents of the memo field in the listing. At the bottom of the Print requester you will find the standard OK, Help, and Cancel gadgets.

Present

The Present option (A.) allows you to start an application or course from the window you are in. A Flow window must be active in order to select this option; if no window is active, then the Present option (along with the Save, Save As, Defaults, and Print Project menu options) will appear ghosted in the menu and be unavailable. Present is AmigaVision's equivalent of the RUN command used in BASIC and other similar languages. (If you click on an AmigaVision application icon from the Workbench, the application will automatically run unless the Edit flag is enabled. See Chapter 1 for more information on this subject.)

Applications

The Applications option lets you create floppy-disk-based copies of your applications and courses and install copies of courses from one disk to another (see Figure 4-8). It has two submenu options, Create Diskette(s) (A]) and Install/Relocate Applications (A[). Create Diskettes is referred to as Create Runtime in the AmigaVision manual and in program version 1.31.

After you have finished creating an AmigaVision course or application, you can use Create Diskette(s) to make a runtime copy of the program and all its files onto a floppy disk, from which anyone who owns AmigaVision can access your applications. Selecting Create Diskette(s) brings up a requester that allows you to select an application to install on floppy disk (see Figure 4-9).

This requester contains two text gadgets, one for selecting the source file (using the standard file requester), and one for typing in a name for the destination file.

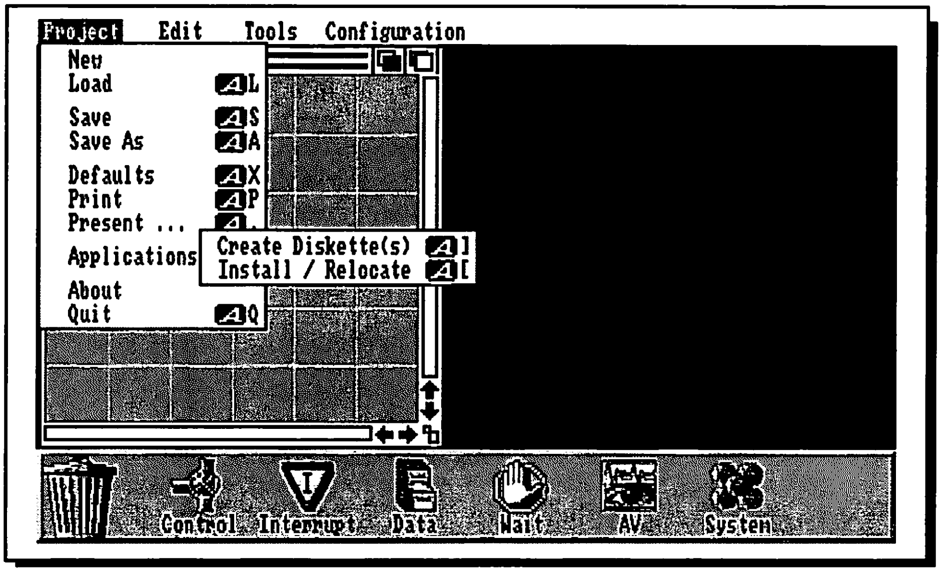
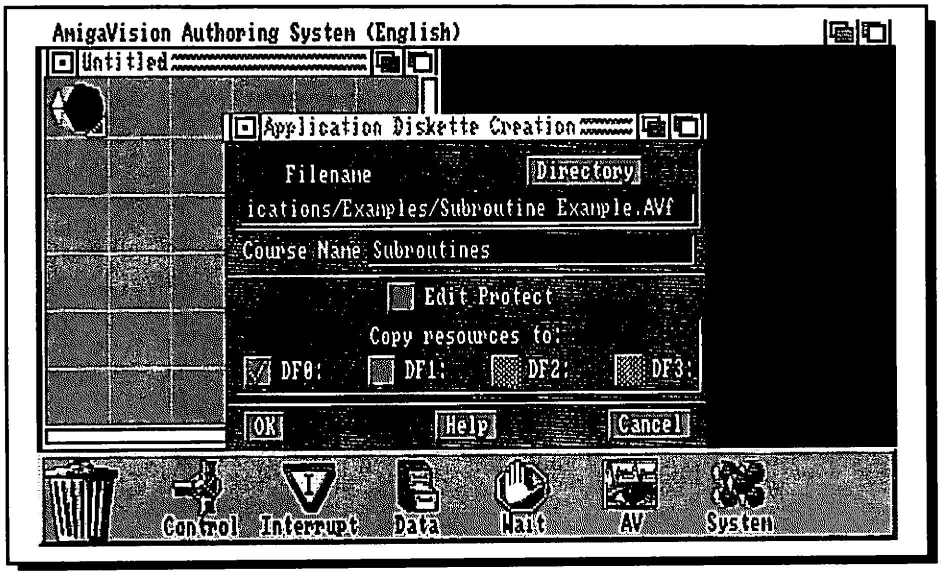


Figure 4-8
The Applications submenu lets you create floppy-disk-based copies of your application



ications/Examples/Subroutine Example.AVf

Course Name Subroutines

[✓ DF0:] DF1: DF2: DF3:

Figure 4-9
The Application Diskette Creation requester lets you select an application to install on floppy disk

After selecting and naming your file, you can protect it from alteration by clicking on the Edit Protect gadget. Doing this creates an AmigaVision program that can be presented, but cannot be viewed or edited. The Edit Protect option is useful when you want to distribute applications but do not want anyone to be able to alter or examine the code. You should, however, choose it only when you have completed the development and testing of the program. Also, because edit-protected AmigaVision programs cannot be converted back to editable programs (at least in the current version of AmigaVision), you should always keep an editable copy of the application.

There are four additional gadgets, labeled DF0:, DF1:, DF2:, and DF3:, that correspond to floppy drives one through four. Only those floppy drives that are present on your system will have functional buttons, the other buttons will be ghosted and won't respond if you click on them. Using these drive buttons, you can indicate which drive(s) you want the Runtime application installed on. If you have two disk drives, you should make use of both when creating Runtime disks; doing so can save you a lot of time compared to using a single floppy disk.

At the bottom of the Print requester are the standard OK, Help, and Cancel buttons. Once you have selected the source and destination file names and clicked on the appropriate floppy-drive buttons, you can then select OK to begin the Runtime-creation process. AmigaVision will first read your course into memory, and then begin the job of copying the course and its data files onto the floppies. In doing so, AmigaVision makes new references for all data files so you will have no trouble running the course from the new disks.

Along with creating your Runtime disks, AmigaVision also writes a text file to the first disk (if there are multiple disks in the Runtime set) called a .Log file. This text file contains information describing any problems AmigaVision might have encountered in creating your floppy-based program. For example, suppose the program uses a digital sound but the sound file was not where the Sound icon said it was supposed to be. In that case, AmigaVision would add an entry to the .Log file saying 'Sound xxxx could not be found'. You would then have to find the problem and correct it in your original flow chart, and then repeat the Create procedure. This is a good example of why it is so very important to make sure your program is well tested and error free before creating the Runtime disks.

During the process of creating the new application disk, Create generates a series of directories on the new disk and names these directories according to their file types. It can set up SMUS, 8SVX, ILBM, or ANIM directories, with each containing the appropriate kinds of files. If the application is spread over several disks, there might be duplicate directories on each disk, but the contents of each disk will be different. For example, if you use more pictures than will fit on a single disk, there will be ILBM directories on two or more disks.

Create also modifies your application, changing the path descriptions in the icons from where they were originally to the directories of the new disks. This way the new floppy-version of the course can find the data files it needs, without you having to edit any icons to reflect the new locations of the data.

Limitations of Applications Create

There are some source-file limitations associated with the Create Diskette(s) option. First, it will not copy single files that are too large to fit onto one disk. So, if you have animations, digitized sounds, or databases that are larger than a floppy can store, AmigaVision will add an error message to its .Log file.

In addition, AmigaVision usually does not copy fonts to the Runtime disks. The reason for this is that so many fonts are copyrighted commercial products that the creators of AmigaVision did not wish to be a party to any software piracy. For the same reason, the program also does not copy instruments used in the SMUS music files.

The only way to get AmigaVision to copy instruments or fonts is to preload them into a Resource icon. With fonts this is fairly easy, as you will know what fonts you used in your program. SMUS instruments are a different story.

Since you may not know exactly what instruments are required by a SMUS music file, trying to preload them can be quite a problem. The solution is to read the .Log file and see if it reports any problems with instrument files. If it does, you must edit your original AmigaVision file, adding those instruments mentioned in the .Log file to the list that the Resource icon will preload. Then, you need to repeat the Create process to make new Runtime disks.

If you preload your fonts using the Resource icon, the font files will be copied — but there are still major problems. For example, if you preload a Times 24 point font into your program, the Create process will copy only the 24-point font file, which would appear on your floppy disk with the file name 24. If you later load another 24-point font, it will replace your Times 24. Also, there will be no Fonts: drawer on the disk — just the files with point sizes for names. Later, when you try to install or use this AmigaVision program, these fonts files would be ignored. So pre-loading fonts really isn't of much use.

For fonts, the best solution is to copy the appropriate font subdirectories into a FONTS: drawer on your floppy disk, making sure to copy the .font file as well as the font itself. To accommodate the Times 24-point font, you would make a font directory, then a Times subdirectory, and then copy the font over to the subdirectory. Here's an example of what steps are necessary to copy the Times font to DF0: through the CLI.

```
CD DF0:
MAKEDIR fonts
CD fonts
MAKEDIR times
COPY sys:fonts/times/24 to DF0:fonts/times
COPY sys:fonts/times.font to DF0:fonts
```

Note that in this example we created a Fonts: directory on DF0:, and a Times subdirectory within it. Then the 24 point Times font was copied to the Times subdirectory, and the Times.font file was copied to the Fonts directory.

You will have to make some provisions for the end user of the application to add the applications fonts to their own Fonts: directory. You can do this fairly easily with an autoexecute batch file (setting the S bit of the batch file with AmigaDOS's PROTECT command). Otherwise you must edit your original AmigaVision program to use some of the standard fonts in place of your specialized fonts.

If you create data files larger than you can fit onto a floppy disk (as hard disk users with three or more megabytes of memory in their systems often do), you will have to take special steps to get your data onto floppies. The technique I prefer is to use a hard-disk backup program such as QuarterBack (Central Coast Software) to copy a large file onto multiple floppy disks, and to install the file from floppy to the destination hard disk after using the Install option for the rest of the AmigaVision application.

You can also use a program such as the public-domain utility Split, which breaks up the files into smaller, more manageable pieces that you can then copy manually onto floppy disks along with the other Runtime files. After you have installed the Runtime course onto its destination drive, copy the split files onto the course disk, and use the AmigaDOS Join command to hook them together so that they will function properly. Both of these procedures — making a backup and splitting and joining large files — are somewhat difficult, but until a future version of AmigaVision offers support for larger data files, it is something we just have to cope with.

A final consideration. If you intend for your application to run from a floppy disk(s), keep in mind that all your data files must be able to fit onto a floppy. In this situation there is no place for long animations, digitized sound files, or large database files.

Applications Install/Relocate

The opposite of Create is the Install/Relocate process. Select this menu option to install an application from floppy that has been previously generated with the Create Diskette(s) option, or if you want to move an existing application from one directory to another.

Once you choose the Install/Relocate option from the Applications menu, you are presented with the Application Install/Relocate requester (see Figure 4-10).

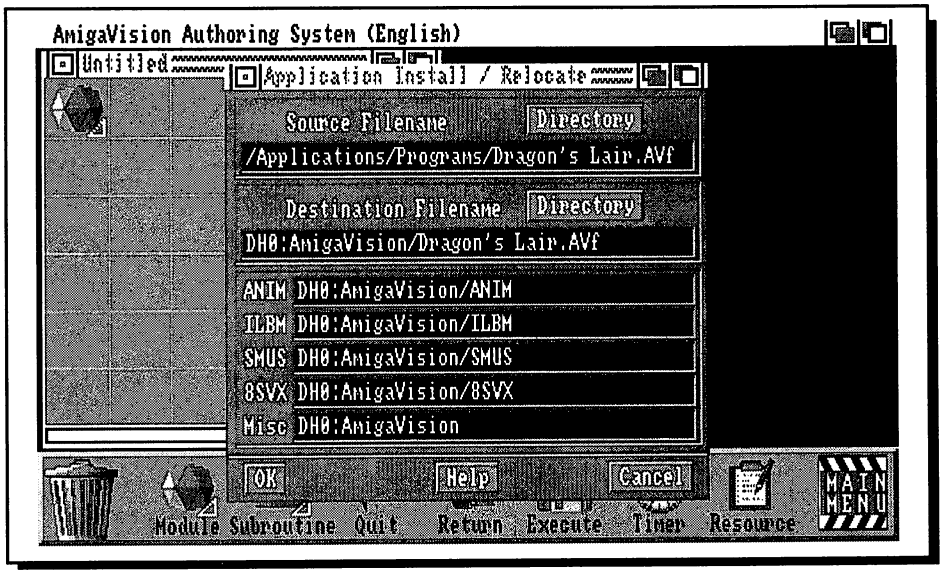

AmigaVision Authoring System (English)
Untitled
Application Install / Relocate
Source Filename
/Applications/Programs/Dragon's Lair.AVf
Destination Filename
DH0:AmigaVision/Dragon's Lair.AVf
ANIM DH0:AmigaVision/ANIM
ILBM DH0:AmigaVision/ILBM
SMUS DH0:AmigaVision/SMUS
8SVX DH0:AmigaVision/8SVX
Misc DH0:AmigaVision

Figure 4-10
The Applications Install/Relocate requester appears when you choose the Install/Relocate option

Using the first Directory command gadget, select the source AmigaVision application you want to install. Then, using the second Directory command gadget, select a destination. (As long as I am placing the file in a new location, I usually use the same name for the destination as I used for the source, otherwise, it would overwrite the old file.) If you want to make some changes in the destinations for the data files, just click in the text gadgets and change them to the locations you prefer. AmigaVision will automatically move the data files to their proper directories, altering the path names for the application to reflect the new homes, just as it did in Create. If for some reason you are dissatisfied or change your mind, click the Cancel button to abort the process. As usual, there is a Help button available to offer you some information on the task at hand.

About

The About menu option displays the version of AmigaVision you are using. The original release version was version 1.31, while the version we are discussing here in this book is release 1.53G. About also tells you the amount of chip and expansion memory available in the system.

If About says you are using the initial release version of AmigaVision (version 1.31), you should contact your local dealer about getting the latest upgrade. This is important not only because of the improvements and additions in this later version, but because there were some bugs in the first release that have been corrected.

Quit

Selecting Quit (AQ) from the Project menu exits AmigaVision entirely. Before this option returns you to the Workbench, it checks to see if you have made any changes to the currently active Flow or Content window. If you have, it gives you the option of saving these changes or closing the Flow window without saving the new changes. (You can also abort the Quit operation by selecting Cancel.)

If you have multiple Flow windows open, the program will run through each of them, checking for changes since the last save and, if it finds some, gives you a chance to save them. In doing this, however, AmigaVision does not move the Flow window it is referencing to the front. This means that if you have multiple windows open and you want to be sure of what you are saving, you should line them up so that you can see at least part of each before selecting Quit. That way you can see which Flow window is active at the time the Quit requester appears.

The Edit Menu

The Edit menu has six options you will be using often during the creation of your applications (see Figure 4-11). These editing tools include Collect, Copy, Info, Preview, Telescope, and Search.

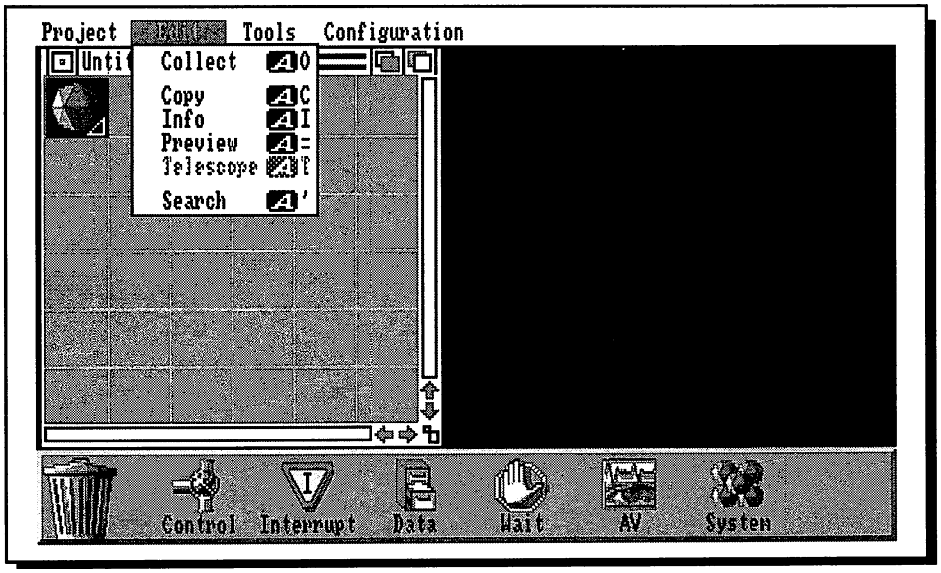


Figure 4-11
What you get when you choose the Edit menu

Collect

Collect (AO) is used to combine a linear sequence of icons into a group under either a Module icon (if collected in a Flow window) or a File Folder icon (if collected in a Contents window). Why would you want to do this? Because doing so is very useful in a number of situations. Let's say, for example, that you have created a flow chart containing just one main Module icon and that all the remaining icons fall directly under it (as we have done in Figure 4-12 later in this chapter). (As you will read in later chapters this is not good program-design form, but AmigaVision beginners will often use it until they have some experience in structuring applications.) With Collect, you can easily grab any number of these icons and place them as children of another Module icon. They can then be telescoped (the Telescope function is detailed later in this chapter) into this Module icon, and the whole group can be moved or copied by just manipulating this single icon.

Collect is quite simple to use. Once you choose Collect from the Edit menu, the mouse pointer changes shape slightly and appears to include a small rectangle. (As long as this rectangle is visible, you are in Collect mode.) To group a section of icons together, place your mouse on either the first icon (the icon at the top of the sequence you want) or the last icon (the icon at the bottom of the sequence you want). Then, holding down the left mouse button, move your mouse either up or down. A rectangle, exactly one icon wide will appear, and as you move your mouse, the rectangle will enlarge to encompass all of the icons you pass over. If the icon sequence you are collecting is larger than can appear in the on-screen area of the Flow window, hold down the right mouse button as well while moving your mouse beyond the Flow window border. This causes the icons to scroll, and allows you to collect more icons than just those that can be shown on screen. You can also use the Cursor Up and Cursor Down keys to scroll through the Flow window. The choice is yours — use whichever feels easier to you.

When you release the mouse buttons, a requester appears asking if you really want to collect the selected icons. Click on OK and a new Module (or File Folder) icon appears, and all the collected icons appear as child icons under and to the right of the parent icon. A small, solid-color triangle is added to the lower right of the new parent icon, to indicate that it has children. (Parent icons without children have a triangle that is outlined only, not solid.)

Clicking on Cancel aborts the collection process, but keeps you in Collect mode. If you should change your mind about grouping icons while in the midst of dragging your mouse over them, you can also abort just by reversing the direction in which you are moving your mouse, returning to the point at which you started, and releasing the mouse button(s). No changes will occur.

An important consideration to keep in mind when collecting icons concerns variables. AmigaVision has both local and global variables. These have a distinct hierarchy, and variables *created* (used for the first time) within a Module or Subroutine icon have meaning only for that module or subroutine's children. (For more information on local and global variables, see the reference on Variable, Module, and Subroutine icons.)

Copy

Selecting Copy (🅰C) allows you to copy either a single icon or a parent icon and its children from one location to another in your Flow window. Copy does not remove the icon(s) from the original location, it just makes a duplicate in the new location.

When you select Copy from the Edit menu, your pointer changes, appearing to have a shadow image of itself. (Copy mode stays in force until you reselect Copy from the Edit menu.)

To duplicate a single or parent icon, simply point to it; then, holding the left mouse button down, drag the icon to a new location. Just as in Collect mode, you can scroll through the contents of the Flow window while copying by holding down the right mouse button and moving the mouse up or down (while keeping the left mouse button depressed), or by pressing the up or down cursor keys (again, while holding the left mouse button down). Once you have moved the icon to a new location, just release the left mouse button and the copy (and all of its children, in the case of a parent icon) will be deposited at this position.

If you want to *move* (not copy) an individual icon or a group of icons you have collected, turn Copy mode off. Then, just put the pointer over it, hold the left mouse button, and drag it to the new location. The icon (and any children it has) will be deposited at this position, and will be removed from the original position.

Finally, if you have more than one Flow window open and want to copy one or more icons from one window to another, you do not need Copy mode (although if you are in Copy mode it won't hurt anything). Just drag the icon from one window to the other, and a new copy of the icon will be placed in the destination window. Like Copy mode, the original stays where it was.

Info

Using Info (ⒶI) you can either define an icon, or view or modify its existing definition. To use this option, click once on an icon to highlight it with a dark gray background, then select Info from the Edit menu. A requester, the style of which depends on the type of icon you selected, will appear. You can then examine the information currently in the fields and gadgets of the requester, and add or edit the information as necessary.

Selecting Info has exactly the same effect as double-clicking on an icon, and in fact, it is much easier to double-click on the icon than it is to click the icon once, move the mouse to the Edit menu, and select Info. Because of this, Info is practically useless and will not be one of your more often-used options.

Preview

The Preview (Ⓐ=) option allows you to test an icon. Highlight the icon you want to test and select Preview from the Edit menu, and the icon will be presented, along with any partner or children icons.

In Preview mode, AmigaVision will attempt to display the selected icon in a manner that is as close as possible to how it will appear during the actual presentation. For example, if the selected icon is a screen icon that calls up a picture, and if a transition is to occur between it and the previous image, the earlier image will load first and the new image will be displayed using the selected transition, so that you can accurately judge how well the transition works.

A preview can be adversely affected, however, by other factors that AmigaVision does not take into consideration. Specifically, if you attempt to preview an icon or group of icons that have some dependence on a variable defined earlier in the flow chart, AmigaVision will not take the value of that variable into consideration for the preview process. As an example, suppose you are previewing a Loop icon in Counted mode that uses a predefined value for the variable *i* as the upper limit for the loop. If AmigaVision doesn't know the value of *i* when it tries to preview the icon, you will get a requester stating that *i* is an unknown variable and instructing you to define it before attempting to use the icon. (This is the same error message you get during a presentation if you attempt to use undefined variables.) As another example, let's say you are using the Speak icon to give voice to a phrase contained in a variable. Again, you will get the undefined-variable message and the preview will abort.

Many icons, including Screen, Anim, Sound, Speak, and Music, offer the Preview option through their requesters: simply clicking on the Preview button lets you see them. Some icons, however, do not offer that option, and in these cases you must select Preview from the Edit menu in order to test the command.

Telescope

The Telescope (🅰T) option is used to hide or expose sections of your program. It affects only parent icons that have children. These include some icons from every icon submenu, specifically, the Module, Subroutine, Screen, Grouped, Select, Form, Keyboard Irq, Mouse Irq, and Loop icons. You can easily identify icons representing parents with children by the characteristic solid triangle on their lower right. If the triangle is merely an outline it indicates that the parent has no children; such parents cannot be affected by the Telescope function. If the triangle is solid, however, you can be sure that the parent has one or more children associated with it.

Children of a parent icon can either be visible, appearing in a column below and one square to the right of the Parent, or they can be invisible, telescoped up inside the Parent. If you click once on a parent icon that has visible child icons and then select Telescope from the Edit menu, all the child icons will seem to vanish. They will still exist, but will be hidden from sight. If you select the parent of children that have been telescoped, selecting the Telescope option causes all the child icons to appear on screen, indented under the parent.

The collapse/expand functionality of Telescope works only one level deep. What exactly does this mean? Well, suppose you have a parent icon with children, and some of these children are themselves parents of one or more generations. Let's say that in your current display all the icons — from the highest order parent on down — are fully expanded and visible. If you select the topmost parent and telescope it inward, collapsing all its children, the children of this icon that are themselves parents will remain unaffected. When you select this topmost parent and choose Telescope again, all the child icons expand and appear just as they did when the main parent icon was collapsed. Conversely, if the families of all the parent icons under the original parent icon are collapsed, using Telescope on the outer parent will not open up the inner parent icons. These will still appear as single icons when expanded, identifiable as parents with children only by the solid triangle on their lower right.

This ability to hide sections of your application can be very useful. It allows you to see more of the overall logic flow of your program without having the entire Flow window filled with long strings of icons that represent the logic of the subsections. Once you have created and debugged a subroutine or submodule section of a program, you can hide it away using the Telescope option and essentially treat it as if it were a single command. Subroutines, once created and tested, should be placed at the bottom of your program flow chart, each collapsed into a single icon using the Telescope function. Then, if you need to look through the list of subroutines, you can scroll through just the main subroutine icons instead of through the entire list of icons that make up each subroutine, saving you time and making the job easier.

Search

The Search (🅰 ') function allows you to quickly find any icon in your application by entering the name of its label. When you select Search from the Edit menu, the Icon Search requester (see Figure 4-12) appears. You can search for the desired icon either forward through the flow chart from the point of the currently selected icon, or, by clicking on the multistate gadget, you can choose to search through the entire icon list. (If no icon has been selected when you choose Search from the Edit menu, the multistate gadget is ghosted, or unavailable, and a search through the entire list is the only type of search available.)

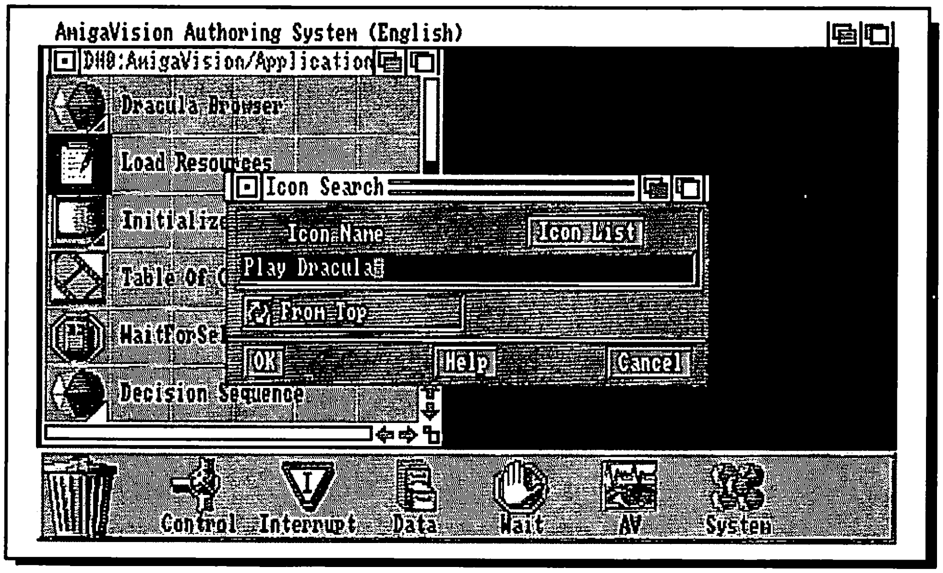


Figure 4-12
The Icon Search requester appears when Search is selected in the Edit menu

To search for an icon, you enter its name into the text gadget in the Search requester. (This name should correspond with the name given to the icon in the Icon Name text gadget found in the requesters of all command icons, excluding a few such as the Goto icon.) Only icons that have been given names can be located with this function. This is one example of why it is a very good programming practice to give every icon a name when you first define it via its requester! In naming an icon, you should attempt to describe both its function and what is happening at the point in your application where it appears. For example, if there are several places in your program that require the user to make a selection, the icon name "Main Menu User Input" is obviously a better choice than merely "User Input."

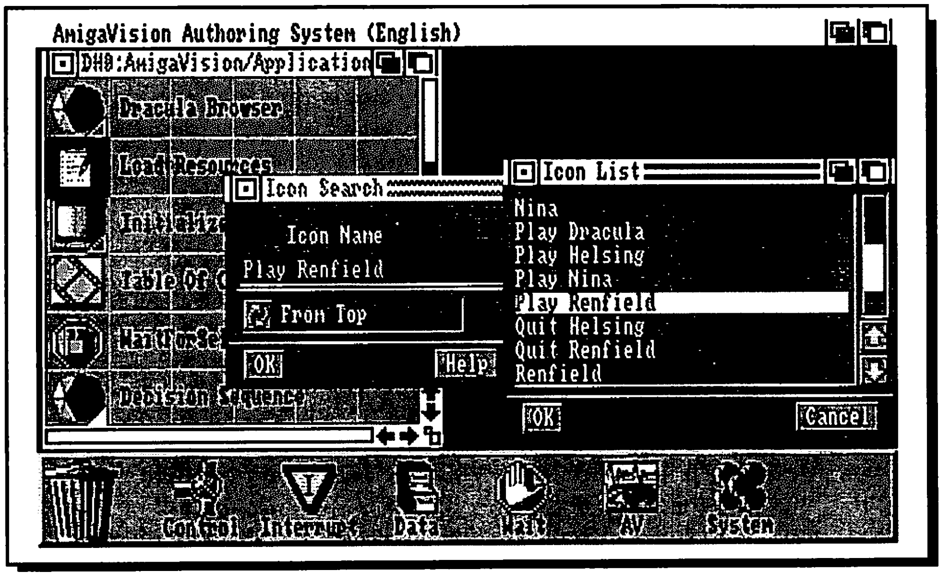
One thing to keep in mind when using the Search option is that it, like AmigaDOS, is case-insensitive. This means that it treats uppercase and lowercase letters as the same characters. So, it does not distinguish between the words "AmigaVision" and "amigavision."

The Search requester can accept 'wildcards' as part of the search string. There are two wildcard characters available, the question mark (?) and the asterisk (*).

The question mark can substitute for a single character at any point in the string. For example, suppose you need to find a screen icon and that you know the name contains five letters but can only remember that the first four are Loui. Entering Loui? into the Search text gadget would find the first five-letter name starting with Loui. You can use as many question marks as necessary. Entering Lou??, for example, will prompt the program to find the first five-letter icon name beginning with the letters Lou.

The asterisk (*) is a more powerful wildcard, as it can substitute for entire sections of text. As an example, suppose you wanted to find an icon named Play Music, but could remember only that it contains the word Music. Entering '*Music' into the text gadget would cause the search function to locate the first icon that has Music as the last five characters in its name. As with the question mark, you can use more than one asterisk wildcard in your search. Thus, if you wanted to search for the icon named Play Loud Music, but knew only that the word Loud was the second word of the name, you could find the icon simply by entering the search string *Loud*.

If you don't remember any part of the icon name you wish to locate, click on the command button labeled Icon List in the Search requester. This brings up a new requester listing all the named icons in your application (see Figure 4-13).



Play Renfield

Nina
Play Dracula
Play Helsing
Play Nina
Play Renfield
Quit Helsing
Quit Renfield
Renfield

Figure 4-13
The Icon Search and Icon List requesters are used to find an icon

If your list contains more names than can be displayed within the requester's window, you can scroll through the list by clicking on the up or down scroll arrows. To select one of the icons listed, just click on its name. That name will appear in the text gadget of the Search icon; you can then select OK to exit the Icon List requester and return to the Search requester. You can also select a name by double-clicking on it in the Icon List window. Doing this places the name in the Search text gadget and closes the Icon List requester — just as if you had clicked on the OK button. Of course if you change your mind you can abort the whole thing by clicking on the Cancel button.

Tools Menu

The third menu is the Tools menu (see Figure 4-14). Here we have three distinct options: the Object Editor, the Database Editor, and the Videodisc Controller. All three of these powerful tools are discussed in later chapters, so for now we will content ourselves with a brief description of each.

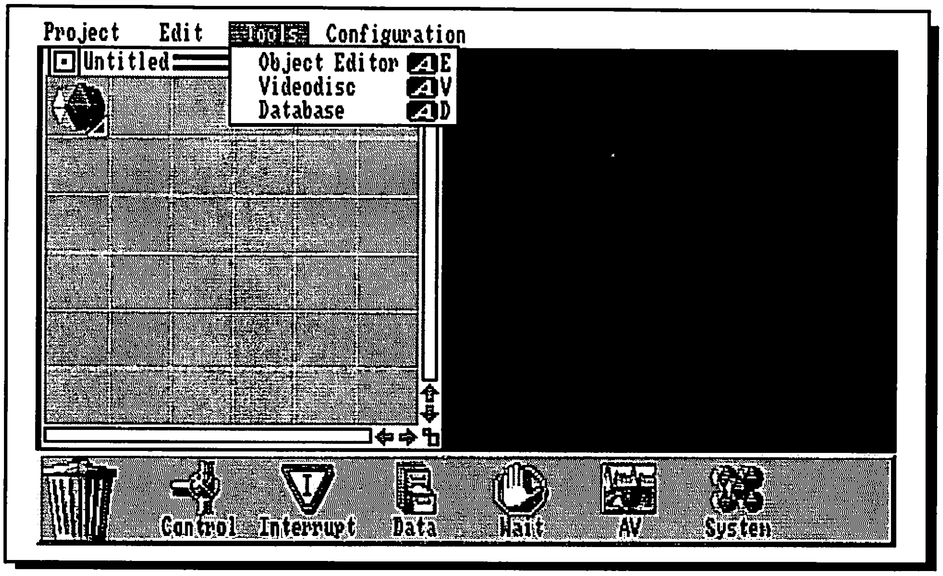


Figure 4-14
What you get when you select the Tools menu

Object Editor

The Object Editor (ⒶE) is in many ways the heart of AmigaVision. It is here you will actually implement your interactive menus and displays. Among the features of the Object editor is the ability to define areas of the screen to be sensitive to user input via the mouse. These areas are known as hot spots. A hot spot can take any one of a variety of shapes, ranging from simple lines, rectangles, and ellipses to complex polygons. Hot spots can also consist of lines of text containing predefined phrases or variables defined within your applications. If you need more complex images, you can add brushes created with programs like DeluxePaint III (Electronic Arts). You can also use full-screen pictures (in any graphics mode or screen resolution), and place invisible hot spots over areas of the pictures. For the ultimate hypermedia displays, you can overlay your hot spots onto frames of video from a laser disc!

The Object editor is also available from a variety of the command icons. These include the Mouse, Keyboard, Gfx, Text, and Irq icons.

Database Editor

The Database Editor (ⒶD) gives you access to AmigaVision's dBASE III-compatible database (dBASE III is an MS-DOS database manager by Ashton-Tate). All new databases must be created and defined from the menu or a separate database program; you cannot set up a database from within an AmigaVision program itself, although you can edit databases previously created with dBASE III or comparable database software such as Superbase Professional (Precision Software).

AmigaVision's database supports four different data formats. These are strings (alphanumeric text), numeric (numbers), Boolean (yes/no), and date (calendar dates). String fields can be up to 254 characters wide; numeric fields can be a maximum of 15 characters wide, including the decimal point. Both date and Boolean fields are fixed at default widths of 1 (Boolean) and 10 (date). Each individual record can have up to 128 different fields with a total of 4,000 characters. The total number of possible records is determined by the amount of storage space available in your system.

You can define some fields as key fields, and then automatically index and sort the records in your databases according to the contents of these fields. Because you can have multiple key fields in the same database, you can perform multiple-level sorts on your records.

Videodisc Controller

The Videodisc Controller (AV) lets you access and control an industrial laser-disc player that is connected to your Amiga's serial port. You can use this controller to explore video discs, gathering the frame numbers of the sequences you want to incorporate into your applications. You can even use it just to watch a laser-disc movie on your Amiga!

Configuration Menu

The Configuration menu (Figure 4-15) contains three options: Workbench Closed, Video Setup, and Preferences. This menu was called System in the original (version 1.31) release of AmigaVision, when it contained only the Workbench Closed option.

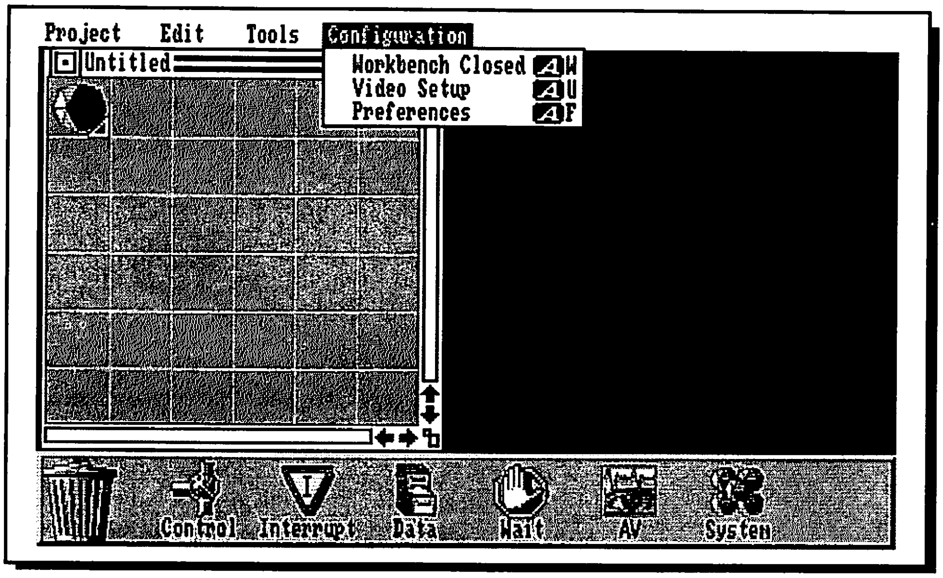


Figure 4-15
What you get when you select the Configuration menu

Workbench Closed

The Workbench Closed (AW) option closes the Amiga's Workbench screen, freeing up valuable chip RAM that you may need for your own applications.

Video Setup

The Video Setup (ⒶU) option brings up the Video Setup requester (see Figure 4-16). In this requester you must specify the type of laser-disc hardware you have (if any) by clicking on its name in the list you see in the window. Also, you must indicate which device driver you wish to use.

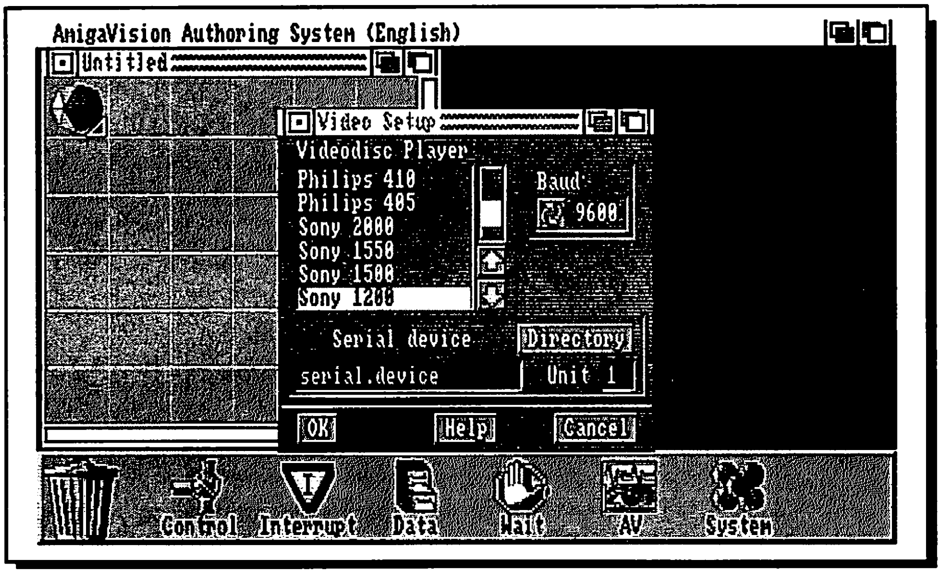


Figure 4-16
The Video Setup requester appears when you select the Video Setup option

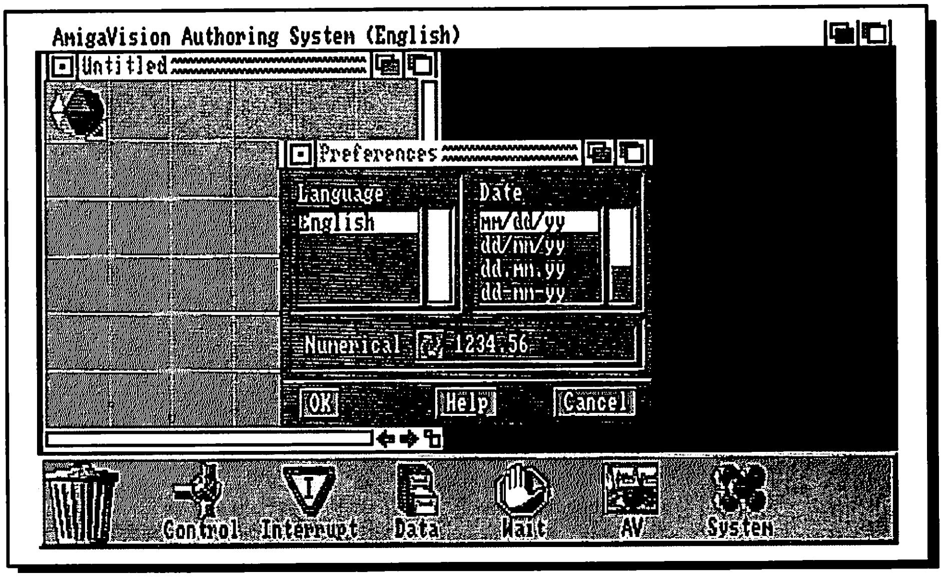
Besides the standard serial.device driver you are most likely to use, AmigaVision allows you to connect the laser-disc player to other serial ports that are added to the Amiga via multiport serial cards. You can use the standard file requester to locate the device driver by clicking on the Directory command gadget. If you are using a non-standard serial device, it may be necessary to change the unit number. Check with the documentation that comes with your multiport serial card to see what value it suggests for the port you are using.

Because different laser-disc players work at different baud rates, you can specify any speed from 1,200 baud to 9,600 baud. In general, you should choose the fastest speed that your laser-disc system supports, although in some situations it is necessary to use a speed slower than the maximum. Make sure that the speed you select here matches the baud rate selected on the laser-disc unit itself. To change rates in AmigaVision, just click on the multistate gadget to cycle through the various speeds.

Once you have configured AmigaVision to work with the laser-disc system you are using, click on the OK gadget. The information you have specified is then saved to your system Devs: directory, and the appropriate driver for your laser disc is placed in a special subdirectory of Devs: called Players. From then on, each time you load AmigaVision, this information is used to tailor the laser-disc commands to your specific hardware.

Preferences

The Preferences submenu (ⒶF) option brings up a requester that customizes your AmigaVision setup to work in the language of your choice (see Figure 4-17). You can also switch between six different formats for displaying dates, using the one you are most accustomed to. And you can select the type of character used to mark a decimal place, switching from a period to a comma depending on what character is used in your country. Once you have made your selections, click on the OK gadget and this information is saved as an AmigaVision default.



Language
English

Date
mm/dd/yy
dd/mm/yy
dd.mm.yy
dd-mm-yy

Numerical 1234.56

Figure 4-17
The Preferences requester allows you to customize your AmigaVision setup

Windows

There are two types of windows used in editing AmigaVision programs: Flow and Content windows (see Figure 4-18). Each serves a different editing purpose, but both share some common characteristics. You can alter the sizes of both kinds of windows using the standard sizing gadgets in their lower right corners. Both windows have horizontal and vertical scrolling gadgets (shaped liked small arrows), which allow you to scroll through large applications. In the upper corners both have the standard Amiga close gadget, which you click on to close the window and remove the contents of it from memory. Remember that if you do not save the contents of these windows before closing them, any changes will be lost forever. If you have manipulated the icons in the window in any way since you last saved them, AmigaVision will give you a chance to save by displaying a special requester (see Figure 4-19). You can save by selecting the Save button, go ahead and close the window without saving by selecting the Close button, or abort the whole process by selecting the Cancel button. (When you choose Cancel, the window does not close.)

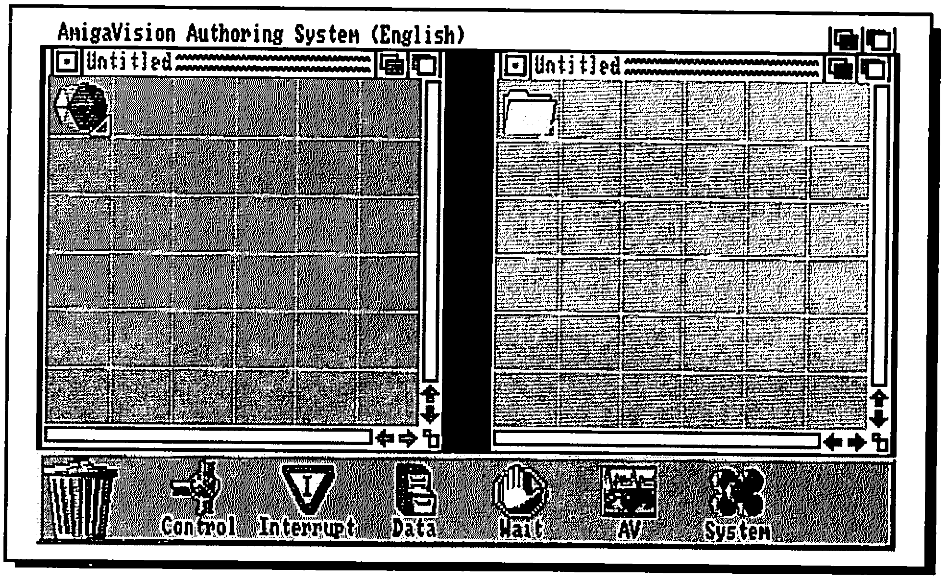
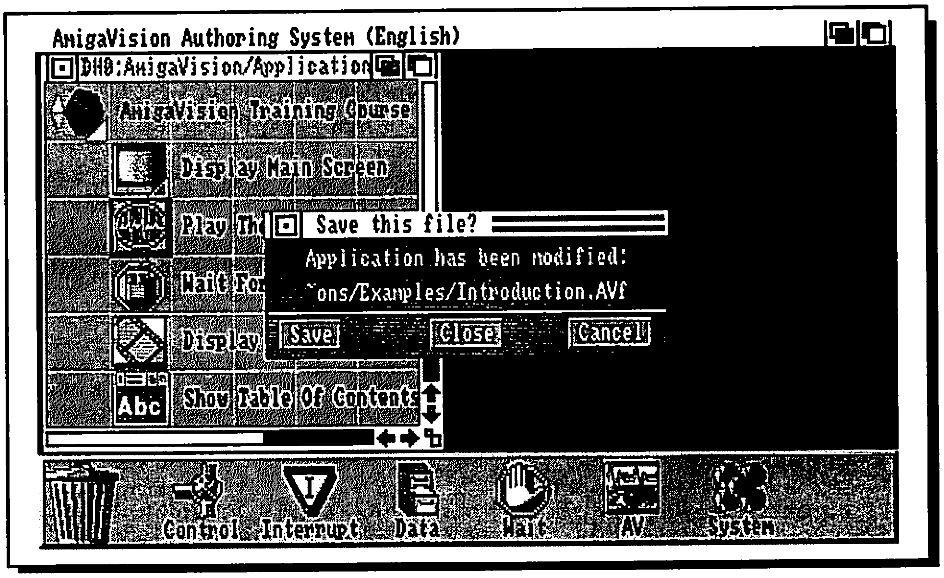


Figure 4-18

The Flow and Content windows are the two types of windows used in editing AmigaVision programs



Save this file?

Application has been modified:
...ons/Examples/Introduction.AVf

Save | Close | Cancel

Figure 4-19
The Save/Close requester allows you to save any icons changed after the last save

In the upper right of each window you will find the standard Window To front/back gadgets also found on most other Amiga windows. If you keep more than one window open on the screen at the same time, these gadgets will come in handy for moving windows behind and in front of the current window. At the top of each window is the name of the application, and by clicking and dragging on any part of this title bar with the mouse, you can move the window to any other position on the screen.

Finally, if you are using Workbench 2.0, you will also see a second gadget in the top right of each window. This is the Tiny Window or Iconify gadget which, when selected, turns your window from its current size to a smaller size. It works like a toggle switch; click it again and the window returns to the size and position it was before the last click.

Flow Window

The Flow window is your work area, and it is here that you place the AmigaVision command icons to generate the executable flow chart that becomes your AmigaVision program. If you have enabled Window Grid mode from the Project Defaults menu, the Flow window will be filled with light gray vertical and horizontal lines that create a grid for placing the icons.

You can have multiple Flow windows open, each containing a different program. To open an additional Flow window, select New Flow from the Project menu or use the keyboard shortcut command (ⒶN).

Each Flow window starts with the Module icon, the round geodesic icon. This first Module icon can act either as a parent to following icons positioned to the right, or as a sibling to icons positioned directly below it. (The connection of various commands is detailed in the Icon Relationships section later in this chapter.) Each Flow window *must* have at least one icon — in the top, leftmost corner. The default icon placed there when you open a Flow window is the Module icon, but you can insert other icons into that position, and then delete the Module icon by dragging it into the Trashcan. If you delete all the icons in a Flow window, however, a new Module icon will pop up. Like nature, AmigaVision abhors a vacuum and insists that there be something in its Flow windows at all times.

Content Window

Content windows are completely different from Flow windows. To start with, they are a different color — a slightly lighter gray (see Figure 4-20). Instead of a Module icon, each Content window contains a File Folder icon, which is used (like the Module icon) as a means of grouping or collecting icons together. When you save a Content window to disk, AmigaVision adds a slightly different suffix (.AVc) to the file names, compared to the .AVf suffix it adds to Flow window files.

Content windows are not used to create AmigaVision programs, and you cannot execute programs from within them. What they do is act as a repository, or organized storage site, for the elements of multimedia applications: pictures, animations, sound effects, music, and text files, and even video sequences. As you might have noticed, all these elements are members of the Audio Visual (AV) group of command icons. In fact, the only icons that can be placed within the Content window and its File folders are Audio Visual icons. Any attempt to place an illegal icon in a Content window will cause an error requester to appear (see Figure 4-21).

To use Content windows to their best advantage, you should have a need to call up groups of data elements (pictures, sounds, etc.) repeatedly in one or more applications. For example, suppose you have a number of SMUS files that are to be used as background music in various applications. By creating a File folder and filling it with these files' icons, you can just grab the music score you want and drag it into the Flow window (see Figure 4-22).

More importantly, you can have multiple File folders in a single Content window, with each folder containing different elements (see Figure 4-23) and acting as a library of multimedia elements.

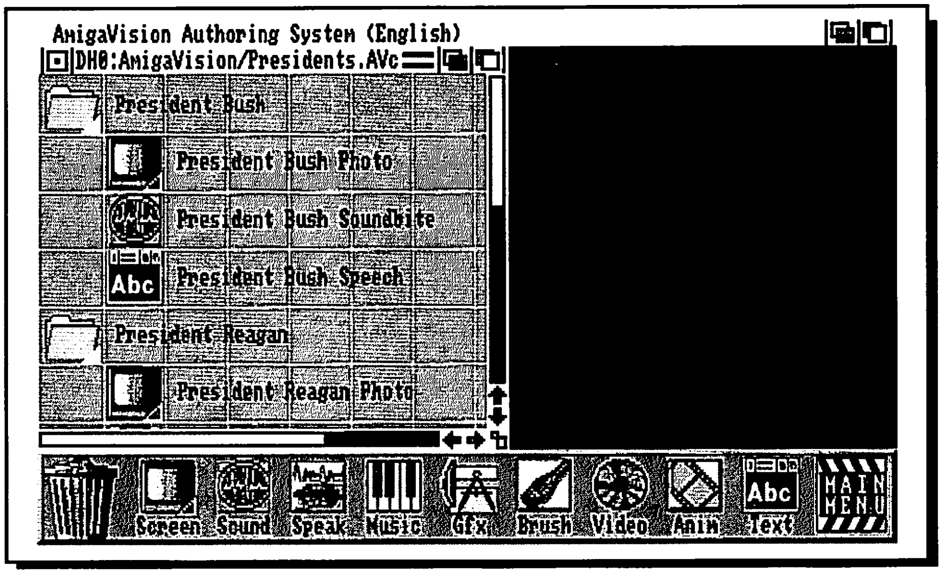
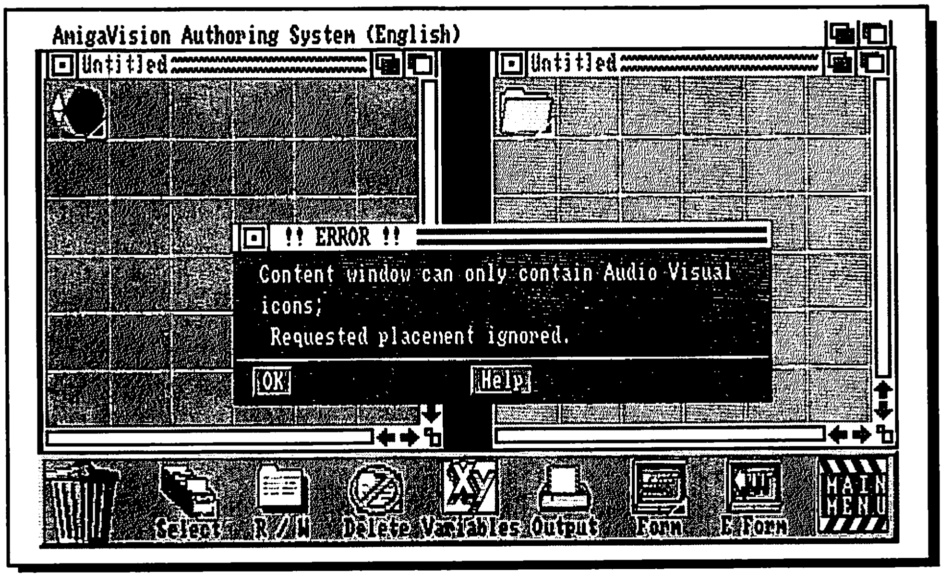


Figure 4-20
The Content window contains different icons than a Flow window



Content window can only contain Audio Visual icons;
Requested placement ignored.

Figure 4-21
The Error requester appears when an icon is illegally placed in the Content window

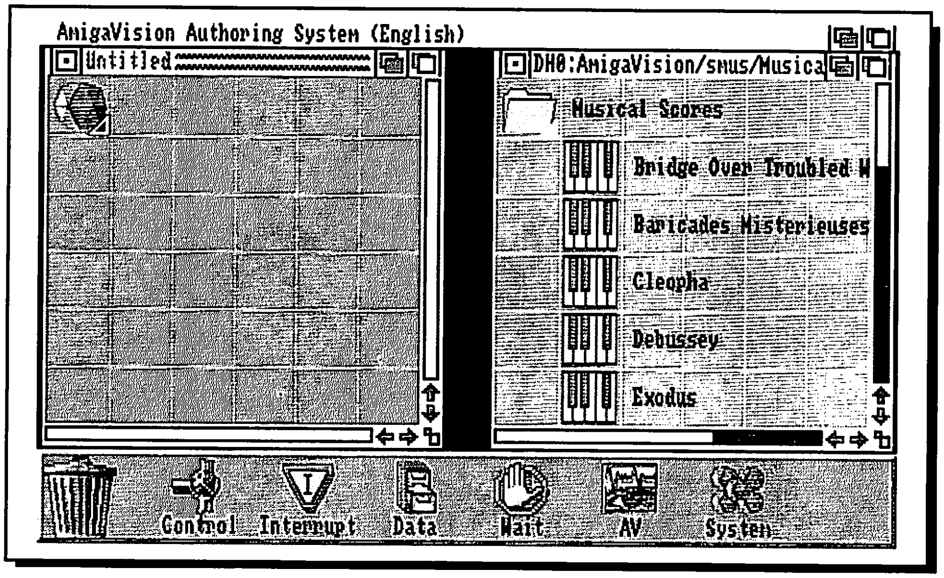


Figure 4-22
A music File folder creates easy access to a music score you will use

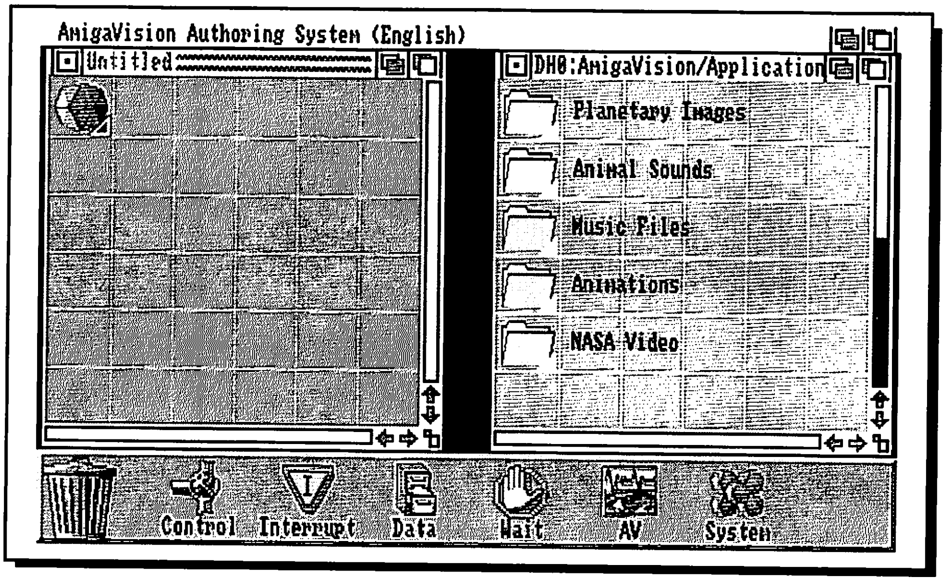


Figure 4-23
An AmigaVision File Folder library acts as a library of multimedia elements

To create multiple File folders, you can either use the Copy command from the Edit menu, or you can simply place a group of icons in a column, then use the Collect command (also from the Edit menu) and group them together. This works exactly as it does in the Flow window, except that the result of the procedure is a File Folder icon instead of a Module icon. Collected icons appear below and to the right of the File folder, in a parent/child relationship.

As mentioned, you cannot run programs out of the Content windows, but you can preview them. To see how they look just double-click on the icon and click on its Preview command button. Being able to preview within the Content window is important, because if it has been some time since you created or looked at the icons, you may forget how they appear or sound.

File folders can be organized other than just as libraries of all sounds or all animations. You can also use File folders to store audio-visual subroutines that you will use again. For example, you might want to organize them into related groups, with a picture or animation, a digitized sound track, and a text file containing documentation. Once set up, such a group could be transferred to a Flow window application by grabbing the File Folder icon and dragging it into position in the Flow window, where it would become an executable Module icon with children (see Figure 4-24). By creating routines ahead of time, you can save yourself a great deal of development time.

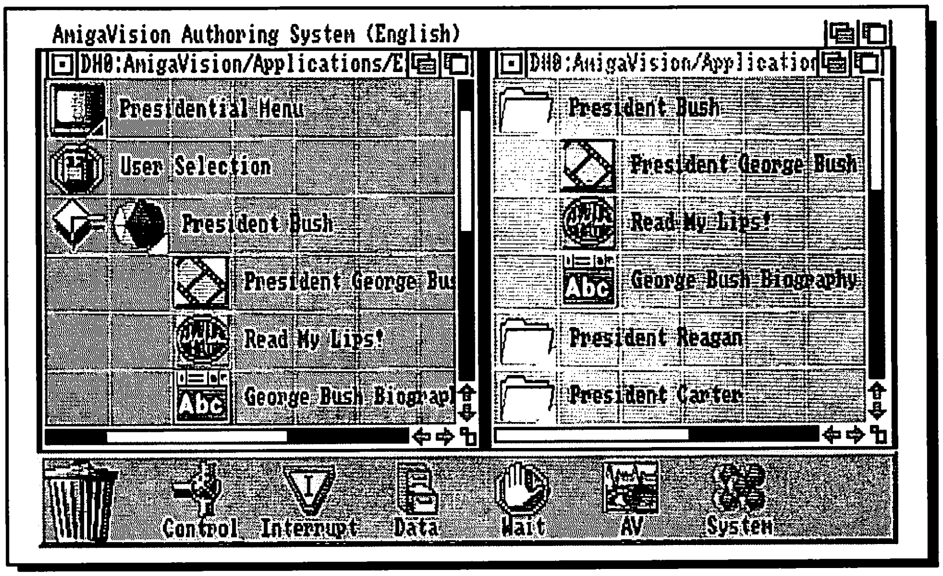


Figure 4-24
File folders become Module routines

Common Requesters

A requester is a special menu AmigaVision uses to prompt you for information it needs to fulfill your requests and commands. Virtually every icon command has its own specialized requester equipped with buttons, gadgets, and input fields. There are several requesters that appear more often than others, however, and these are worthy of special mention.

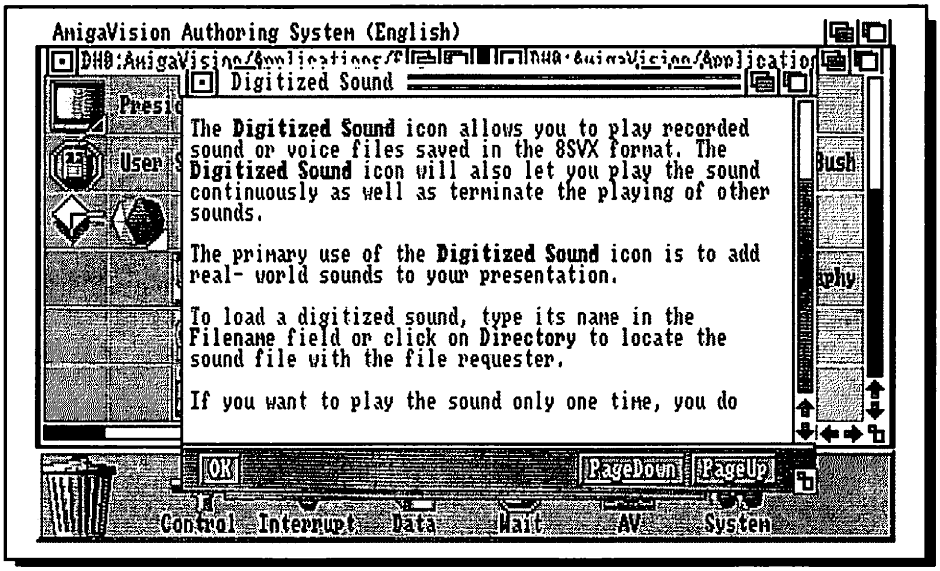
File Requester

The File requester (see Figure 4-5) is one of the most common requesters in AmigaVision. It is displayed whenever you need to load or save an application or load any of the various data files. It is accessed by clicking on the Directory button found on the many icon-specific requesters such as the Screen, Sound, and Anim requesters. (For details see the section on the Load option of the Project menu earlier in this chapter.)

Help Requester

Among the most important of all requesters is that associated with the Help button, a button that appears on nearly every requester in AmigaVision. Clicking on the Help gadget opens a special Help requester — a window that contains context-sensitive information pertaining to the task at hand. For example, if you click on the Help button in the Video requester, you will get specific information pertaining to defining the laser-disc icons and their gadgets and fields. If you select the Help gadget from the Sound icon's requester, you will get instructions on using digitized sounds. While the information available via the Help requester is not as extensive as what is found in written documentation like your AmigaVision Users Guide or here within the pages of *AmigaWorld Official AmigaVision Handbook,* it can often answer general questions that might come up.

The Help requester is actually a specialized text window (see Figure 4-25). It has features common to all Amiga windows (scroll bars and arrows, a window-close gadget, a window-resize gadget, window front and back gadgets, etc.). It also has a couple of extra features — specifically, PageUp and PageDown buttons — and an OK button. The PageUp and PageDown buttons let you scroll through the text in the window a page at a time (a page is defined as the amount of text the window can display). The OK button acts just as the window-close gadget: it closes the Help window and returns you to what you were doing when you clicked on Help.



The **Digitized Sound** icon allows you to play recorded sound or voice files saved in the 8SVX format. The **Digitized Sound** icon will also let you play the sound continuously as well as terminate the playing of other sounds.

The primary use of the **Digitized Sound** icon is to add real- world sounds to your presentation.

To load a digitized sound, type its name in the Filename field or click on **Directory** to locate the sound file with the file requester.

If you want to play the sound only one time, you do

Figure 4-25
A sample Help requester appears whenever a Help gadget is chosen

Specify Value Requester

The Specify Value requester (see Figure 4-26) is a general-purpose numeric-input requester. It is used by many command icons that require the user to input numbers. Besides a numeric keypad, it has a backspace button (<<) and a sign-change button that changes the value in the number field from positive to negative or vice versa. To the right of the keypad is a variable window, displaying the names of all variables currently defined and available to the particular icon (remember that AmigaVision has both local and global variables, and depending on where the icon is in the flow, not all variables may be available to it). To select a variable instead of entering a number, just click on the variable name and it will appear in the number field. To exit the Specify Value requester, click on OK or press the Return or Enter keys. If you are using a variable, you can also exit simply by double-clicking on the variable name. AmigaVision will then use whatever alternate value or variable you specified.

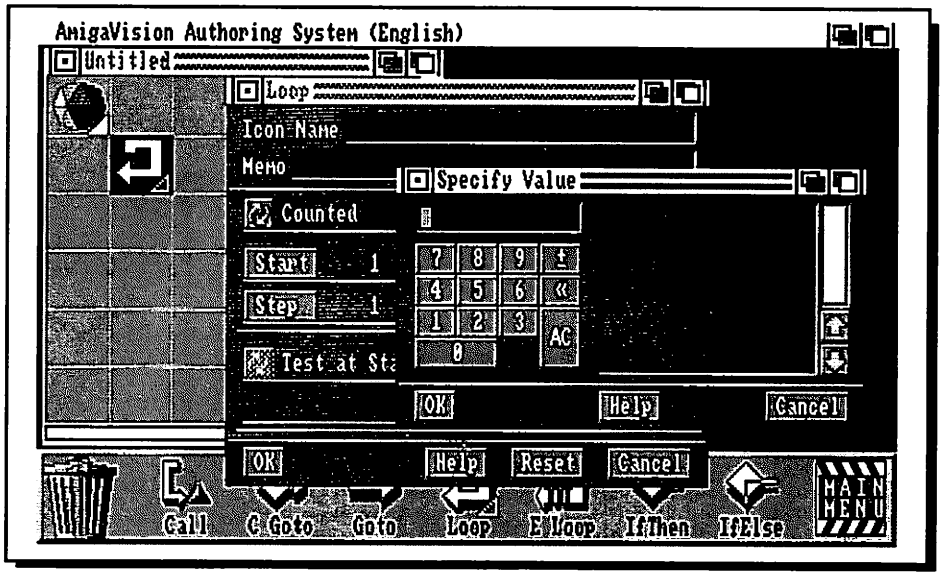


Figure 4-26
The Specify Value requester is used by many command icons requiring numerical input

Specify Variable Requester

The Specify Variable requester (see Figure 4-27) is used for the sole purpose of selecting variables from a list of currently available variables. To use it, either enter the variable name from the keyboard (by clicking in the input field, typing in the name, and pressing the Return key or clicking Enter) or click on the variable name in the window. If the list of available variable names is longer than can fit in the window, you can use the scroll arrows to move up and down through the list.

There are many other requesters in AmigaVision, but in general the rest are used specifically for a single icon. For that reason, we will postpone discussion of them until we cover their corresponding icons. The only other major requester that is used by a number of different icons is the Expression editor, which we will also detail in Chapter 13.

Gadgets

AmigaVision requesters make great use of gadgets and text-input fields (also referred to as string gadgets). There are a total of four different gadgets used in AmigaVision: Command, Check Box, Multistate, and Text.

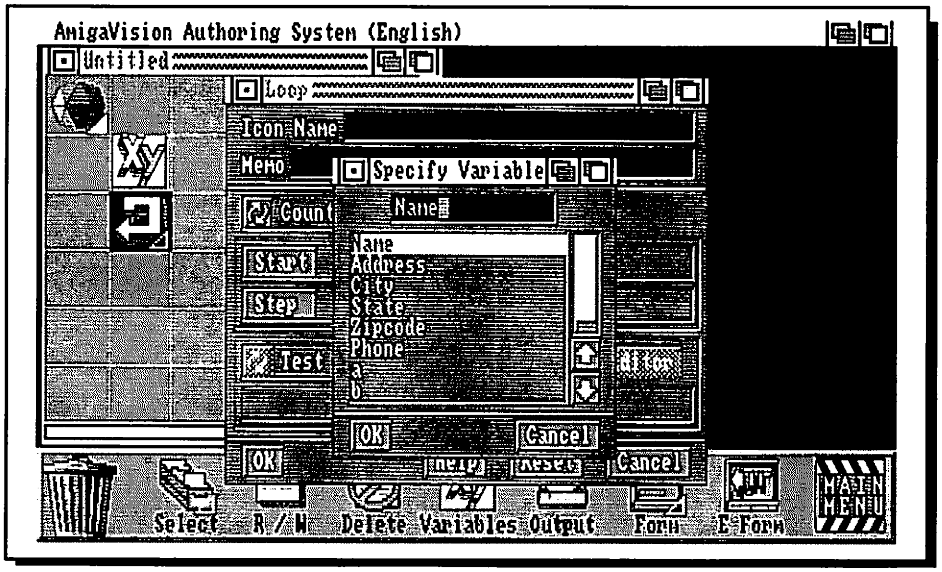


Figure 4-27
The Specify Variable requester is used to select variables from a current list of variables

Command gadgets are action-causing buttons. That means that when you click on a command gadget, you will get an instant response, usually in the form of some other requester's appearance. The most common Command gadgets are the Directory, Preview, and Help buttons.

Check Box gadgets are toggle switches. They are used to turn functions and options on and off. When the option it represents is turned on, a Check Box button contains a check mark (hence the name). When the option is off, that box is blank. Common examples of Check Box gadgets are the Pause buttons found on the requesters of the Resource icon and many of the Audio Visual icons, the Any Key button on the Keyboard requester, and the Any Click button on the Mouse requester.

Multistate gadgets are buttons used to cycle through more than two options. They are easily identified by the circular arrows they contain. Clicking on one causes one of the options to become active. Examples are the Stereo/Left Speaker/Right Speaker buttons on the Sound and Speech requesters, the Resolution, Palette, and Colors buttons on the Screen requester, the Color Cycle buttons on the Graphic requester, and the Video and Sound buttons on the Videodisc requester.

Text gadgets (also called string gadgets) are input fields used to get some text or numeric data into the requester. Common examples are the Icon Name and

Memo fields found at the top of icons, the Filename field associated with the Directory command buttons, and the Text String field in the Speech requester.

To enter text into a Text gadget, click on the text string area and then simply type characters from the keyboard. The cursor keys may be used to move left or right, and if you insert new characters into an existing string, those characters to the right of the cursor will be shifted right to make room for the new. You can also use the Backspace and Delete keys to erase characters to the left or under the cursor. When editing text in these fields you can use the Right Amiga-X (🄰X) key combination to delete everything in the field, the Shift-Left Cursor combination to move the cursor to the leftmost character in the field, and Shift-Right Cursor to move it to the end of the character string.

Icon Relationships

The icons you use in AmigaVision can exist in one of four possible relationships with other icons: Parent, Child, Sibling, and Partner. These relationships can have a direct effect on the order of execution of your program, as well as on the value of variables (local versus global). Normally, AmigaVision executes all applications in a top-down manner, starting at the first icon and working its way down. Before going from one icon to the one below it, however, AmigaVision first checks to see if any icons exist in between — to the right of the current icon. If such icons do exist, AmigaVision executes them before moving down the list. The only exception to this is when the icon to the right is a Partner icon; in this case the Partner (and its Child and Sibling icons) are executed only if the result of an expression evaluates as true.

Parent

There are ten different icons that can function as Parent icons (see Figure 4-28). Parent icons are easily identified by the presence of a triangle on the lower right of the icon. This triangle indicates that you can place Child icons underneath them, one square to the right. Child icons can be hidden from sight (telescoped) under the parent by clicking once on the Parent icon (its color will darken, indicating it has been selected) and choosing Telescope from the Edit menu. Child icons themselves can be Parent icons, and can have their own set of Child icons, creating a hierarchical icon structure.

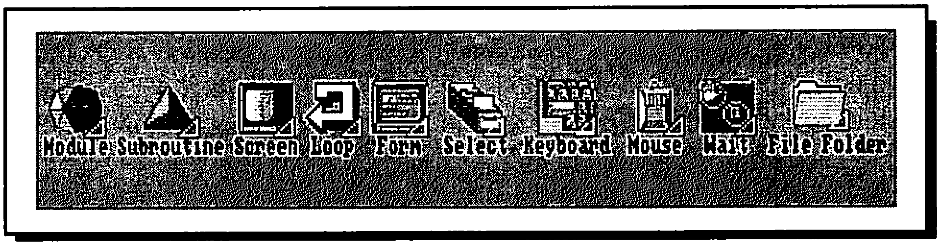


Figure 4-28
The Parent icons are easily identified by the presence of a triangle on the lower-right corner

If a Parent icon has children, the triangle it contains is a solid white; if there are no children, the triangle appears as an outline. This triangle enables you to tell at a glance if an icon contains children, even if they are hidden from sight by the Telescope function.

Parent icons have access only to variables created at their own level or higher. So, if a variable is first defined in a Child icon, neither the Parent or any higher-level icons will be aware of its existence or value.

Child

The opposite of the Parent icon is the Child icon. Child icons are a step lower than their parents in a program's hierarchical structure. Usually, when AmigaVision encounters a Parent icon, it executes any corresponding Child icons before moving on to the next Sibling icon directly below the parent.

Sibling

Sibling icons are icons that are directly above or below each other. They may each have a Partner icon or one or more Child icons. AmigaVision executes Sibling icons in a top-down manner, that is, those on top are executed before those on the bottom.

Partner

Certain icons require that a companion icon, a Partner icon, be placed directly to the right of them. There are five different icons that require partners (see Figure 4-29).

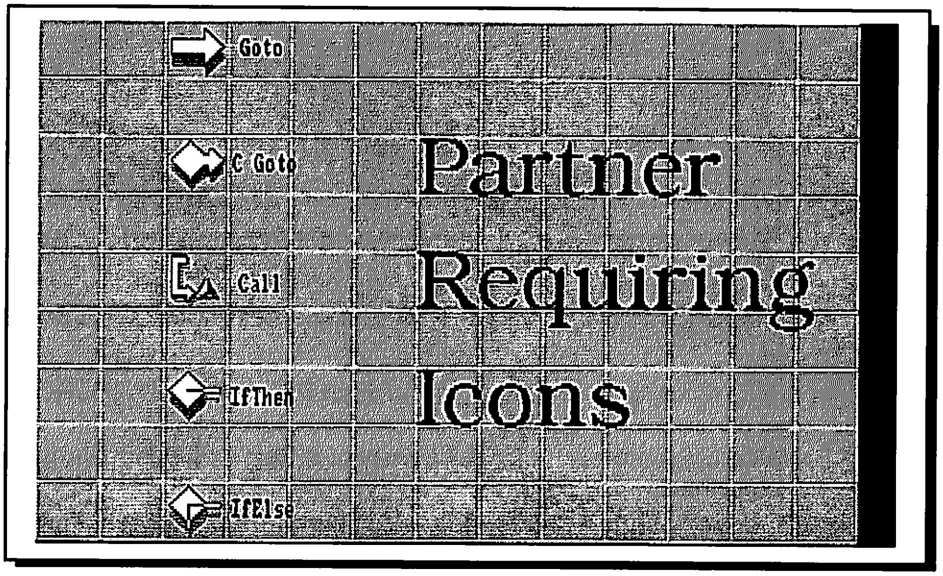


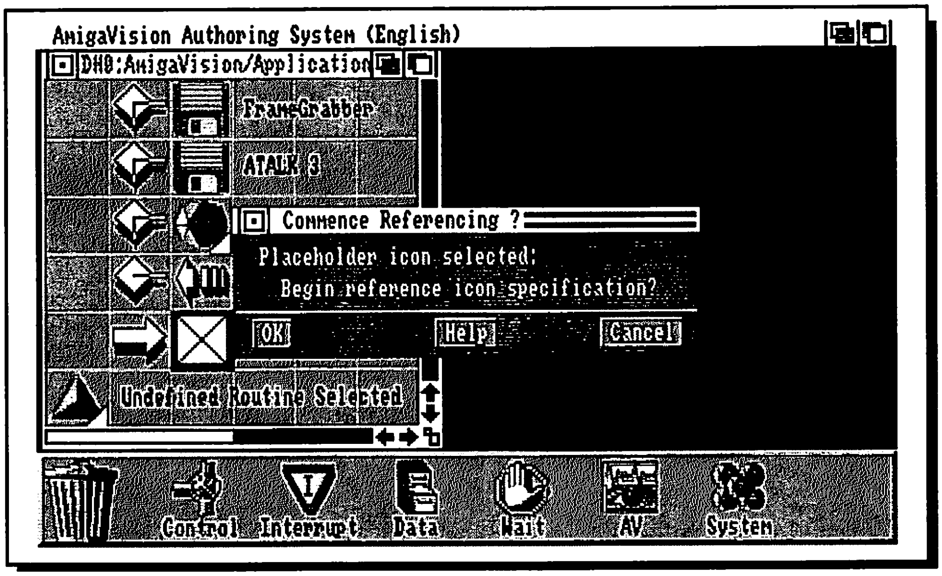
Figure 4-29
There are five Partner-requiring icons in AmigaVision

Three of these five icons, If-Then, If-Then-Else, and Conditional Goto, have an expression associated with them. Only if this expression evaluates as true in the context of that part of the application will AmigaVision execute the corresponding Partner icon.

The other two partner-requiring icons, Goto and Call, do not require any expression. Their partners are always executed.

The Goto, Conditional Goto, and Call icons each have a special partner called a Placeholder icon. These place holders represent icons in other parts of the application you want execution to branch to. When you position one of these icons in a Flow window, AmigaVision automatically places a blank white square beside it, indicating an undefined branch location.

To specify where program execution should branch to, double-click on this blank icon and you will be presented with the Commence Referencing requester (see Figure 4-30).


AmigaVision Authoring System (English)
DH0:AmigaVision/Application
FrameGrabber
ATALK 3
Undefined Routine Selected
Control
Interrupt
Data
Wait
AV
System

Commence Referencing ?

Placeholder icon selected:
Begin reference icon specification?

OK   Help   Cancel

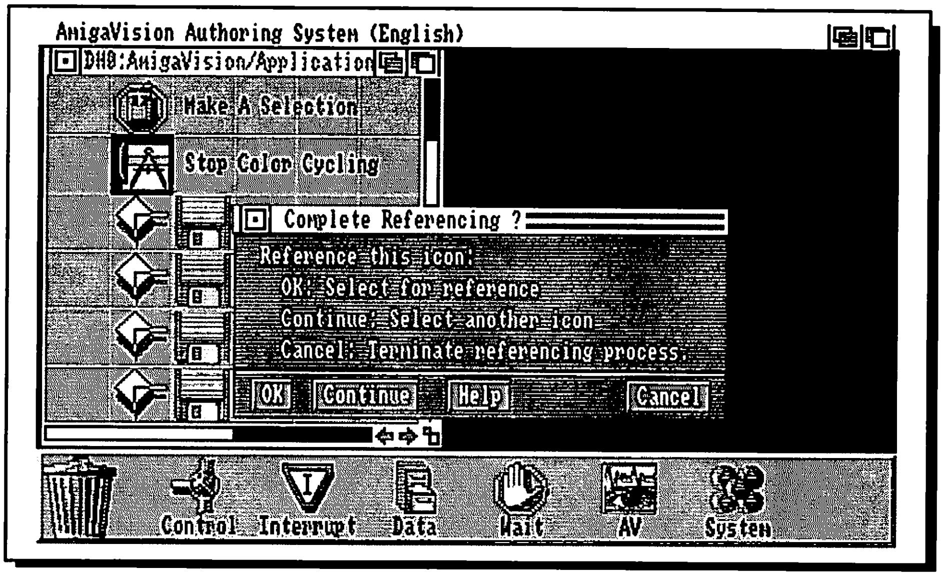
Figure 4-30
The Commence Referencing requester allows you to specify an icon to which program execution should branch

Click on the OK button if you want to continue to define the program branch; click on Cancel to abort. Then, using the scroll arrows, move through the application until you get to the icon you want to branch to. When you reach that icon, double-click on it and the Complete Referencing requester (see Figure 4-31) will appear.

To confirm that this is the icon you want, click on OK. The Flow window returns to the position in the application where the branch will occur, and an image of the destination icon appears as the partner of the original icon.

If you selected an incorrect icon in the Commence Referencing requester, click on Continue and you will be able to continue looking for the desired destination icon. If you change you mind entirely and no longer want to select a destination icon, click on Cancel and the whole process will abort.

Only one partner is allowed per icon. If you want an If-Then or If-Then-Else to perform multiple actions, use a Module icon as the partner. Then you can place as many icons as you need as children or siblings of the Module icon.



Complete Referencing ?

Reference this icon:
OK: Select for reference
Continue: Select another icon
Cancel: Terminate referencing process.

OK Continue Help Cancel

Figure 4-31

The Complete Referencing requester appears when you double-click on the icon to which you want to branch

Now that we know what the various components of the AmigaVision operating environment are and how to use them, we can begin taking a look at the process of editing AmigaVision programs. That, in fact, is the subject of our next chapter.

– 5 – Editing Your Program

AmigaVision programs are created in the Flow and Content windows, and unlike traditional computer languages, are entirely graphic in nature. When it comes to editing your applications, however, AmigaVision's editing features are very similar in concept to those you would find in a traditional programmer's text editor or word processor.

Among AmigaVision's flow-editing features are Insert mode, and the Move, Copy, and Search functions. You can have multiple windows open at the same time, and can freely copy individual icons or entire sections back and forth within the same window or between different windows. Also, you can send all or part of the flow chart to either a printer or a disk file.

All of these operations are controlled by using the mouse to point at, click on, and drag icons, and to select items from pull-down menus. If you are already an experienced mouse user, you will find editing AmigaVision applications easy and natural. And if this is your first time behind the wheel of a mouse, you should consult the *Amiga Users Guide*, which came with your computer, for instructions on using the mouse before you begin authoring AmigaVision applications.

The Basics

As mentioned in earlier chapters, when you first load AmigaVision you are presented with a screen containing AmigaVision's Main menu and an empty Flow window (see Figure 5-1).

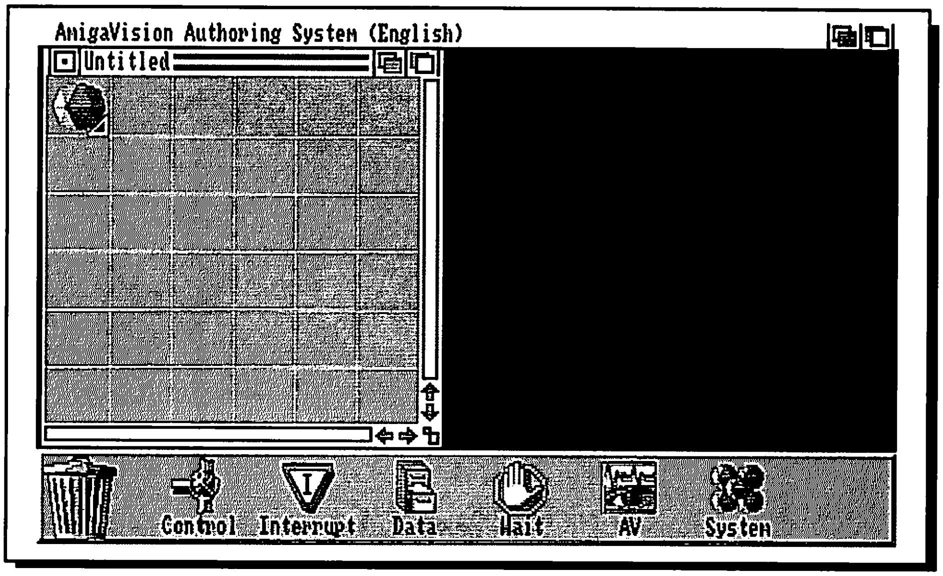


Figure 5-1
You are presented with the AmigaVision Flow window when you first load AmigaVision

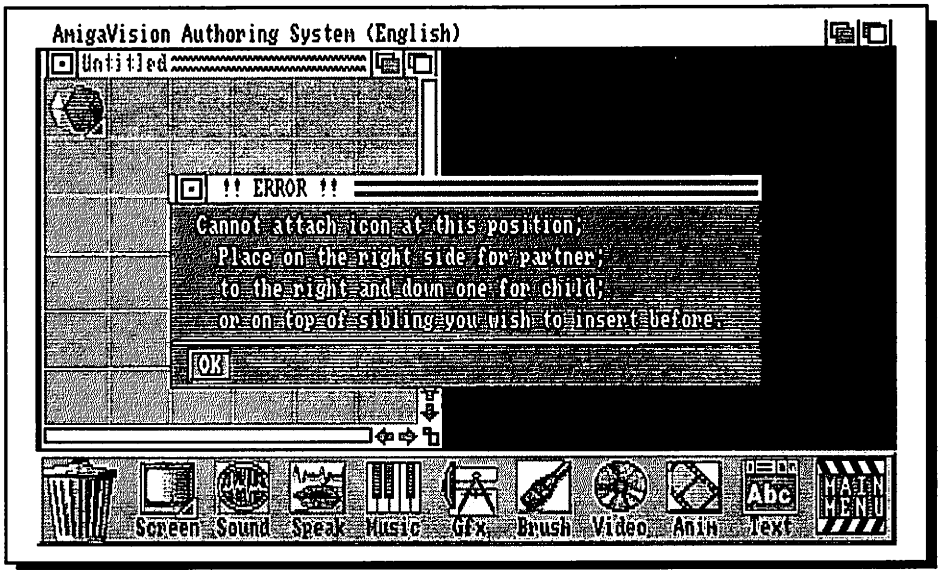
You can find icon commands only on the six submenus in AmigaVision. When you first start AmigaVision, what you see at the bottom of the screen is the Main menu. There are seven icons in this menu.

On the far left is the Trashcan icon. When you want to delete an icon, just grab it (by positioning the mouse pointer over it, and pressing and holding down the left mouse button), drag it over to the Trashcan icon, and release the mouse button to deposit it there. Once you place an icon in the Trashcan, the icon is gone and cannot be recovered. If the icon is a parent with children, the child icons are also deleted. The Trashcan icon is found on all of the AmigaVision icon menus.

The other icons are, in order of appearance from left to right, Control, Interrupt, Data, Wait, AV, and System (called Module in V1.31). None of these Main menu icons are actually commands; instead, each represents a submenu. The submenus are where the icon commands are found, and in each menu, the icons are grouped loosely by function. In the next six chapters we will examine each of these submenus and their icons in great detail.

You must place AmigaVision command icons in either the Flow or Content windows in accordance with the rules of icon relationships (see Chapter 4). This means that Sibling icons must be placed directly above or below each other in the same column, children must go below and to the right of their parents, and partners need to be directly to the right of their corresponding icons. Attempting to place an icon on the grid without following these rules will result in an error message.

To add an icon to the Flow window, move the mouse pointer to the icon command you want to use and press and hold the left mouse button. This grabs the icon, and lets you transfer it to the Flow window simply by moving the mouse. As long as you hold the left mouse button the icon will follow your mouse movements. When you release the left mouse button, the icon will also be released. If you position the icon over one of the squares in either the Flow or Content windows, the icon will be deposited at that spot when you release the button. If you release the icon anywhere outside the windows, the icon will disappear and nothing will happen. If you position it within a window in an illegal position, you will get an error message that reads: "Cannot attach icon at this position" (see Figure 5-2).



```
Cannot attach icon at this position;
  Place on the right side for partner;
  to the right and down one for child;
  or on top of sibling you wish to insert before.
```

Figure 5-2
An icon-positioning error message

Inserting Icons

Often when you are creating applications, you will find it necessary to insert an icon between two existing icons. To do this, all you need do is grab the new icon from its menu, position it on top of the icon you wish to follow the new icon, and release the mouse button. The new icon will be inserted into the icon sequence, and the icons below it in the Flow window will move down one square.

If you need to insert a whole sequence of new icons, you can position them one icon at a time using the procedure described above, or, if the new icons are coming from another location within the same or another window, you can telescope them up into a parent icon and insert just the parent into the proper position.

Moving Icons

Moving an icon from one location in your flow chart to another is also very simple. Just grab the icon, drag it to the new location, and release the mouse button. The icon (and any children it has) will be positioned in the new spot. If the new location is not in the same area of the grid as the source icon, you can make the whole flow chart scroll up or down: just move the icon to the top or bottom of the Flow window and, while holding the left mouse button down, press either the right mouse button or the keyboard's up or down cursor keys — whichever you prefer. The contents of the Flow window will scroll up or down accordingly. Once you have found the spot in your application in which you would like to deposit the icon, just release the cursor key or the right mouse button to stop the display from scrolling, and drop the icon into that position. When you do this, the icon will disappear from its old position, and the icons below that position will move up one square. At the same time, the icon will be displayed in the new location and all icons below it will move down one square to make room for it. If the icon you move is a parent, any child icons attached to it will automatically move with the parent.

Copying Icons

Copying an icon means making a duplicate of it. To copy an icon you need to select Copy from the Edit pull-down menu. When enabled, the Copy option has a check mark next to its name in the Edit menu and the mouse pointer appears to have a shadow.

When in Copy mode, grabbing and moving an icon to a new location causes the icon to be duplicated in the new position. Unlike moving an icon, the original icon stays in place and is not deleted.

If you copy an icon from one Flow or Content window to another, you do not need to turn on Copy mode. In these cases just grabbing and moving the icon from one window to another is treated as a copy, with a duplicate being deposited at the destination, and the original icon left in position. (For more information on the Copy feature, see Chapter 4.)

Search Icons

If you have given names to the icons in your flow chart (and it is a very good practice to do so), you can use the Search feature from the Edit pull-down menu to find a specific icon. As your experience with AmigaVision grows, you will create more and more complex and detailed applications, sometimes nested many levels deep. It is with these more complex programs that the Search feature becomes essential.

To use, select Search from the Edit menu. Enter the name of the icon (or as much as you remember), using wildcards if necessary. If the name exists the Search function will find it. If the icon has been hidden by the telescope function, Search will find it and open (telescope) the parent icons to display the located icon. (For more information on the Search feature see Chapter 4.)

Collecting Icons

Grouping sibling icons together under a new parent Module icon can be done with the Collect feature found in the Edit menu. When you choose Collect from the Edit menu, a check mark will appear next to the option's name in the menu and a small box will appear on the right edge of the mouse pointer. With the mouse pointer, point to the icons you want to collect and press the left mouse button. A rectangle will outline those icons, and when you release the mouse button, a Module icon will appear. The collected icons will act as children of the Module icon, and will appear below and to the right of the Module icon on the flow chart. (For more information on the Collect function, see Chapter 4.)

These, then, are the basic editing features of AmigaVision. Using only a mouse, you can create complex applications by dragging the needed icons into position on the flow chart. Altering and editing is also done with the mouse in much the same manner as it is done in modern word processors.

At the lowest level, editing AmigaVision programs concerns the definition of the specific icons. This is done by double-clicking on each icon, which brings up a requestor with the gadgets and input fields that supply the command with the information needed to properly function.

In the next six chapters, we will look at each of the six icon submenus, and all the commands found in each. We will also carefully examine the workings of each of the commands that make AmigaVision a truly wonderful programming language.

Section Two

Command Reference

The Control Command Menu

The Interrupt Command Menu

The Database Command Menu

The Wait Command Menu

The Audio Visual Command Menu

The System Command Menu

– 6 – The Control Command Menu

We begin our study of the AmigaVision command icons in the Control menu. You can access this menu from the Main menu by clicking on the Control icon — the first icon to the right of the ever-present Trashcan icon.

The Control icons (see Figure 6-1) are icons that can command the program flow to change — by jumping from one position in the program to another, or enabling select sections of the code to be executed under conditional program control. It is these program flow-control commands that allow you to overrule the normal top-down execution of the program.

There are three basic types of Control icons. The first type is used to branch to another location in the flow chart, the second type is used for conditional decision making (with one or more icons being executed if a logical expression is determined to be true), and the third is used to make the program loop within a section of the flow chart.

Branching Icons

Branching icons include the Call, Conditional Goto, and Goto control statements. Both Goto and Conditional Goto let you branch to a new location in the flow chart, from which program execution will continue just as if the new location had immediately followed the branching command. The Call command enables you to jump to the beginning of a subroutine, which is a collection of commands that are separate from the main program. A subroutine is usually a section of program code that will be executed a number of times from different locations in the program. By placing commands in a subroutine and by calling the group instead of repeating the same code in several locations in the program, you achieve an

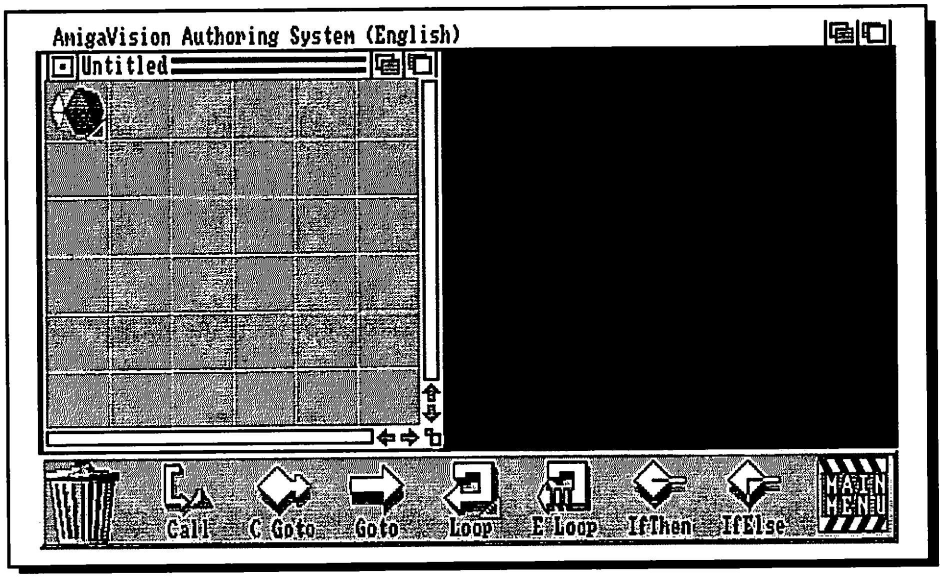


Figure 6-1
The Control icons determine the direction of program flow

overall reduction in the size of the program as well as an increase in the program's modular structure and its readability. When a subroutine is called, program flow switches to the subroutine. When the subroutine finishes, program execution returns to the command icon immediately following the icon that summoned it.

Call

The Call icon (see Figure 6-2) is used to branch program execution from the main body of the program to a subroutine. When the subroutine finishes executing, either by reaching its end or by encountering a Return icon, program execution returns to the main body of the program and continues with the Sibling icon immediately following the initial Call icon.

The Call icon requires a Partner or companion icon in order to function. The partner of a Call icon is not a standard command icon, however — instead it is merely a reference to the Subroutine icon you want to branch to.

To use the Call icon, drag it into the Flow window and place it in the appropriate position in your program. As soon as you release it, a white square, representing the partner of the Call icon, appears. This square is the Placeholder icon. It acts as a dummy argument and remains blank until you explicitly define it (see Figure 6-3).

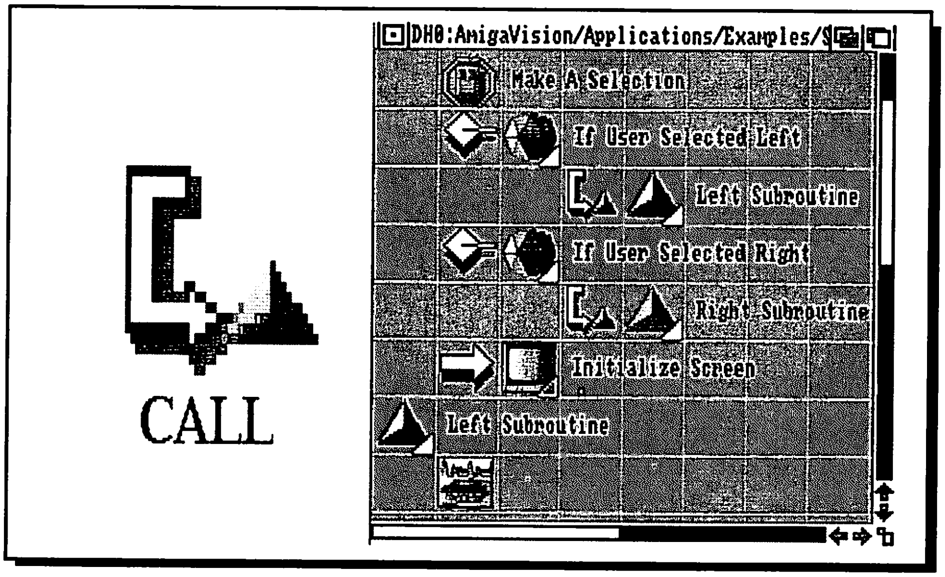


Figure 6-2
The Call icon in an application

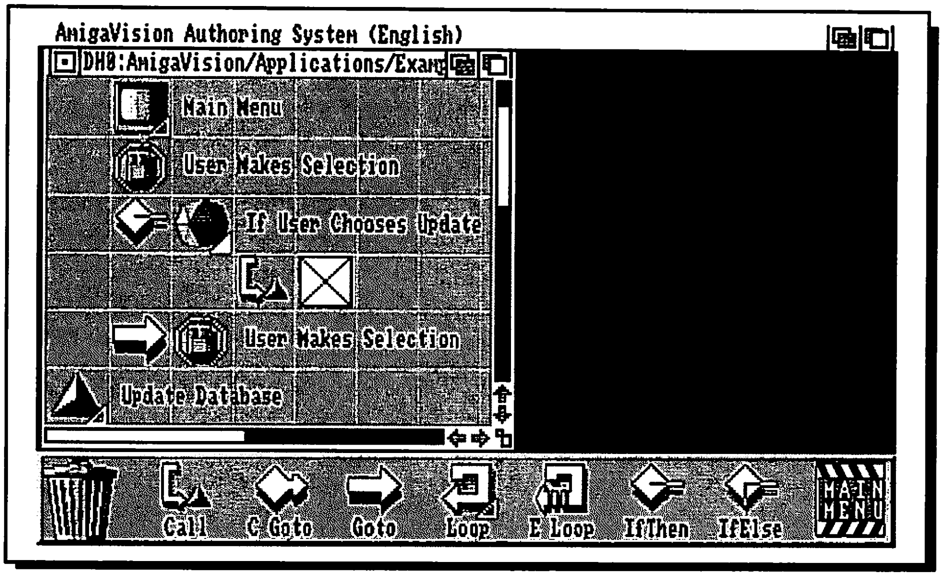
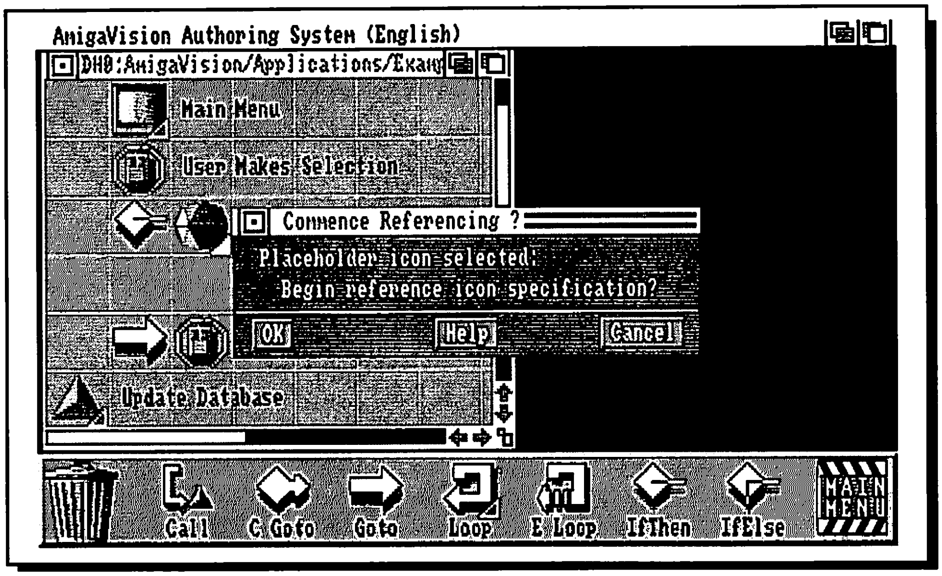


Figure 6-3
The Placeholder icon remains blank until you explicitly define it

Unlike most AmigaVision icons, the Call icon has no specific requester of its own. In fact, if you double-click directly on the Call icon you will get a message requester stating, "The Call icon has no requester: it branches to the event positioned beside it."

Defining the placeholder involves specifying exactly which Subroutine icon you want the Call icon to branch to. To begin, double-click on the white Placeholder icon. When you do this, AmigaVision places a requester on the display with the message "Commence Referencing?" (see Figure 6-4). If you want to continue to define the Placeholder icon click on OK; otherwise, click on Cancel to abort the process or on Help to get more information about defining placeholders.


AmigaVision Authoring System (English)

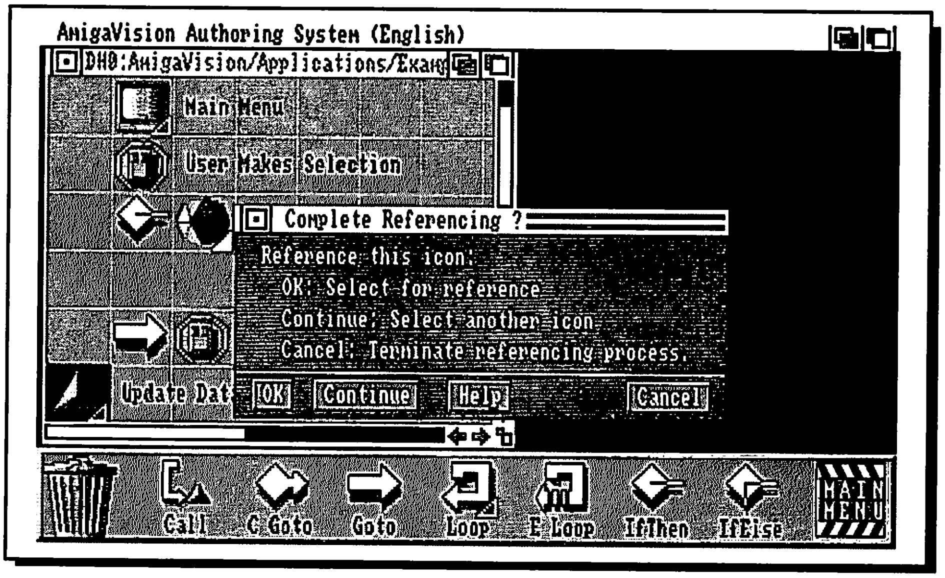
DH0:AmigaVision/Applications/Exam

Commence Referencing ?

Placeholder icon selected:
Begin reference icon specification?

Figure 6-4
Commence Referencing requester appears when the Placeholder icon is double-clicked

After clicking on OK, move your mouse pointer to the Subroutine icon you want (you may need to scroll through the program to find it), and double-click on it. (The Subroutine icon appears as a three-dimensional embossed triangle.) Another requester will appear, this one asking "Complete Referencing?" (see Figure 6-5). If you are sure you want to choose this Subroutine icon as the partner of the Call icon, click on OK. If you do not want the subroutine you selected, you can either select Continue, which allows you to continue on and select another subroutine, or Cancel, which aborts the icon-referencing procedure altogether.



Complete Referencing ?

Reference this icon:
OK: Select for reference
Continue: Select another icon
Cancel: Terminate referencing process.

OK | Continue | Help | Cancel

Figure 6-5
The Complete Referencing requester appears when a Subroutine icon is double-clicked

When you finish selecting a subroutine, two things happen. First, you return to the position in the Flow window where you started — which is the location of the Call icon. Second, the white Placeholder icon disappears and the Subroutine icon replaces it. If the Subroutine icon you selected has a name (it should if you are following proper AmigaVision programming practices), the Call icon's partner will display that name to its right. This makes it very easy to know which subroutine the Call icon is accessing (see Figure 6-6).

The Call icon can reference only a Subroutine icon, and you cannot substitute any other icon for it. If you do try to select some other icon, you will get a requester informing you that you have made a referencing error (see Figure 6-7).

Keep in mind that once you have jumped to a subroutine via the Call icon command, there are two ways to exit the subroutine and return to the location in the main program. The first is for the program to execute a Return icon. The Return icon, found in the System icon menu, works just as the RETURN command found in the traditional BASIC computer language. You can position the Return icon in the flow chart as a simple sibling icon (that is, as a single icon) or as the partner of another icon such as the If-Then icon.

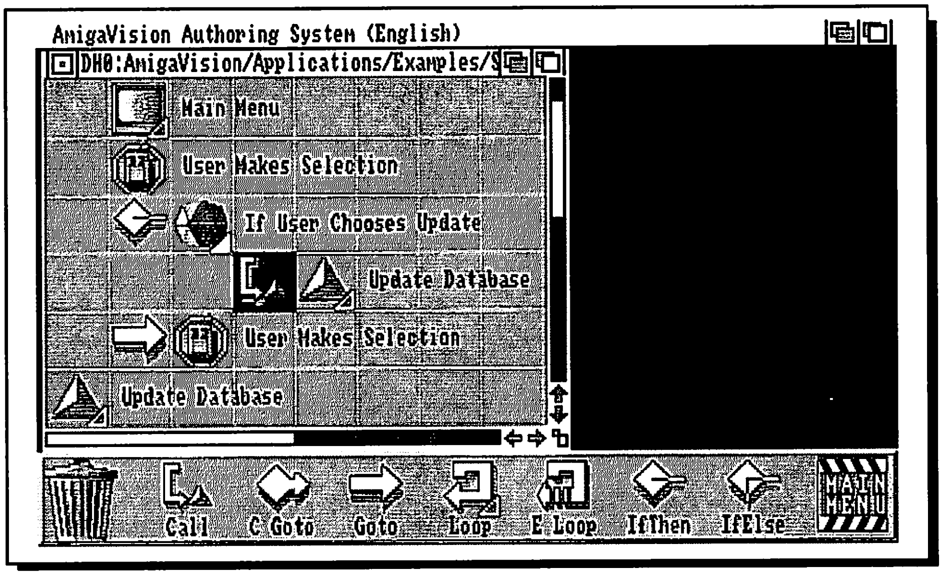
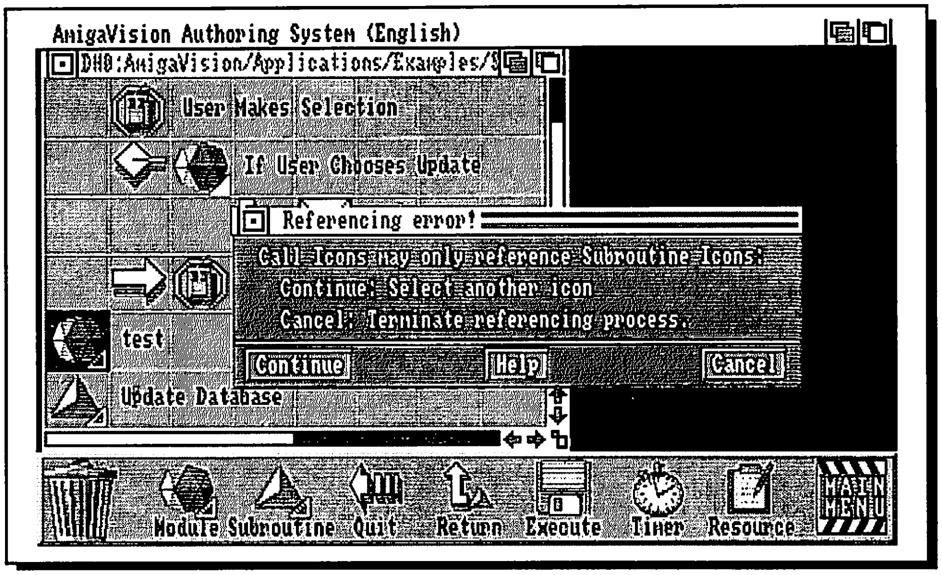


Figure 6-6
The defined Call icon



Referencing error!
Call Icons may only reference Subroutine Icons:
Continue: Select another icon
Cancel: Terminate referencing process.
Continue Help Cancel

Figure 6-7
The Referencing Error requester

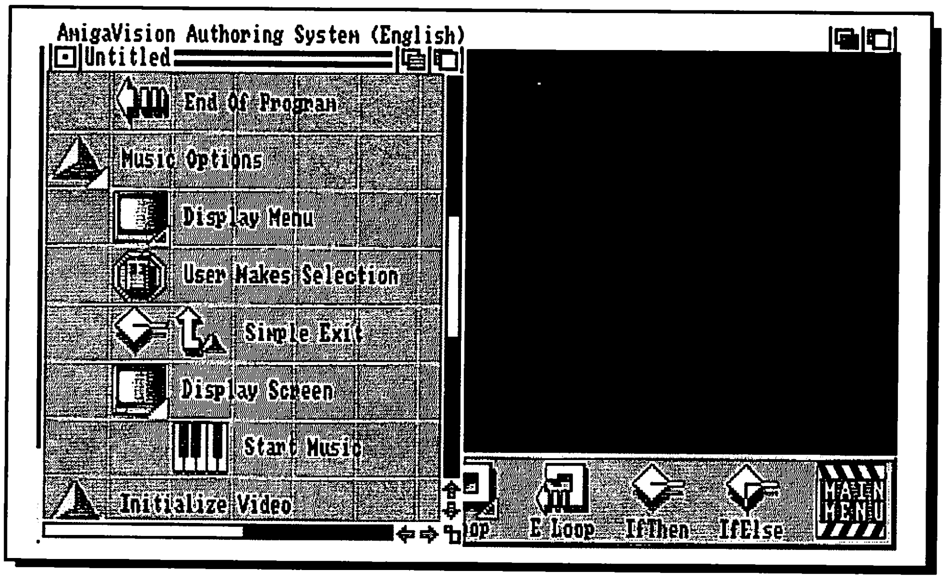


Figure 6-8
Multiple subroutine exits

The second method of exiting a subroutine is simply to finish it by executing the last icon in the routine. Figure 6-8 shows a subroutine with both types of exits available. The first exit can occur if the result of the expression within the If-Then icon evaluates as true. If this happens, the partner, which is the Return icon, is also executed. If the expression in the If-Then icon evaluates as false, the partner is ignored and the next sibling icon is executed. In this case, the sibling is a Screen icon, which displays another screen and has a Music icon as a child. After the new screen is displayed and the music has started, AmigaVision attempts to execute the next sibling in the subroutine. Finding none, it knows that the subroutine has ended. At that point program execution returns to the main program, starting with the Sibling icon immediately following the Call icon.

Goto

The Goto icon is an unconditional branching command, meaning that the action it represents always occurs. Goto allows you to branch to another section of the program and thus skip over a section of the program that you do not want executed. Because it is a branching icon it requires a Reference icon as a partner, just as the Call icon does. When executed, the Goto icon causes a jump to another position within the program and continues execution from the icon at that point.

Unlike the Call icon, there is no automatic returning from the branch with the Goto icon. Instead, program flow just continues onward from the point that you jump to. Like the Call icon, however, the Goto icon has no specific requester of its own. If you were to double-click directly on the Goto icon, you would get a User Information message requester stating "The Goto icon has no requester: it branches to the event positioned beside it."

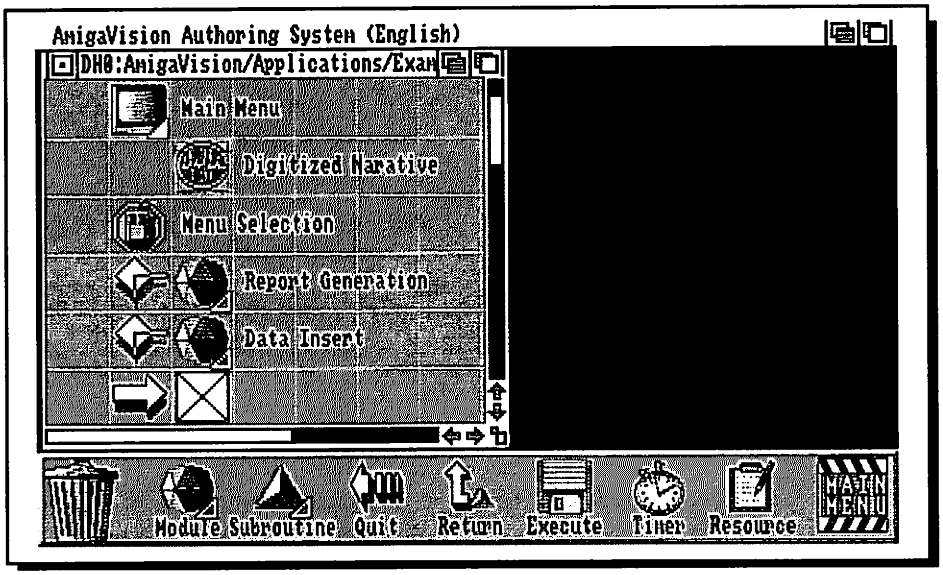


Figure 6-9
The Goto icon with blank Reference icon

To use the Goto icon, position it in the Flow window. The blank Placeholder icon will appear, and will remain blank until you explicitly define it (see Figure 6-9). The definition process works exactly as it does with the Call icon. Double-click on the reference icon and you will see the Commence Referencing requester (see Figure 6-4). Scroll through the Flow window until you find the icon you want to branch to and click on it. When the Complete Referencing requester (see Figure 6-5) appears, you can click OK to accept the selected destination icon, choose Continue to select a different icon, or select Cancel to abort the process entirely. Once you have accepted an icon, you automatically return to the Goto icon and the blank Placeholder icon changes to a reference duplicate of the icon at the branch destination (see Figure 6-10).

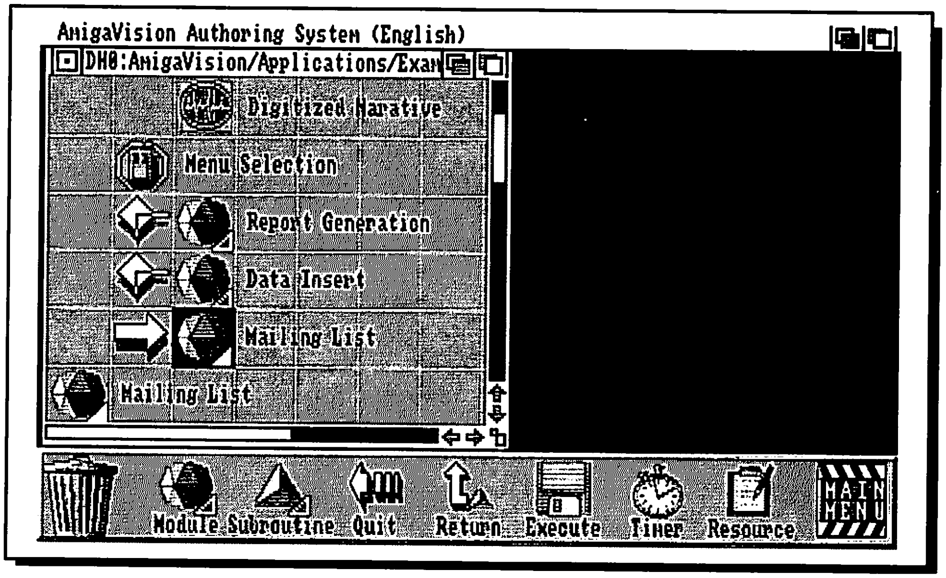


Figure 6-10
A defined Goto Reference icon

The Goto icon is different from the Call icon in several ways. First, as was already mentioned, the Goto branch causes a permanent jump, meaning that program control does not return or change unless the program encounters another branch-control statement. Also, unlike the Call icon, which can branch only to a Subroutine icon, the Goto icon can branch to any type of icon.

Goto does have some limitations. For one thing, it cannot branch to itself (which is a very good thing, for if that happened the program would freeze up). More importantly, a Goto icon can branch only to other siblings, and to its own parent and grandparent icons. This means the Goto icon cannot branch to any icon that is positioned to the right of it on the flow chart grid. That precludes children of sibling icons, their children, and so on. Also, you cannot branch to the children of a Loop, Subroutine, Select Record, or Interrupt icon unless the Goto itself is a sibling of these child icons. If you wish, however, you can branch to the parent Loop, Interrupt, or Select Record icons. You can also branch directly to a Subroutine icon, although this is a very bad programming practice: You would not be able to exit the Subroutine except by another branch statement.

The ability to perform unconditional branching to any type of icon makes the Goto icon command a very powerful feature of AmigaVision. But be warned —

overuse of the Goto icon is not a good practice. Programs that rely on excessive use of the Goto command can quickly become "spaghetti code" — program code that has no clearly defined structure. This form of program is very difficult to debug (to find and remove errors and problems), and even harder to understand. Keep in mind that proper AmigaVision authoring is an exercise in structured programming. Try to keep your program well ordered and defined, making full use of the power of subroutines (accessed with the Call icon) wherever possible.

Conditional Goto

The Conditional Goto command is similar to the regular Goto command in that it can permanently change the point of program execution from one position in the program to another by branching (jumping) to the new position. Unlike Goto, however, which is an unconditional branch, in order for the branch to occur with a Conditional Goto icon, the command must include some expression that will evaluate as true. When this expression is true, the Partner icon will be executed, meaning that the branch to the other location will occur. On the other hand, if the expression evaluates as false, branching will not occur and the next Sibling icon beneath the Conditional Goto will be executed.

Like the other branching icons (Call and Goto), Conditional Goto uses a reference icon as a partner. Until you explicitly define it, the Reference icon remains a blank Placeholder icon. Defining where the branch should go is performed exactly as it is done elsewhere: just double-click on the reference icon and then click on the destination icon. All the requesters and options involved in defining the reference icon for the Conditional Goto command work as described for the Call and Goto icons.

While defining the reference icon here is much the same as doing so for the Goto icon, a difference lies in the conditionality of this command. If you double-click on the Conditional Goto icon you will find that the Expression Editor requester (see Figure 6-11) appears. The Expression editor is an extraordinarily powerful requester that contains a wealth of commands and options and is used by many icons in AmigaVision. Its features will be discussed in depth in Chapter 13.

Using the Expression editor you can define some logical or mathematical expression (equation) that will be evaluated when AmigaVision reaches the corresponding Conditional Goto icon. If the expression evaluates as true the branch will occur, if not, no branch will occur.

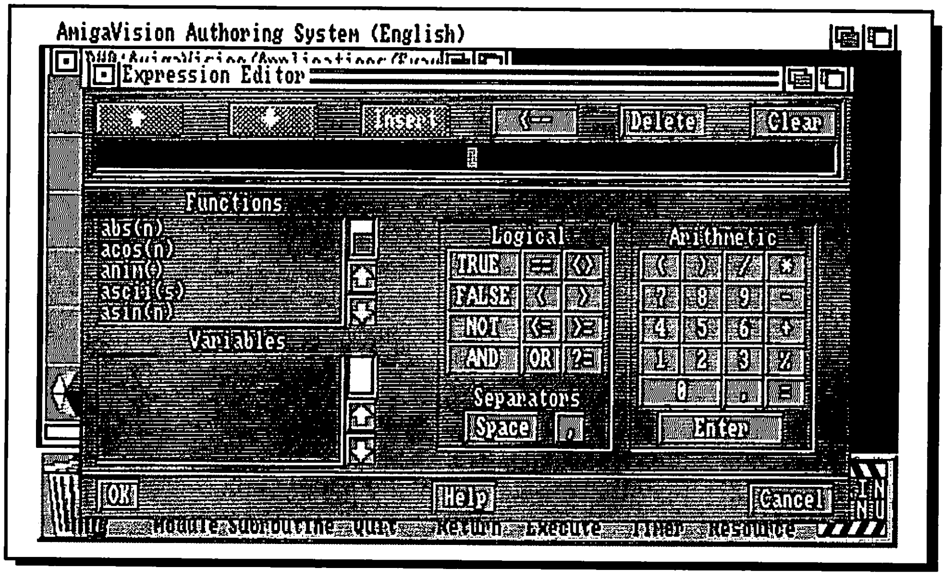


Figure 6-11

The Expression Editor requester appears when the Conditional Goto icon is double-clicked

At this point you might be wondering what kind of expressions are possible, and if this whole concept is new to you, you might also wonder why you would want to do this in the first place.

The types of expressions you can use are almost boundless. They can include any variable you have created. Consider the following examples:

X == 10

Y == COS(Z)

Cost > 25.50

NumberOfLives == 0

User == "John Smith"

Notice the use of the double equal sign (==). This is not used as an assignment of value as X = 10 is. In other words, X == 10 does not give the variable X the value of 10. Instead, it asks the question "Does X equal 10?" If you use this expression in the Conditional Goto icon, AmigaVision will ask that question when it evaluates the Conditional Goto. If X does indeed equal 10 at that moment, the expression will evaluate as true and the program will branch to the reference icon. If X does not equal 10 then no branch will occur.

Variables aren't the only things you can include in an expression. The Expression editor has many functions that you will find very useful. Here are some examples of other powerful uses:

```
Anim() == 45
Video() => 5100
Response() == "Dracula"
```

The Anim() function is used to return the number of the current frame of an animation when it is being played by the AmigaVision Anim icon. The Video() function returns the number of the current frame of video from a laser-disc player. By using these functions in your programs you can monitor the events as they occur, then perform some action when you reach the desired frame. The Response() function is especially powerful. It can return a response, according to your specifications in the Object editor, to identify which hot spot the user selected. In this case, the user could respond either by clicking the left mouse button or by pressing a key.

We will discuss the functions and features of the Expression editor in detail later. For now, though, you have an idea of what a logical expression is and how you can put it to use. This concept will be coming up again and again, so if you are new to programming and find it a little difficult, just keep studying this section and it will become clearer.

To add an expression to the Conditional Goto command, double-click on the icon. When the Expression Editor requester appears, type in the expression you want from the keyboard or click on the editing gadgets with the mouse to define the expression (see Figure 6-12). The Expression editor contains a large number of functions as well as a list of any variables you have defined that are available from the current position in the program. There are also a number of Boolean, or logical, functions you can use, and a complete mouse-controlled numeric keypad. Once you have entered the expression correctly, press the Return key or click on the OK gadget. The requester will disappear and the Conditional Goto command will then be evaluated using your expression. If you have made an error (such as a syntax error) in formulating your expression, AmigaVision will inform you of it by displaying an error requester. At that point you should reenter the expression editor and correct your mistake (see Figure 6-13).

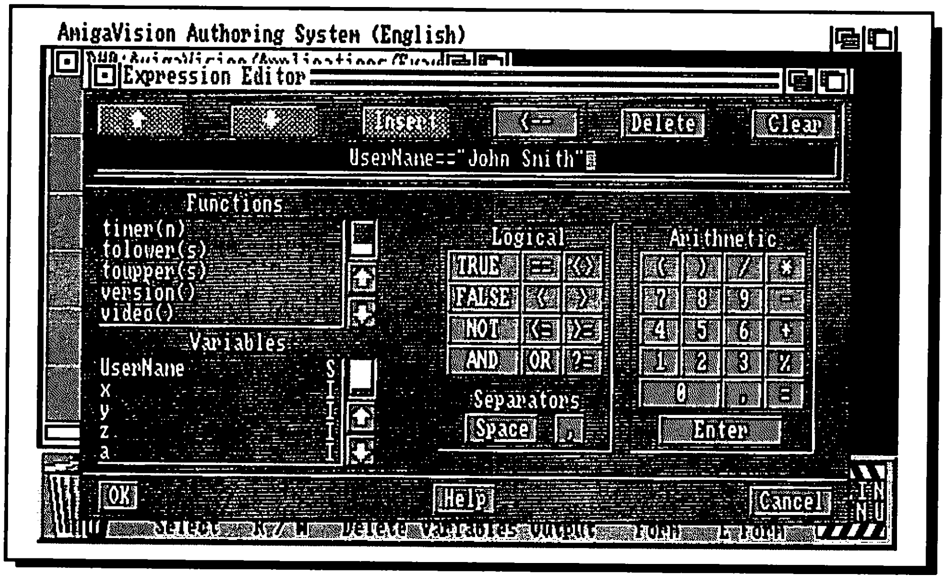
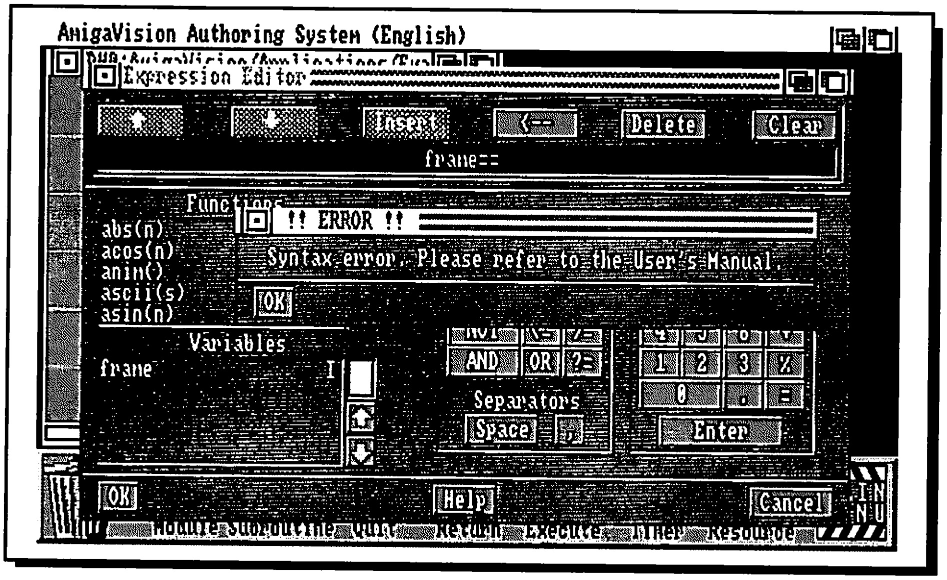


Figure 6-12
Defining your expression with the Expression editor



!! ERROR !!

Syntax error. Please refer to the User's Manual.

OK

Figure 6-13
The Expression Editor Syntax Error requester

As with the Goto icon, the Conditional Goto icon can branch to any type of icon, but it cannot branch to either itself or to icons positioned to the right of it on the flow chart. Also, just as with the Goto icon, you cannot use the Conditional Goto command to branch to the children of Loop, Subroutine, Select Record, or Interrupt icons unless the Conditional Goto itself is a sibling of these child icons. And, just as with Goto, you can branch to the parent Loop, Interrupt, or Select Record icons. You can also branch directly to a Subroutine icon, although this is a very, very bad programming practice: You would not be able to exit the Subroutine except by another branch statement.

Conditional Icons

Conditional icons are used to evaluate expressions. These include the If-Then and If-Then-Else commands, both of which must have a Partner (companion) icon associated with them. These commands require some form of logical expression to be defined, and if the expression evaluates as true, the Partner icon is executed. The partner can be a single command icon or a Module icon with a multitude of children. Using Modules, you can set up very complex routines to be conditionally executed. Conditional Goto allows conditional program flow branches.

If-Then

The If-Then icon is a decision-making command icon that requires a partner. It is used to evaluate a user-defined expression and then to take some conditional action. If AmigaVision determines that the results of the expression are true, the partner of the If-Then icon is executed. If the results are false, then the partner is ignored and AmigaVision continues on to the next Sibling icon in the flow chart. The If-Then command can be read like this:

IF <expression> is true THEN execute <partner icon>

To use it, place the If-Then icon into position in the flow chart. Its Partner icon can be any icon except another one that also requires a partner (meaning the partner cannot be an If-Then-Else, Goto, Conditional Goto, or Call icon, or another If-Then icon).

To define the expression that the If-Then icon will use, double-click on the icon. AmigaVision will respond by displaying the Expression Editor requester. Using the mouse or keyboard, enter the expression you want to use as the condition to be evaluated, and then either press the Return key or click on OK. When AmigaVision executes the If-Then icon, it will evaluate the expression you define here. If the result is true, the Partner icon will be executed; if false, the partner will be ignored. If you make a syntax error in entering the expression, AmigaVision will display an Error requester to tell you that a mistake has been made.

Although the partner of the If-Then icon cannot have any siblings, it can have its own children. For example, it is perfectly legal to use a Screen icon as the partner, and then place another Audio Visual icon as a child of the Screen icon (see Figure 6-14).

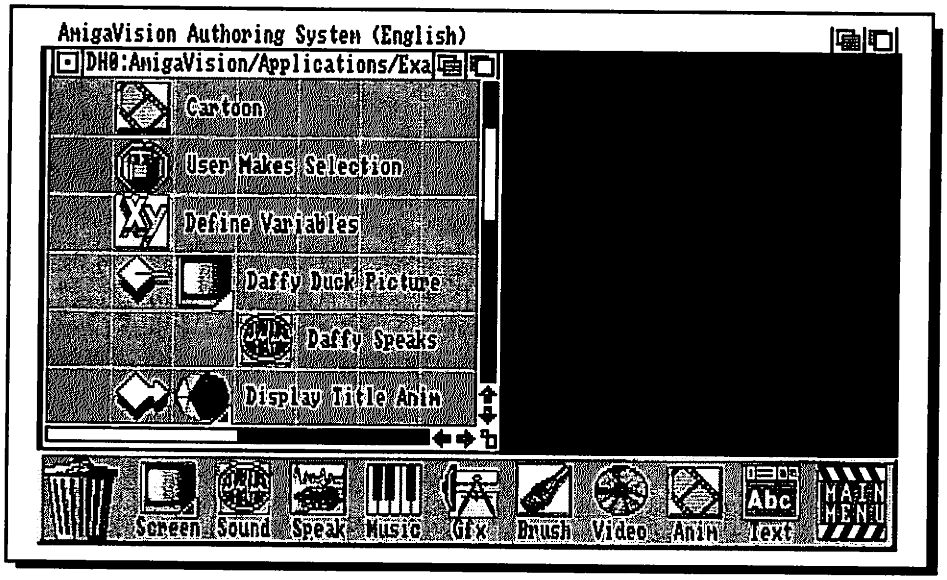


Figure 6-14
If-Then icon and partner with a child

For even more complex conditional statements, use a Module icon as the partner of the If-Then statement. Then, place any number of icons (of as many types as you wish, including parents of yet more icons) as children of the Module icon. AmigaVision will then execute all these if the If-Then evaluates as true (see Figure 6-15).

If-Then-Else

The If-Then-Else icon is a more powerful version of the If-Then command. This conditional decision-making icon is used to decide what icons to execute based on a user-defined logical expression. It allows selective execution, with one icon being executed when the expression is true, and another when the expression is false.

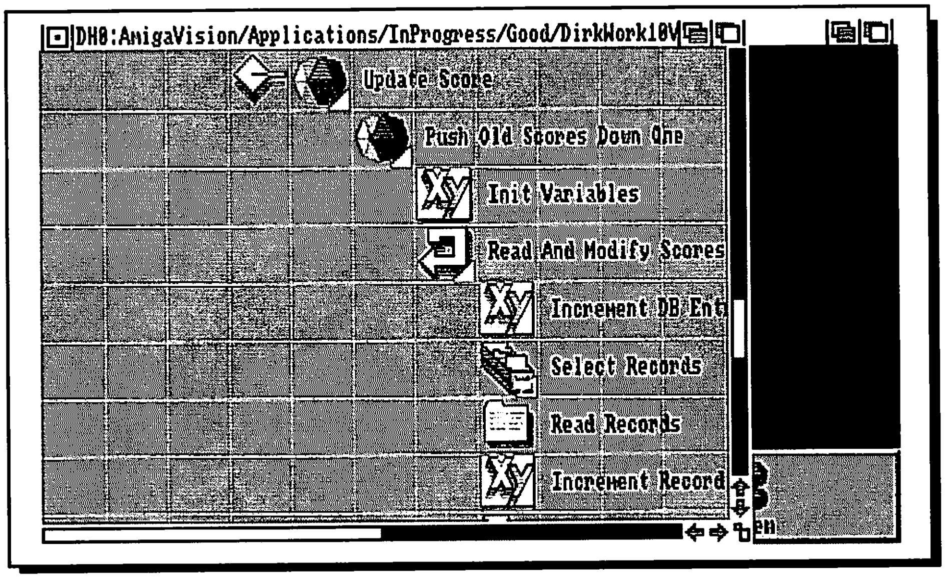


Figure 6-15
Complex Module routine as partner to an If-Then icon command

Do not confuse this with the way the If-Then icon works. The If-Then process decides, based on the defined expression, if its partner should be executed. When the expression is true, the partner is executed, as is the next Sibling icon. If the expression is false, the partner is ignored but the next sibling is still executed. Regardless of whether the expression proves true or false, If-Then causes AmigaVision to execute the next sibling.

If-Then-Else, on the other hand, does not always cause execution of the Sibling icons. If the defined expression proves to be true, the partner is executed and the next Sibling icon is skipped. If the expression evaluates as false, the partner is skipped and the next Sibling icon is executed. This difference makes for a very important distinction between the If-Then and If-Then-Else icons.

The If-Then-Else icon is defined in exactly the same manner as the If-Then command. Double-clicking on the icon opens the Expression editor, and the logical expression is entered via the mouse or keyboard.

As far as Partner icons go, the If-Then-Else icon has the same limitations as the If-Then command. The partner cannot have siblings, and it cannot require another partner, although it can support children. The most common type of partner for it is the Module icon, although you can use other Parent icons, including Screen, Loop, Select Record, and Form.

One of the most powerful aspects of the If-Then-Else command is the flexibility it offers in allowing you to build more complex decision structures by combining several If-Then-Else icons as siblings. As an example, look at Figure 6-16, which is a video browser of the film *Who Framed Roger Rabbit?* (1988, Touchstone Pictures and Amblin Entertainment). Here we have three If-Then-Else icons and one If-Then icon. Depending on which option the user selects, the program will execute one of the four sequences. If the user chooses "Filming A Cartoon," the laser disc will display that sequence, and skip all three of the following If-Then commands. If the user selects "Donald & Daffy," the program checks the first If-Then-Else, executes the second, and skips the next two. If the selected sequence is "Jessica Sings," AmigaVision checks the first two If-Then-Else commands, executes the third, and skips the last If-Then command. Finally, if the user picks "Special Effects," the first three If-Then-Else commands are checked and bypassed and only the last command executes.

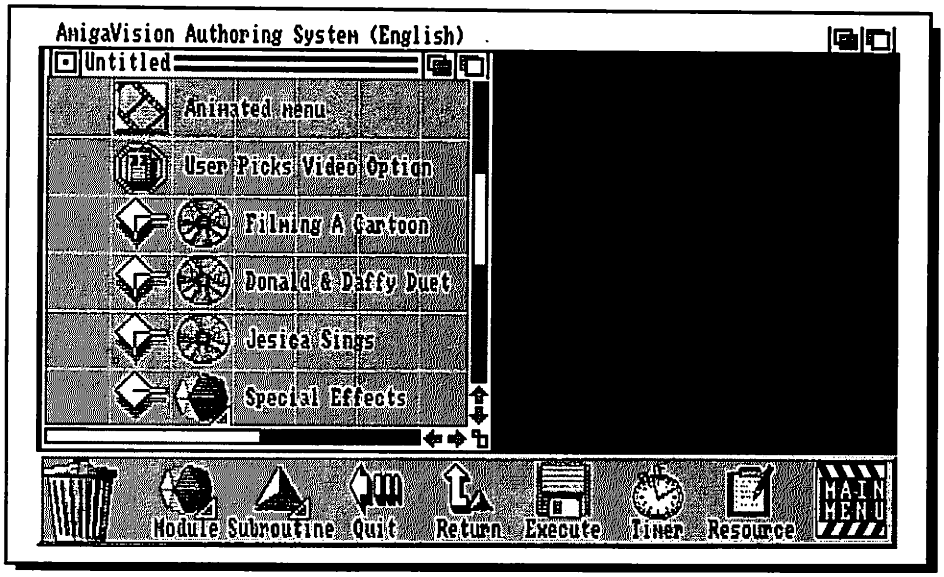


Figure 6-16

A complex If-Then-Else decision structure used to create the film, Who Framed Roger Rabbit?

This grouping of commands serves two important needs. First, it is used to logically organize the decision-making process. These groups of If-Then-Else commands help give complex decisions a more understandable, structured organization. Second, and just as important, is that once one of the commands is evaluated as true, the others are skipped. This serves to speed up the overall execution of this section of the program. If you use just If-Then statements for each condition, even when the user response is determined and the appropriate video sequence plays, the remaining If-Then commands are still evaluated.

Loop

Moving away from conditional statements, we find the Loop icon command. The Loop icon does what its name implies — it creates a loop in the program that is executed a variable number of times. As the triangle on its lower right indicates, the Loop icon is a parent, capable of having any icons (except the Subroutine icon) as children. It is the children of the Loop icon that are executed repeatedly. When AmigaVision encounters a Loop icon, it executes the children of the loop over and over until it has repeated a specified number of times, until a Loop Exit icon is encountered, or until a forced exit occurs by branching with the Goto or Conditional Goto.

Because the Loop icon can contain other Loop icons, and they can contain still more Loop icons, it is possible to create program structures known as nested loops — loops with loops inside. You can use these loops for very intense data processing, as the internal loops are executed multiple times by the outer loops. As an example, suppose we have a loop nested three deep, that is, a loop inside another which is inside yet another. If we specify that each loop be execute just ten times, the icons in the outer loop would be executed only the ten times. The icons in the middle loop, however, would be executed one hundred times (10 * 10), while the icons in the inner loop would be executed one thousand times (10 * 10 * 10)!

There are three types of loops that you can define with the Loop icon. The first, the Conditional Loop, works with a user-defined expression. As long as the condition specified in the expression evaluates as true, the loop will continue to operate. Once the condition becomes false, the loop will be completed.

The second form of Loop icon is the Endless Loop. It will continue looping forever until exited with the Loop Exit, or until a Goto or Conditional Goto branches the program out of the loop.

The third type of loop is the Counted Loop. This is the traditional type of loop programmers have used since high-level languages first appeared. For the Counted Loop, the programmer defines a starting and ending value along with an amount to increment the counter each time through the loop. The loop will continue until the counter equals or exceeds the assigned stop value or until a forced exit occurs.

To use the Loop icon, place it into position in the program flow chart. You can immediately begin to place child icons beneath and to the right of the Loop icon (see Figure 6-17). Depending on your personal preferences and the needs of the particular program you are writing, you can define the Loop icon's mode before or after adding the child icons.

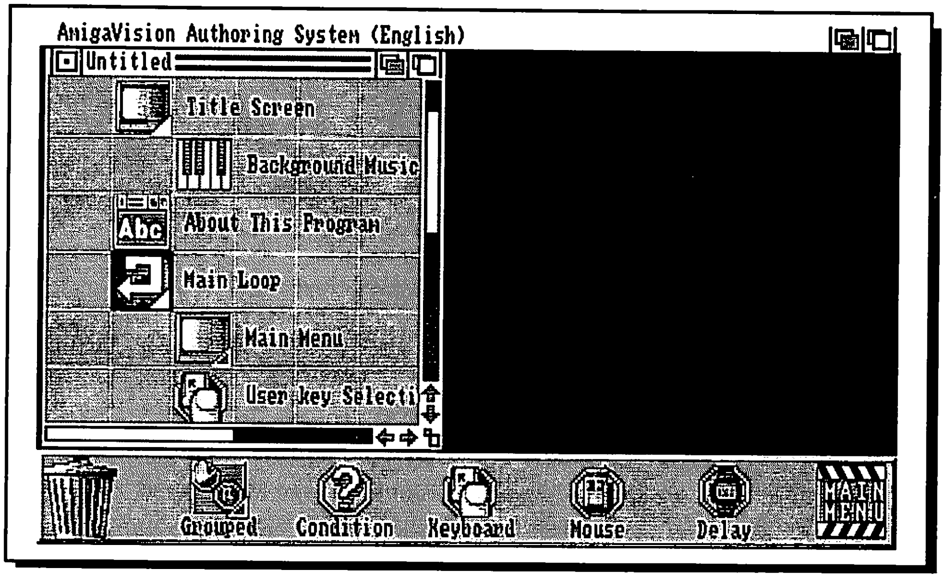


Figure 6-17
The Loop icon as a parent

When you are ready to define the Loop icon, double-click on the Loop icon and AmigaVision will display the Loop Definition requester (see Figure 6-18). The default looping mode is the Endless Loop.

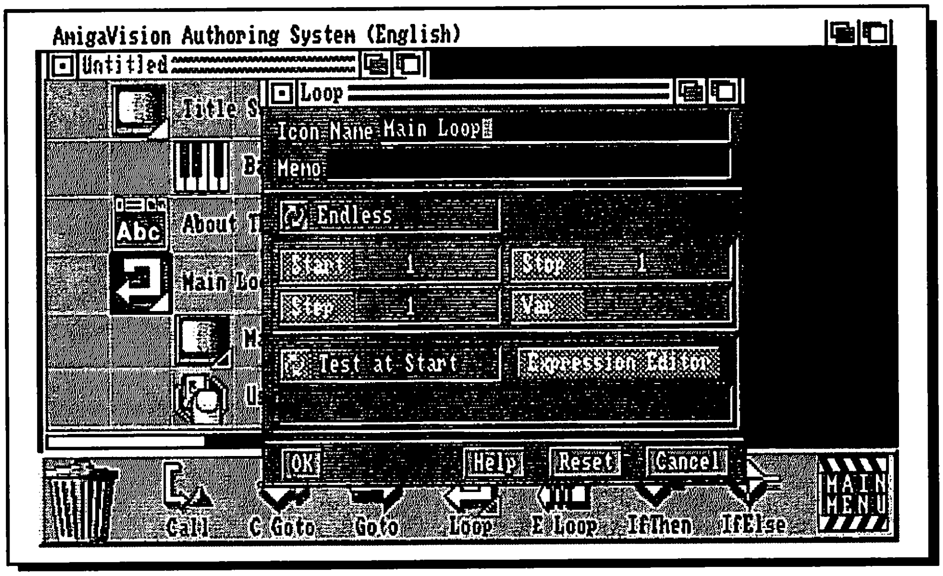


Figure 6-18
The Loop Definition requester in Endless Loop mode

Endless Loop Mode

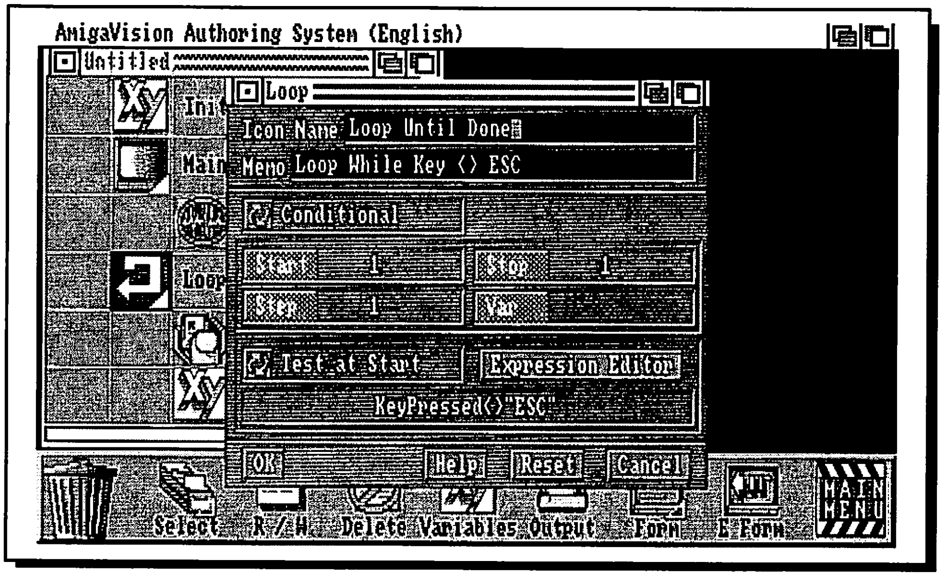
If Endless Loop is what you want, click in the icon Name field and give the Loop a name, and, if you wish, add a memo in the Memo field (memos are useful when you want to add a comment to help clarify the purpose of the command). Except for the button next to the word Endless and the standard OK, Help, Reset, and Cancel buttons, all the command gadgets on the requester are ghosted, or disabled. (The Reset button is used to reset the parameters of the requester to what they were when it was first displayed.) Because there are no other items available to define an Endless Loop, clicking on the OK gadget will exit the loop requester.

Conditional Loop Mode

If you want a loop to continue until a certain expression is no longer true, you should define it as a Conditional loop. As you can see in Figure 6-18, there is a multistate gadget in the Loop requester next to the word Endless. Clicking on this gadget allows you to select between the different types of loops AmigaVision supports: in this case you would click on it until the word Conditional appears.

When you switch Loop mode to Conditional, the requester changes. Two gadgets switch from their ghosted, or inactive, state and appear in normal type, indicating that they are now accessible. One of the buttons, a multistate gadget, switches from Test At Start to Test At End when you click on it. The other is a command button labeled Expression Editor.

To define the expression that determines the conditional exit, click on the Expression Editor button. When the standard Expression Editor requester appears, enter your expression using the mouse or keyboard. While creating the expression, give serious thought to exactly how it works: as long as the expression remains true, the loop continues; when it becomes false, the loop stops. Once you have defined the expression, press the Return key or click on the OK gadget; the expression will be assigned to the Conditional Loop and you will return to the Loop requester. Once you have defined the expression, the text field below the Expression Editor button will show the expression just as you entered it (see Figure 6-19).


AmigaVision Authoring System (English)

Untitled

Loop

Icon Name: Loop Until Done

Memo: Loop While Key <> ESC

Conditional

Start 1 Stop 1

Step 1 Var

Test at Start

Expression Editor

KeyPressed()<>"ESC"

Figure 6-19
The Loop Definition requester in Conditional Loop mode

The Test At Start/End gadget allows you to select where in the loop the expression is checked. If you choose Test At Start, AmigaVision checks the conditional expression at the beginning of the loop. If it finds the expression to be false, the loop will be terminated. If, on the first time through the loop, the condition evaluates as false, none of the Loop icon's children will be executed.

If you select Test At End mode, the conditional expression is tested at the end of the loop. So no matter what, the Loop icon's children will be executed at least once. This can have an important effect on your programs, so you should choose the location for conditional evaluation carefully.

Counted Loop Mode

The final mode of the Loop icon is Counted Loop. Counted mode allows you to specify exactly how many times the icons in the loop will be executed. To use it, click on the multistate gadget until the Loop requester registers Counted mode. As before, the requester will undergo a change — some gadgets will become enabled while others will become disabled. In Counted mode, the Start, Stop, Step, and Var buttons are enabled (see Figure 6-20).

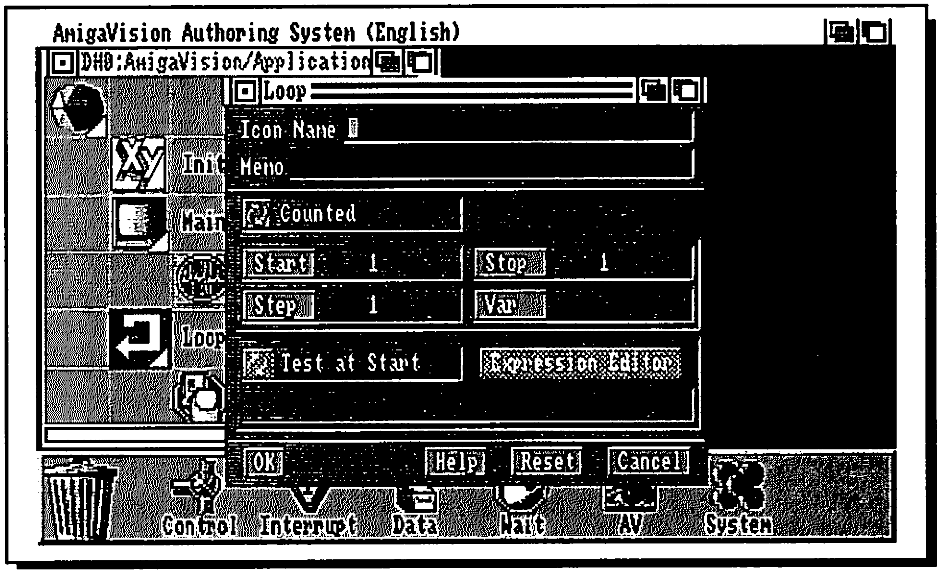


Figure 6-20
The Loop Definition requester in Counted Loop mode

The Start and Stop buttons allow you to define the starting and ending values for the loop. You can determine the size of the increment by defining the Step value. To assign values for any of these three parameters, you first click on the button you wish to define. A Specify Value requester will then appear (see Figure 6-21).

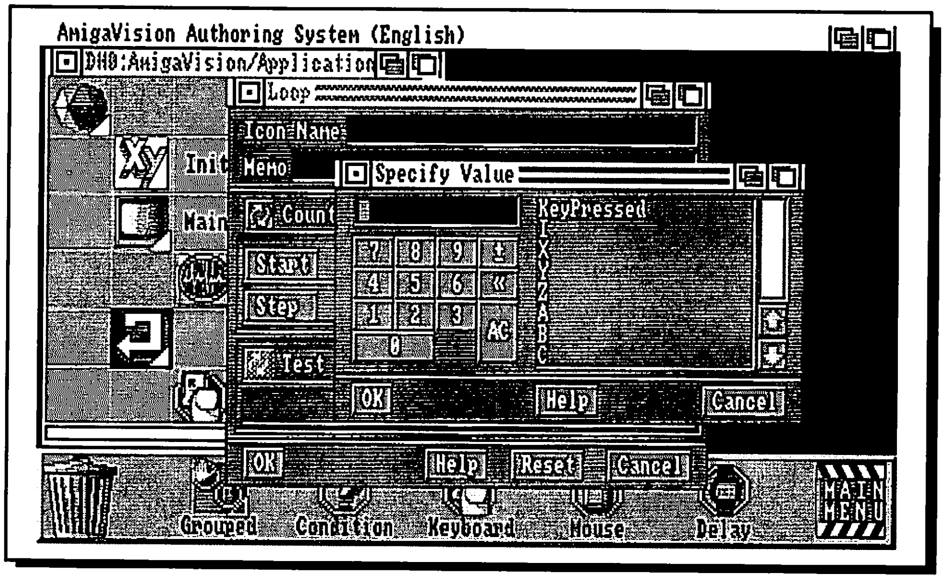


Figure 6-21
The Specify Value requester supports only integer input

Here, you can enter a positive or negative integer value by clicking on the numeric keypad or, if any are available, you can choose a variable from the list on the right side of the requester by clicking on it with your mouse. The Specify Value requester supports only integer (whole number) input. If you want to use non-integer values in your loop, you can do so by defining them as variables and then using the variables as your Start, Stop, or Step values.

The Var button allows you to assign the value of the counter to a variable, so that the value will change each time through the loop. When you click on the Var button, a Specify Variable requester appears (see Figure 6-22). Select the variable you want to use by clicking on it in the list (remember that a variable must have already been created before it can appear in the list), and then clicking on the OK button. (You can also select and accept a variable by just double-clicking on it in the list.)

Once you have assigned a variable to the counter using the Var gadget, every time AmigaVision runs through the loop, the variable is updated with the current counter value (see Figure 6-23).

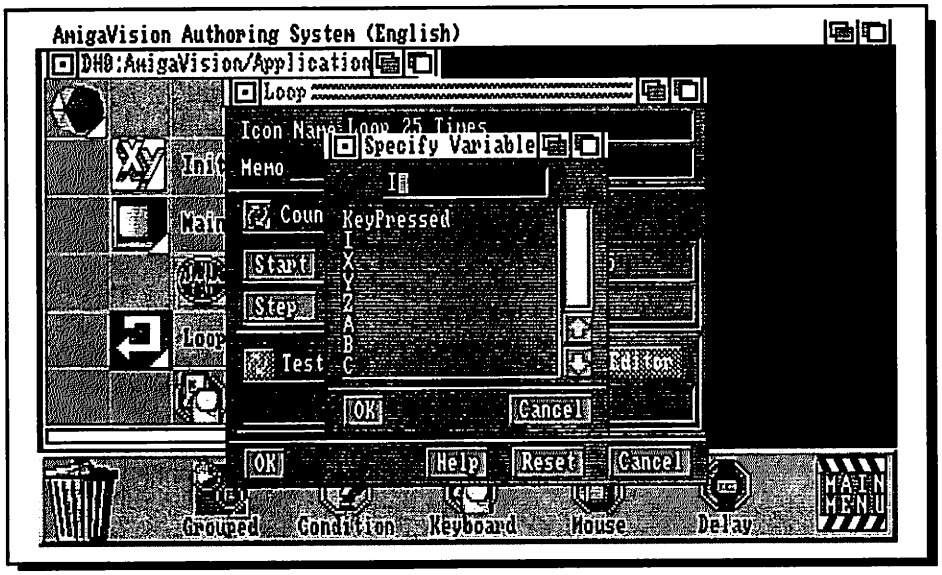


Figure 6-22
The Specify Variable requester appears when the Var button is clicked

The Step value determines the amount by which the Start number will be incremented each time through the loop. For example, if the start value is 1, the stop value is 100, and the step value is 1, the loop will occur 100 times. In traditional BASIC syntax this is a For..Next loop and is written like this:

```
For I=1 to 100 Step 1
```

If, however, the step value is 0.1, the loop will occur 1000 times:

```
For I=1 to 100 Step 0.1
```

Of course you don't have to use a Start value of 1; any number will do. For example, if the loop has a start value of 5, a stop value of 50 and a step value of 5, the routine will count by 5 and loop 10 times. If you assign a variable to equal the current value of the counter, the first time through the loop the variable would equal 5, then next time through it would equal 10, the time next 15, and so on until the final loop when it would equal 50. For an example of a defined Counted loop, see Figure 6-23.

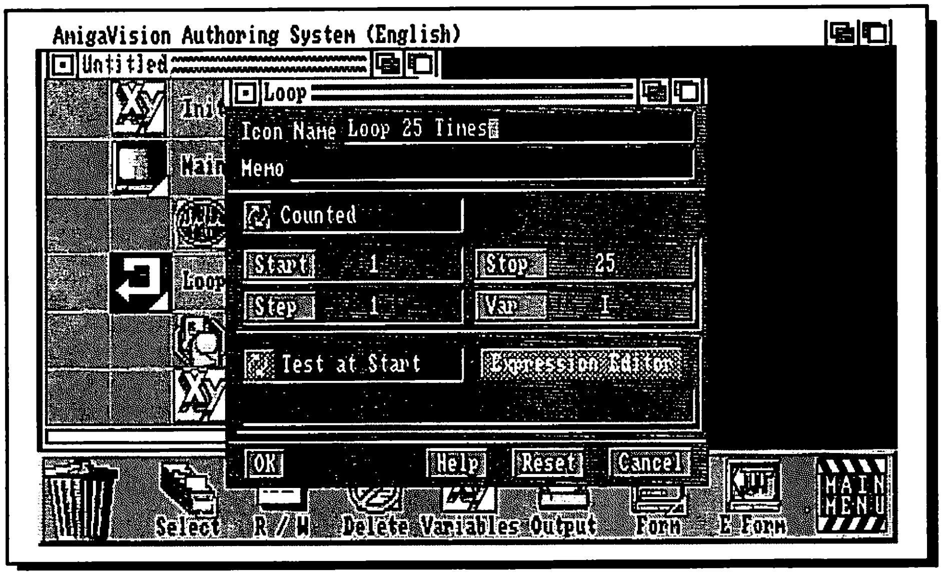


Figure 6-23
A defined Counted loop doesn't necessarily have to use positive values only

As mentioned before, the Counted Loop doesn't have to use positive values only. You could, for instance, define the start value to be 10, the stop value to be 1, and the step value to be -1. In this case, the loop would count backwards from 10 to 1, decreasing by 1 each time through the loop. If a variable had been assigned, it too would have a constantly decreasing value, starting at 10, then equalling 9, 8, 7, and so on.

Loop Exit

To exit a loop, you can use the Loop Exit icon. When AmigaVision encounters a Loop Exit icon within an Endless, Conditional, or Counted loop, it exits the loop immediately. It does not complete the loop, meaning that any icons following the Loop Exit icon are ignored and not executed.

To use the Loop Exit icon, just place it into position in your flow chart. It can be used as a simple Sibling icon, or act as a partner of another icon like the If-Then icon. If you double-click on the Loop Exit icon, you are presented with a requester that allows you to give the Loop Exit icon a name and to enter a memo for it (see Figure 6-24). Otherwise, the icon has no other definable attributes.

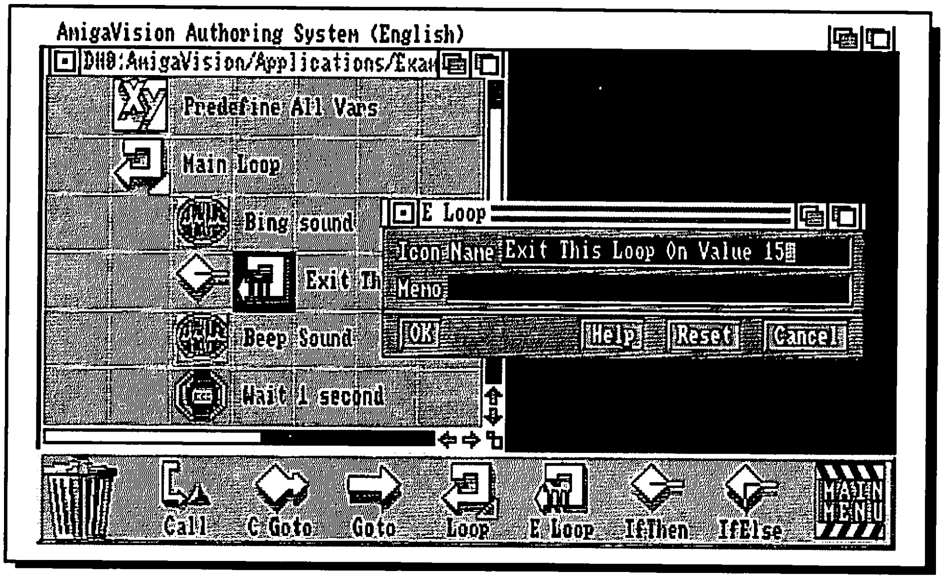


Figure 6-24
The Loop Exit requester allows you to name a Loop Exit icon

You can also use the branching icons Conditional Goto and Goto to exit a loop. Use one of these in place of the Loop Exit icon if you want program flow to exit the loop and continue on from a position in the program other than the Sibling icon following the initial Loop icon.

– 7 – The Interrupt Command Menu

The issue of using interrupts (IRQ) in computer programs traditionally involves a complex discussion. Fortunately for us, AmigaVision contains relatively easy-to-use commands that allow programmers to take advantage of this very powerful programming technique without much hassle.

Exactly what is interrupt programming? Well, before discussing it in computer terms, let's first explain it in human terms, using a parent with small children as an example. The parent is busy with day-to-day homemaking duties, and while working at some task, one of the children comes up and asks for a drink of water. The parent stops what he or she is doing and takes care of the child's request, then resumes the original task. Shortly thereafter, another child interrupts and asks for lunch. The parent feeds the child, then again resumes working. The children temporarily interrupted the main task, but afterward, the work was continued.

Like the parent in the above example, your computer is constantly working on a variety of tasks. And like the parent, it is always being interrupted with requests to do other things. In fact, these interruptions are very systematic, occurring on the order of 60 times per second — the exact rate at which the computer's screen is refreshed (updated). Because the screen-refresh interruption occurs at such exact intervals, and because it occurs so often, programmers like to use it for tasks they want to repeat continually.

Consider a simple example. Suppose you want your program to monitor the keyboard and respond somehow when it finds that a user has pressed the ESC or Help keys. One way to do that is by writing code that constantly checks for these events. Although you can do this (if your program is supposed to be doing other things at the same time), the effect of such constant polling is an overall degradation in the performance of your program.

AmigaVision gets around this problem by using system interrupts to help watch for events that you want to be aware of. By defining an interrupt event, you can set your program up to automatically execute whole sections of code when that event occurs. The AmigaVision Interrupt menu is accessed from the Main menu by clicking on the Interrupt icon (see Figure 7-1). This brings up a submenu that contains the actual Interrupt Command icons (see Figure 7-2).

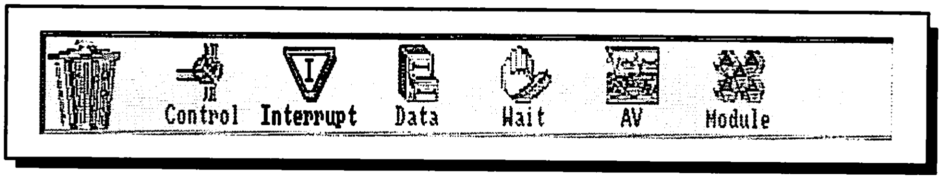


Figure 7-1
The Main menu Interrupt icon

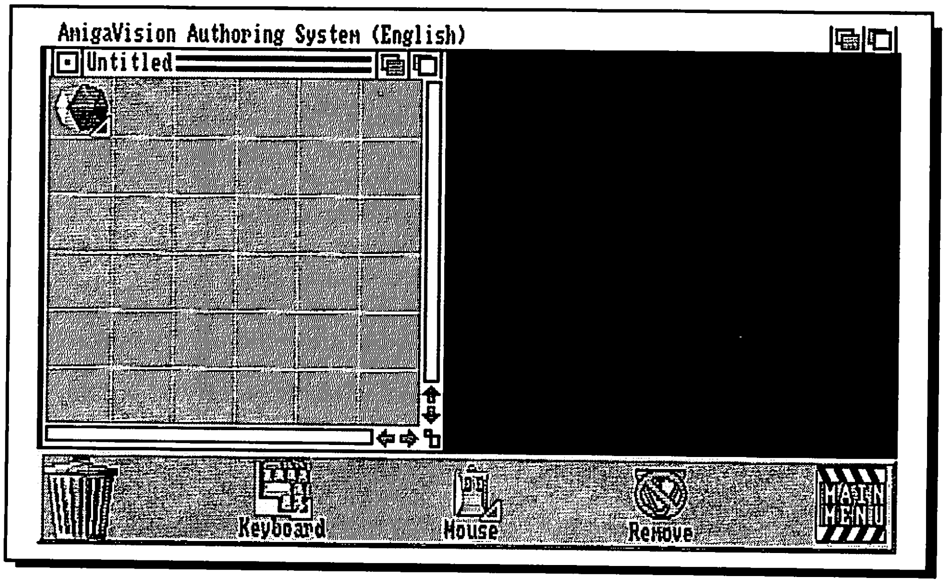


Figure 7-2
The Interrupt Command menu appears when the Interrupt command menu is clicked

Once the program has finished running through this interrupt code, execution of the main program resumes where it left off. AmigaVision checks for events that can trigger an interruption 60 times a second, but while an interrupt routine is being executed, the Interrupt icon is disabled. Disabling the interrupt prevents it from accidentally being called from within itself — a situation that could cause an endless loop to occur.

AmigaVision supports two types of interrupt events: keyboard and mouse events. Each of these has its own specific command icon. In addition, a third command icon is supplied to remove interrupts that are no longer needed.

An interrupt's activity is based on its position in the overall application. Its range of activity includes its own level and below, meaning that its activity is recognized only by its Sibling icons and their children. An interrupt has no effect on its Parent icons or upon generations before its parents.

This range of activity has two implications. First, it allows you to program interrupt actions that are localized in scope. Different interrupt events can be associated with different parts of a program. By placing an Interrupt icon in a lower level of the program (as the child of another child) the interrupt can affect only sections of your program at that level and below.

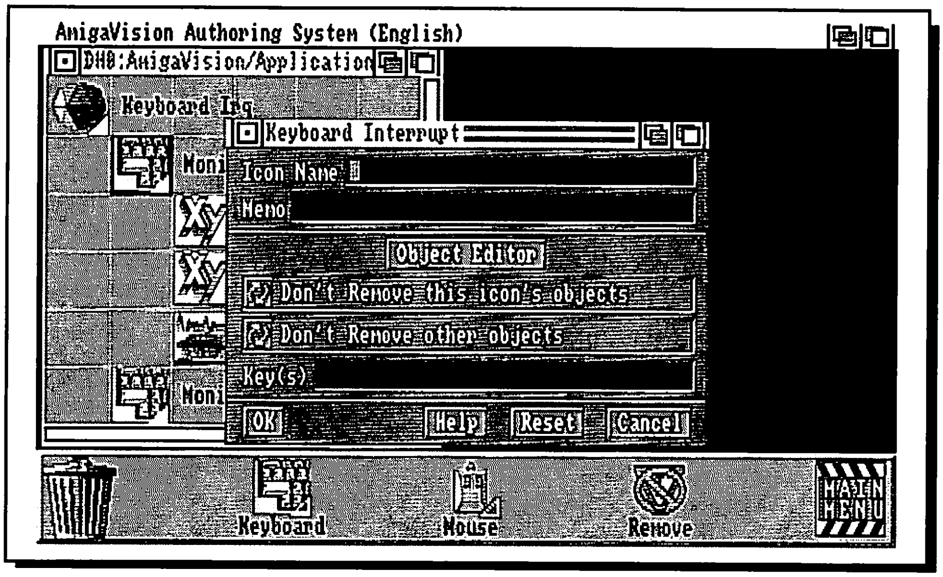
Second, if you want an interrupt event to be globally active (active anywhere in your program) you must define it at the highest level of your program. This means it must be either a sibling or a child of the initial Module icon found at the beginning of every AmigaVision program.

The Keyboard Interrupt

The Keyboard Interrupt command icon allows you to specify which keyboard events trigger an interrupt routine. To define an event, double-click on the icon. When you do, the Keyboard Interrupt requester will appear (see Figure 7-3).

As you can see in Figure 7-3, the Keyboard Interrupt requester has the standard Icon Name and Memo fields, as well as the OK, Help, Reset, and Cancel buttons associated with most icon commands. In terms of defining the command, the most significant icons are the Key(s) text gadget, the two multistate gadgets that specify how other graphic objects on the screen should be treated when the interruption occurs, and the Object Editor button. Since both the Mouse Interrupt and Keyboard Interrupt icons have the same multistate gadgets, these gadgets will be discussed later in this chapter.

The Key(s) field is used to indicate which specific keys will cause an interrupt event. For example, suppose you want the letter *b* to trigger a branch to the keyboard-interrupt routines. By entering a lowercase *b* into the Key field, you are instructing AmigaVision to trigger an interrupt event whenever a user presses the *b* key. When that event occurs, AmigaVision stops what it is doing, jumps to this Interrupt icon, and executes all the Child icons of that particular Interrupt icon.


AmigaVision Authoring System (English)
DH0:AmigaVision/Application
Keyboard Int
Moni
Moni
Keyboard
Mouse
Remove
MAIN MENU

Keyboard Interrupt

Icon Name

Memo

Object Editor

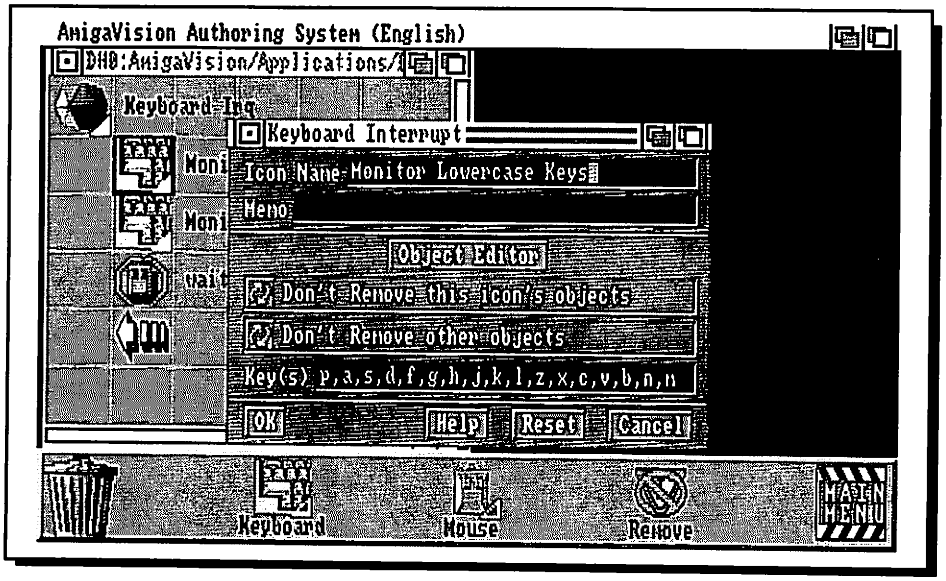
Don't Remove this icon's objects

Don't Remove other objects

Key(s)

OK Help Reset Cancel

Figure 7-3
The Keyboard Interrupt requester


AmigaVision Authoring System (English)
DH0:AmigaVision/Applications/
Keyboard Int
Moni
Moni
wait
Keyboard
Mouse
Remove
MAIN MENU

Keyboard Interrupt

Icon Name Monitor Lowercase Keys

Memo

Object Editor

Don't Remove this icon's objects

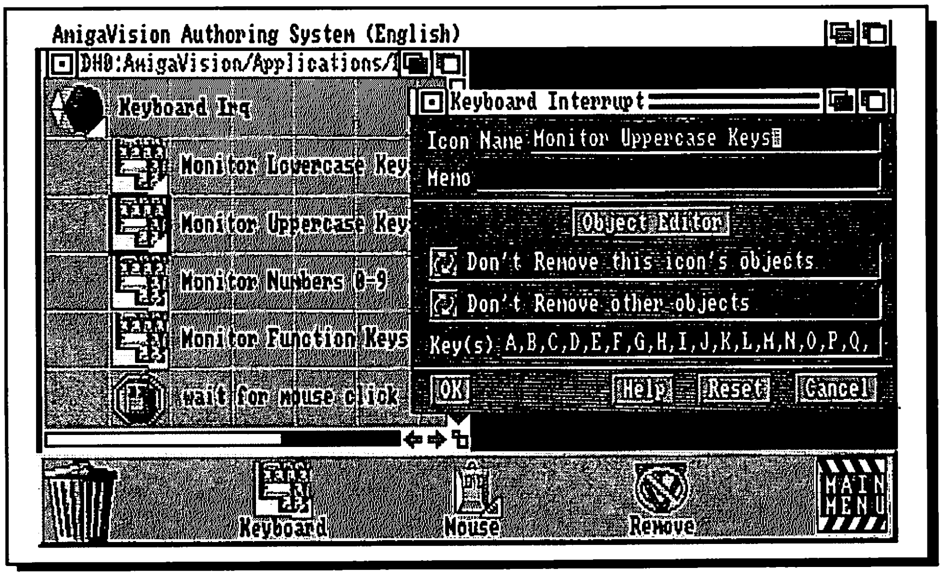
Don't Remove other objects

Key(s) p,a,s,d,f,g,h,j,k,l,z,x,c,v,b,n,m

OK Help Reset Cancel

Figure 7-4
Defining multiple keys in a single Keyboard Interrupt requester

You can use a single Keyboard Interrupt icon to define multiple keys as event triggers. Just enter them in the Key field, separating them with commas (see Figure 7-4). Because the Key field has a limit on the number of characters it can accept, you can at most define 40 single keys (40 characters separated by commas) in one requester. If your program requires more than 40 keys, you can define multiple Keyboard Interrupts. In one program I created, I needed to monitor all the lowercase and uppercase letters, all the number keys, and all ten function keys (F1-F10), with each group triggering a different set of events. The easiest way to set this up was to define four Keyboard Interrupt icons, each specific to a different group of characters (see Figure 7-5). There is no limit to the number of keyboard-interrupt routines you can use in an application, but there are some considerations you should keep in mind. When using multiple interrupts that are active at the same time (all in the same hierarchical level in the program), you should use each for separate events. In other words, don't use the same key in multiple concurrent interrupts. If you program more than one Interrupt icon to watch for the X key, only the last interrupt will be executed if the event occurs.

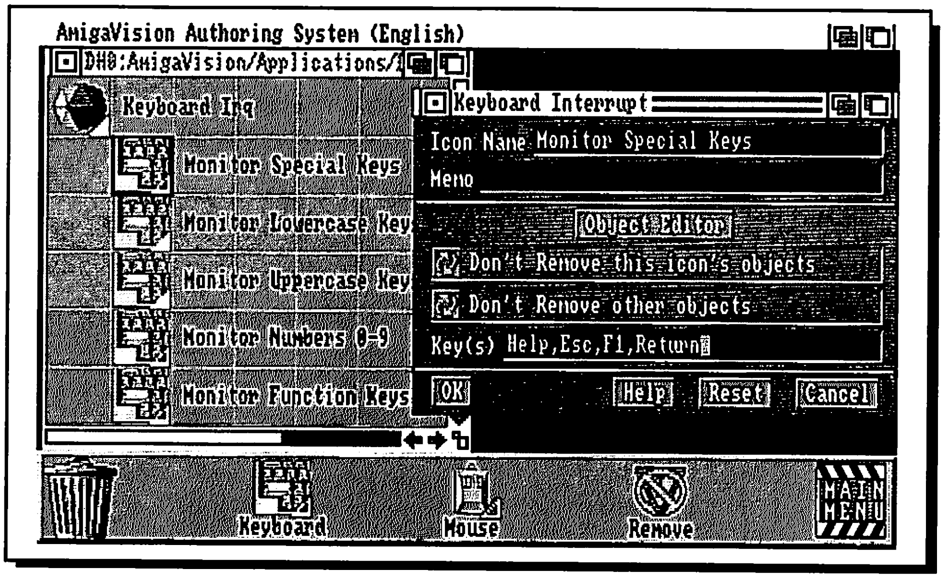

AmigaVision Authoring System (English)
DH0:AmigaVision/Applications/
Keyboard Irq
Monitor Lowercase Key
Monitor Uppercase Key
Monitor Numbers 0-9
Monitor Function Keys
wait for mouse click

Keyboard Interrupt
Icon Name Monitor Uppercase Keys
Memo
Object Editor
Don't Remove this icon's objects
Don't Remove other objects
Key(s) A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,

Figure 7-5
Defining multiple keyboard interrupts

As you may have gathered from reading the preceding paragraph, you are not limited to just ASCII characters in defining interrupts. AmigaVision's Keyboard Interrupt icons support a number of special keys in addition to the standard alphanumeric keys. These include function keys (F1-F10), Escape (Esc), Help, Tab, Delete (Del), Space, Enter, and Return keys, and the four cursor keys (Up, Down, Left, and Right). There is no provision for handling the Alt or Amiga keys, either alone or in combination with other keys.

To use any of these nonalphanumeric keys, simply enter the name of the key in the Key text gadget. If you want to enter more than one name, separate them with commas just as you would do with standard characters (see Figure 7-6).



Icon Name Monitor Special Keys
Memo
Don't Remove this icon's objects
Don't Remove other objects
Key(s) Help,Esc,F1,Return

Figure 7-6
Using special keys to trigger interrupts

With the exception of the special keys listed above, AmigaVision's keyboard interrupts do not accept words as event triggers. For instance, while you can use Esc or Help as an event, you cannot use words like Louis or Amiga.

Once you have defined a key that you want to trigger an event, you must specify what is to happen when the user presses that key. This involves writing the section of AmigaVision code that will be executed when the interrupt event triggers the program flow to branch to the Interrupt icon.

Interrupt icons are Parent icons, and you can place child icons to the right of and below them in the flow chart. It is these child icons that will be executed when the interruption occurs.

Exactly what types of icons you use as children of the Interrupt icon depends totally on what you need to have happen. With the exception of the Subroutine icon, any icon can be a child of the Interrupt. Although a Subroutine icon cannot be the child of an Interrupt icon, however, subroutines *can* be called from within an interrupt.

Figure 7-7 shows one example of using an interrupt. This routine is executed whenever the user of the application presses the Help key: a Help screen opens and some music plays. The display remains and the music continues until the user either clicks the mouse or presses a key. When one of those events occurs, the music stops, the Help screen disappears, and the interrupt routine finishes. Program execution then returns to where it was when the user pressed the Help key.

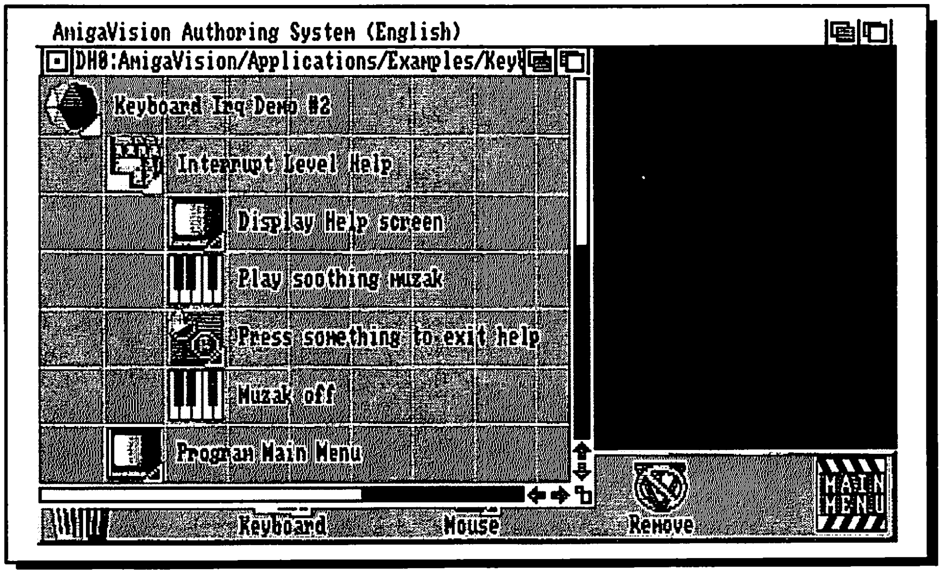


Figure 7-7
A sample Keyboard Interrupt routine

There are many uses of the Keyboard Interrupt. Exactly what you do with it will depend on the needs of your program, your level of programming expertise, and, of course, your imagination.

Besides executing sections of code when a specified event occurs, the Interrupt icon can also generate graphic displays via its Object Editor access. Click on the Object Editor button in the Interrupt requester, and you will be presented with the Object Editor screen. From here, you can display rectangles, lines, circles, ellipses, text (with variables), and graphic brushes. The program will display whatever graphic elements you specify whenever the Interrupt icon subroutine is executed.

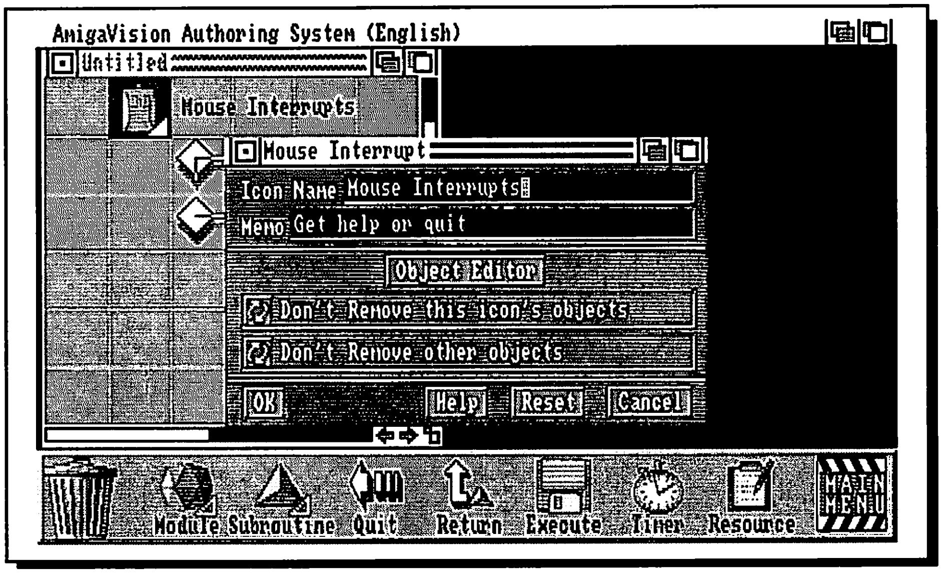
Keep in mind that graphics generated from the Object editor for use in an interrupt will be displayed before AmigaVision executes any other part of the interrupt code. That means these graphics will be displayed in the screen resolution and mode that is in effect at the time of the interruption. If your application uses the same screen sizes and modes at all times this will not cause any difficulty. If, however, your interrupt routine contains screens of different resolutions or modes, for at least a moment the graphic elements generated in the Object editor will be in a different display mode. In this case, you will have to take a different approach to adding graphic objects. One solution to this situation is to avoid using the Object editor from within the Keyboard Interrupt icon, and instead, add a Mouse Wait or Key Wait icon after the new screen and access the Object editor from within one of those icons. Both approaches are equally valid; the task for you as programmer is to consider all possibilities when designing your interrupt code.

The Mouse Interrupt

The second type of interrupt supported by AmigaVision is the Mouse Interrupt. Like the Keyboard Interrupt icon, a Mouse Interrupt icon is recognized in a program only by its siblings and their children. To define a Mouse Interrupt icon, position it in your program and double-click on it. This will bring up the Mouse Interrupt requester (see Figure 7-8).

The Mouse Interrupt requester is quite simple, consisting of an Icon Name and Memo Field, two multistate gadgets that affect how graphics are handled during the interrupt (see below for a complete description of these functions), the standard OK, Help, Reset, and Cancel buttons, and most importantly, a button for accessing the Object editor.

The Object editor is of paramount importance in Mouse interrupts because the interrupt events are defined completely within the Object editor by the addition of mouse "hot spots" to the screen. Unlike objects you create when you access the Object editor from within other icons, those you create in the Mouse Interrupt icon are screen hot spots, which do not merely return a string of text in response to a mouse click. Rather, when a user of your program clicks on a hot spot, an interrupt event causes program execution to branch directly to the Mouse Interrupt icon and its children. No additional polling of responses is required.



Icon Name: Mouse Interrupts
Memo: Get help or quit

Figure 7-8
The Mouse Interrupt requester appears when the Mouse Interrupt icon is double-clicked

Figure 7-9 shows a Help button being defined in the Object editor. For this example, I used a paint program to create a brush shaped like a button, and loaded it into the Object editor. In this illustration, the brush is being defined to return a response of Help whenever the user clicks on it. Because it is defined as an interrupt event, the button will appear on screen whenever the interrupt is active.

If the user clicks on this button, program flow will immediately branch to the Mouse Interrupt icon and the application will begin to execute that icon's children. If one of those children uses the Response() function from the Expression editor, that function will return the text string "Help," indicating it was the Help button that was selected. In a simple situation, where there is only one hot spot, the Response() would not be necessary. In many cases, however, you will have multiple buttons or hot spots, and the value returned by Response() will be necessary to determine exactly which event triggered the interrupt.

As an example, Figure 7-10 shows a Mouse Interrupt routine that has both Help and Quit hot spots. If the person using your program clicks on Help, a Help screen appears and remains until the program detects that the user has clicked with the mouse or pressed a key. If, on the other hand, the user selects Quit, the program is exited.

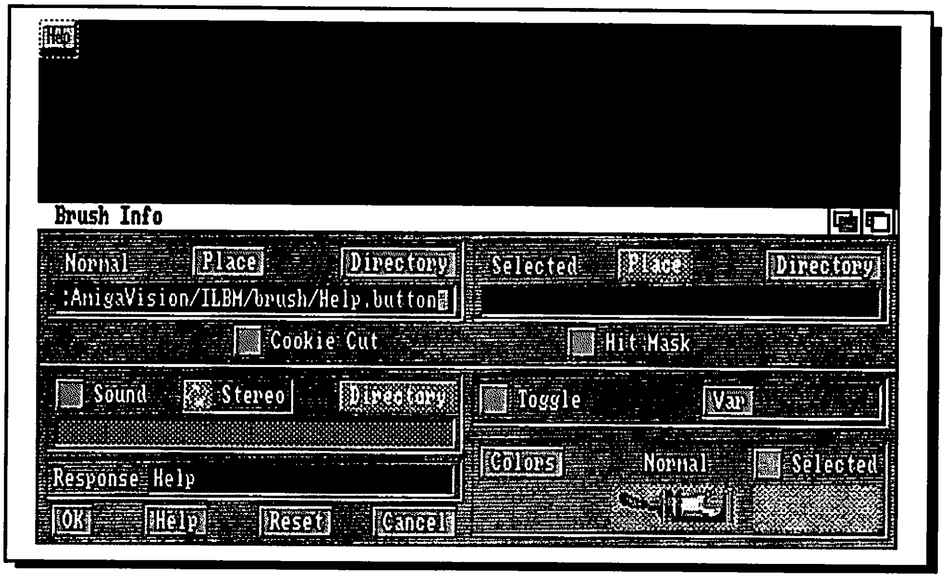


Figure 7-9
Defining interrupt hot spots in the Object editor

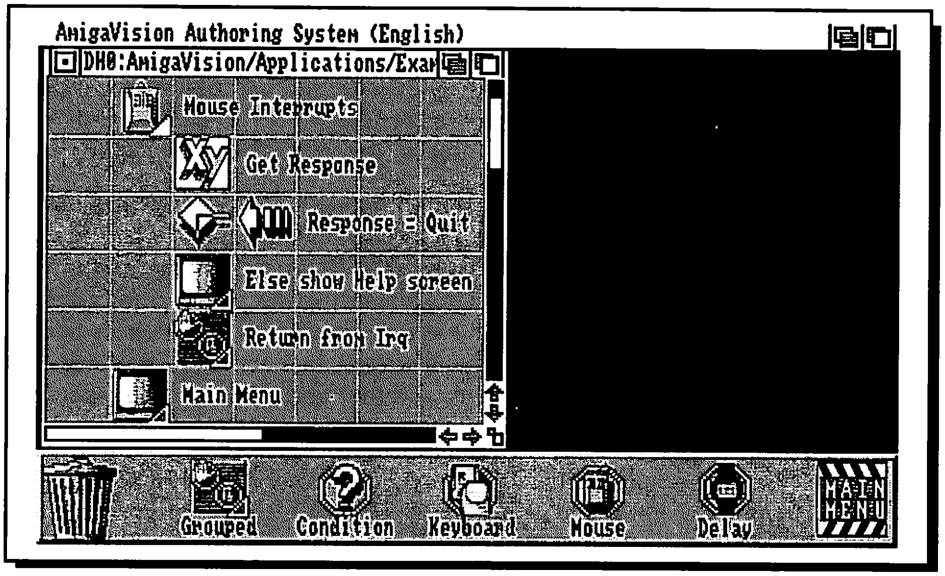
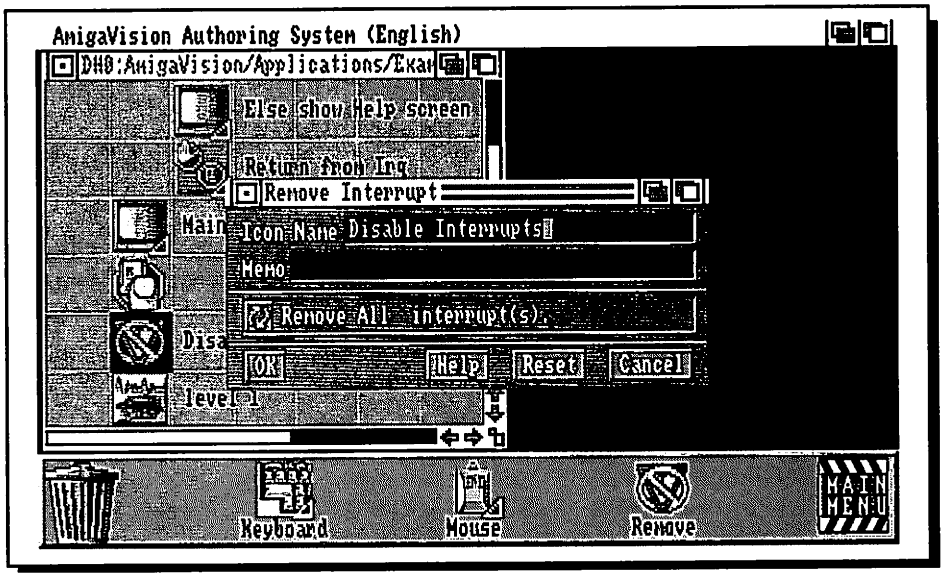


Figure 7-10
A Mouse Interrupt routine containing Help and Quit hot spots

The Remove Interrupt Icon

The Remove Interrupt icon is used to remove one or more previously defined and active interrupts. It can "turn off," or disable, existing Mouse Interrupt or Keyboard Interrupt icons.

To remove an interrupt from service, place the Remove Interrupt icon in position in your program and double-click on it to bring up the Remove Interrupt requester (see Figure 7-11). This requester has only one significant feature: a multistate gadget that, when clicked, toggles between Remove All Interrupt(s) and Remove Last Interrupt(s).



Icon Name: Disable Interrupts

Memo:

Remove All interrupt(s).

Figure 7-11

The Remove Interrupt requester removes an interrupt from service

Choosing Remove Last Interrupt disables only the most recent interrupt, while selecting Remove All Interrupts disables all current interrupts. Like the Interrupt icons themselves, however, Remove Interrupt works only at its own level in the program. In other words, it has no effect on interrupts defined at the level of its parents or previous generations.

The Interrupt Object Options

The Keyboard and Mouse interrupt requesters share a set of multistate gadgets that are used to control graphic objects when an interrupt event occurs. These gadgets are concerned with two groups of objects: those created by the Interrupt icons themselves, and those already present when the interrupt event occurs.

The first gadget in these requesters determines how the interrupt-generated objects will be treated. It has three options:

- Don't Remove this icon's objects
- Temporarily Remove this icon's objects
- Permanently Remove this icon's objects

The graphic objects that are affected by these options are those created within the Object editor when accessed from the Interrupt icon itself. They might be rectangles, lines, polygons, circles, ellipses, text strings, or graphic brushes. Once you define these objects, they will show up on screen whenever the interrupt is active — regardless of what else you might be displaying at the time. In other words, if your interrupt places a circle in the middle of the screen, the circle will be there as long as the interrupt is active, even if you load other screens from disk as well.

Choosing the first option, "Don't Remove this icon's objects," informs AmigaVision that when the interrupt event occurs, it is to keep the objects on the screen. No matter what else happens in your interrupt routine, the objects (such as the Help and Quit buttons) stay put.

Option two, "Temporarily Remove this icon's objects," informs AmigaVision that when the interrupt event occurs, you want it to temporarily remove the interrupt objects from the display. Consider an example with the Help button. Although you may want a Help button to remain on screen throughout the program, you also probably want to display a Help screen when the user clicks on the button. You would not, however, want the Help button to show up at the same time the screen is being displayed. If you select option two, the Help button will disappear while the program executes the interrupt routine. As soon as the user exits the interrupt, the Help button will be restored. If you had a whole screen full of objects, they would all disappear while in the interrupt routine, and would all be restored after exiting it.

Option three, "Permanently Remove this icon's object" removes the Help button when the interrupt occurs, but it does not restore it when the interrupt routine finishes. This has the effect of essentially disabling a Mouse interrupt, because if the object that triggers the interrupt is no longer available, you cannot click on it to trigger the interrupt a second time. If you want an interrupt to occur only once, this third option is useful.

The second gadget in the Interrupt requester is used to control objects on the screen that were placed there by the main program, not by the Interrupt icons. This gadget allows two options:

- Don't Remove other objects
- Temporarily Remove other objects

These modes affect externally generated objects in much the same manner as they do the interrupt objects. The first mode leaves the objects unaffected by the jump to the interrupt routines. The objects stay on screen throughout the interrupt. The second mode removes the objects while the interrupt code is executing. As soon as control is returned to the main program, the objects are restored to the screen.

Interrupt Code Considerations

You should keep in mind that changes made to the display in the interrupt routine are not automatically restored when the interrupt is finished. For example, if your interrupt routine loads and displays a new screen, the old one will not be visible when the main program resumes execution. In such a case, you must restore the screen yourself from within the interrupt. One way to do this is to use variables for screen names. That way, when you exit the interrupt routine, you can just reload the old screen.

As mentioned in your *AmigaVision Users Guide*, if an animation, sound, or music file is playing when an interrupt occurs, the sound, music, or animation will stop. When the interrupt is finished, it will be restarted, but at the beginning — not where it left off! What the AmigaVision manual does not tell you is that you can set up these events through the Sound, Music, and Animation icons so that they are not affected by the interruption.

On each of these commands there is a Check Box gadget labeled Pause. If you click on this button, a check mark appears on it. When Pause is thus enabled, AmigaVision will complete execution of the corresponding icon before continuing on to the next. That means your sound, music, or animation file will play to completion before any other icon is executed.

If such a file, (with the Pause option enabled), is playing when an interrupt event is triggered, the action will stop, the interrupt will be serviced, and on return, the icon will restart just as the AmigaVision manual indicates. However, if the Pause button is not enabled, it's another story.

If you have not instructed AmigaVision to pause while these icons execute, it will start the animation, sound, or music, and continue on to the next icon and begin executing it. This is a powerful feature, and is what allows AmigaVision to perform several different audio-visual events simultaneously.

If an interrupt occurs while one of these "unpaused" events is occurring, the event will continue during the interrupt! This means that you can still run sounds and animations from the main program while the interrupt routines are being executed. This is a powerful feature, but be forewarned that it can cause problems if your interrupt routines are written in such a manner as to disrupt the existing routines. For example, suppose that while you have an animation running, you try to display a Help screen (which could override your animation and cause it to stop) from within the interrupt. As another example, let's say that some music is playing, and that your interrupt routine needs a sound channel to play another sound effect. The secret here is to plan your interrupt routines carefully, making sure they will work properly in any place in your program from which they can be triggered.

– 8 – The Database Command Menu

One of AmigaVision's strengths is the database functionality of its menus and icons. AmigaVision contains a full complement of command icons that allow you to use a database in your AmigaVision applications. One thing you must understand, however, is that these commands are only for manipulation of the databases; they are not used to create the database itself.

Although you can create databases within AmigaVision, (we will discuss database design and creation in Chapter 14), that is not the subject of this chapter. Here, we will concentrate on the Database command icons.

The Database command icons can access databases created within either AmigaVision or a program that produces dBASE III-compatible files (dBASE III is a popular database manager in the MS-DOS world, and its file format has become a standard). You don't have to buy an IBM-PC in order to create dBASE files, as many popular Amiga programs, including Superbase Professional (Precision) and dBMAN (VersaSoft) generate them.

The Database Command Icons

You access the database commands by clicking on the Data icon found on the Main menu (the Data icon looks like a small filing cabinet). Selecting it replaces the Main icon menu at the bottom of the screen with the Database Command menu (see Figure 8-1).

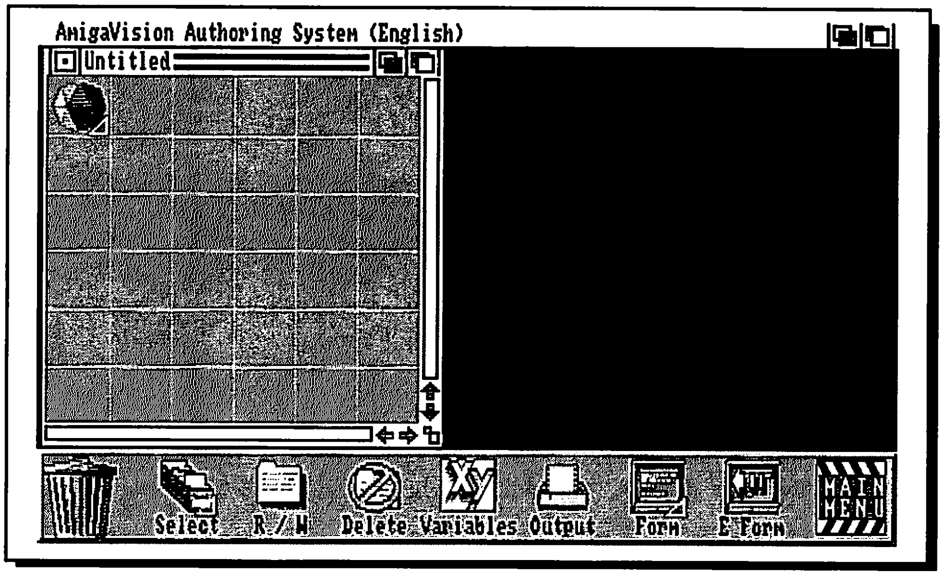


Figure 8-1
The Database Command menu replaces the Main menu when the Data icon is clicked

There are seven commands on this menu. From left to right they are: Select, R/W, Delete, Variables, Output, Form, and E Form. Of these seven, the Variables and Output icons can be used both for database programming and in sections of code unrelated to databases, while the other five are used only for database operations.

The Variables Icon

One of the most important commands in AmigaVision is the Variables icon. It is this icon that allows your programs to use and manipulate both text and numeric data. These variables can be used for many functions — to manipulate data for database input and output, to calculate numbers, to manipulate text, and to store and evaluate a user's responses during the execution of the application.

While other icons (such as the Module and Subroutine icons) can also generate variables, only variables created with the Variables command are global in nature. Once defined, global variables can be accessed from any location in your program at their level or below. Global variables are available as long as the application is active, and they maintain their value unless the application explicitly changes them.

By contrast, variables created with the Module or Subroutine commands themselves are local variables, which have meaning only within the context of the routine that created them. Therefore, variables created within the Subroutine or Module commands lose their value when those routines finish executing.

It is very important to keep in mind that while global variables are available everywhere in your program at their own level or lower, AmigaVision cannot recognize them until you define them. Consider the situation where the variable X is created and defined in the middle of an application. Any attempt to reference X in the first half of the program will result in an Unknown Variable error, which can cause your program to abort or function incorrectly. In most cases, it is a very good idea to initialize your global-variables at the beginning of your program, so they are ready for use anywhere in the program they may be encountered.

To use the Variables icon, drag it into your AmigaVision Flow window and place it in the desired position. You can use Variables icons as partners to icons that require a partner, as siblings to other icons, and as children to Parent icons. They cannot function as Parent icons themselves, however. Once you position a Variables icon, simply double-click on it to bring up the Variables requester (see Figure 8-2).

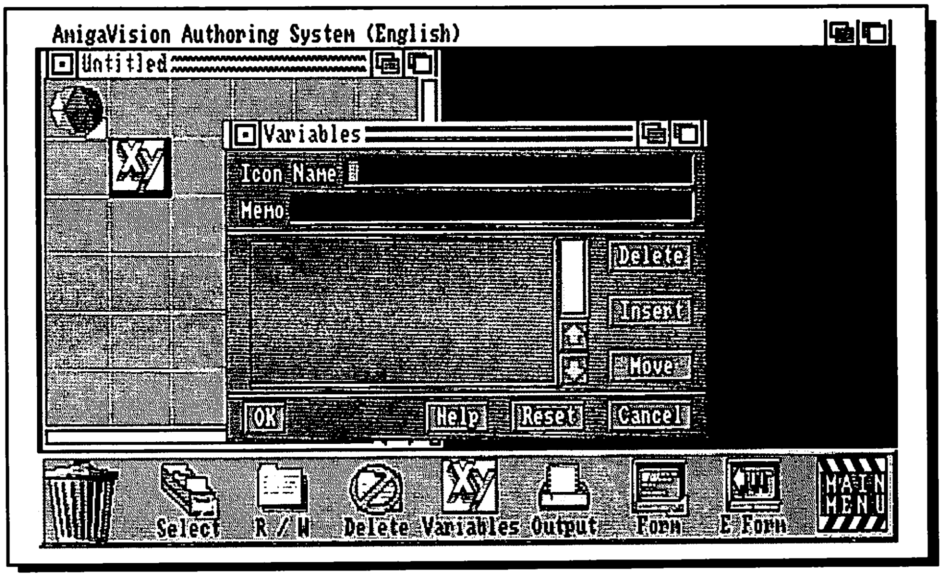


Figure 8-2
The Variables requester allows you to add, delete, or move variables

From the Variables requester you can add, delete, or move variables. To add a new variable, click on the Insert button on the Variables requester. This brings up the Expression editor, a requester you can use to create, evaluate, and modify variables (see Figure 8-3).

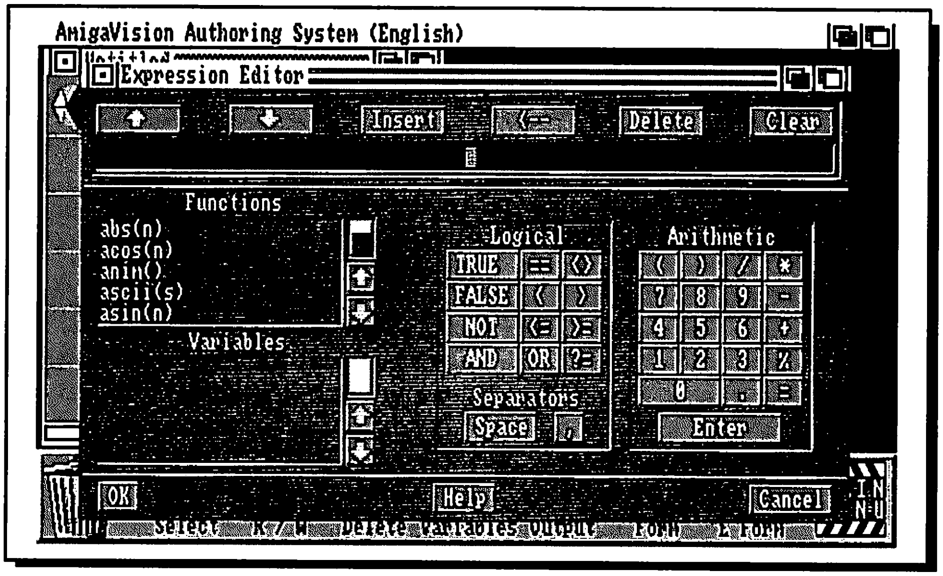


Figure 8-3
The Expression editor allows you to create a new variable

From the Expression editor, you can create a new variable by entering a value into it. For example, to create a variable A and give it a value of 5, you simply type the expression A=5 into the input field (see Figure 8-4).

Once that is done, clicking on the OK button returns you to the Variables requester. On returning, you find that the Variables window is now showing something new — the variable A and its defined value (see Figure 8-5). Now that you have defined it, the variable A can be used, evaluated, and modified by other commands in your program.

You are not limited to creating one variable per Variables icon; in fact, you can use a single Variables icon to create many new variables. To do so, enter a variable name and expression, press the Return key, enter the next variable and expression, and so on. Once you are finished, click on the OK button to return to the Variables requester, and your new variables will show up in the Variables List window

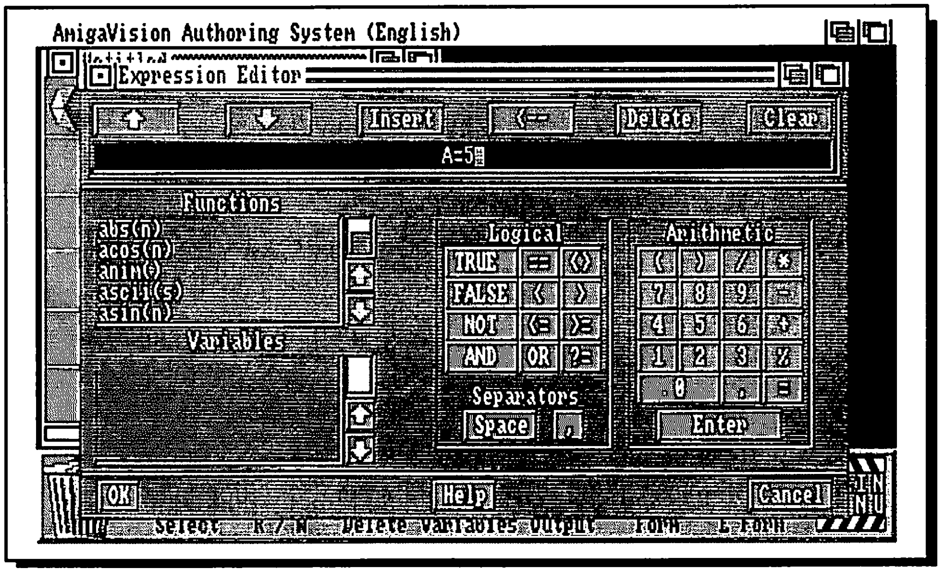


Figure 8-4
Adding a variable in the Expression editor

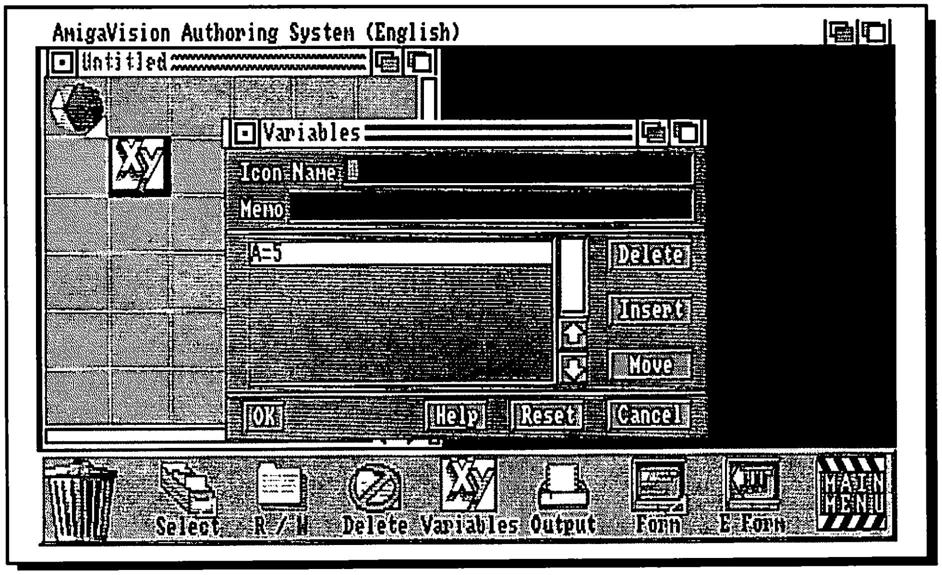
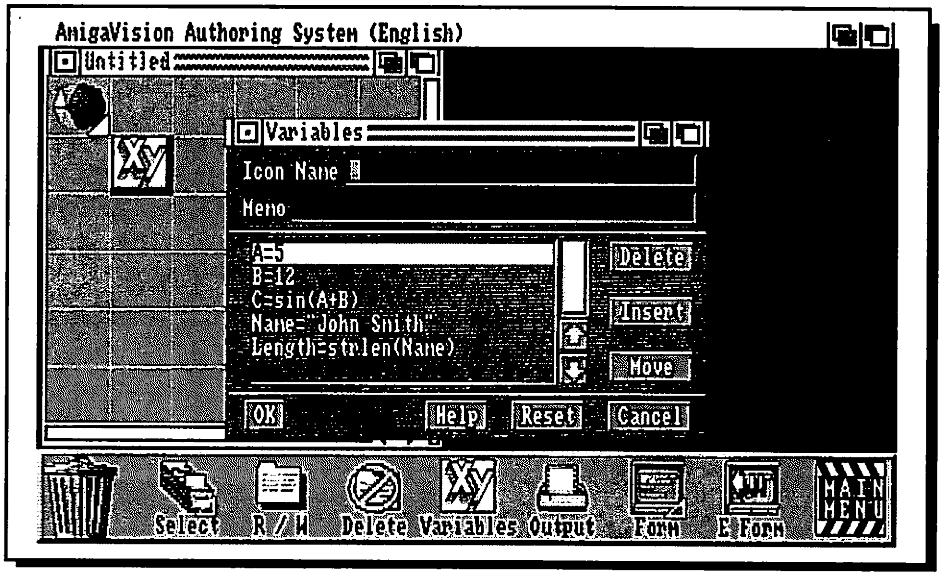


Figure 8-5
The variables list in the Variables requester

(see Figure 8-6). If you have more variables than can be shown in the Variables List window, you can use the window's scroll gadgets to move up and down through the list. Although each Variables icon can contain many variable definitions, I have found it to be a wise programming practice to limit the number of variables per icon to around ten. That way, you can quickly see what is being done in a given Variables icon, and you don't have to read through a huge list of variables to find the one you want.



```
A=5
B=12
C=sin(A+B)
Name="John Smith"
Length=strlen(Name)
```

Figure 8-6
Defining multiple variables with a single Variables icon

As you can see from Figure 8-6, you are not limited to defining a variable only as a constant. Instead, you can use mathematical operators, logical operators, dates, character strings, string-manipulation functions, and much more. Because AmigaVision is a computer programming language, it offers a very rich assortment of data-manipulation functions, which are accessible from the Expression editor. (The Expression editor is such an important requester that we will devote an entire chapter to it later in this book.)

Once you have defined a Variables icon, you need to position it in the variables list. If other variablės are present in the Variables window, you can specify where in the list the new variable is to be inserted. Just click once on the variable ahead of where you want to place the new variable; AmigaVision will highlight the name you click on. Clicking on a name indicates to AmigaVision that you want to move this highlighted variable (and all others following it) down one slot in the list and insert a new variable at the open position.

Other functions in the Variables requester are Move and Delete. Move allows you to transpose two variables in the variable list. To accomplish this, click once on a variable to highlight it, then click on the Move button. You will notice your mouse pointer change when you do this. A small, horizontal arrow with points on both the left and right ends appears just below the pointer to remind you that you are in Move Variable mode. If you change your mind about moving a variable, just click a second time on the Move button; the mode will be turned off and your pointer will return to its normal appearance.

A variable's position within the Variables window can be important. For example, consider the situation where one variable is defined as a function of another, as in this list:

```
X=7
Y=12-X
Z=X+2*Y
```

If these variables are created in this order, their values will be 7, 5, and 17. However, if the positions were different, an entirely different set of answers would result. It is important when creating variables to keep in mind that execution of a computer program is carried out sequentially. Change the sequence and you very often change the results.

You can use the Delete option to remove an existing variable from the list if you decide you do not want it to be there. Just click once on the variable you wish to remove in the Variables List window, then click on the Delete gadget. The variable will be deleted, and any variables following it in the list will move up one line in the list. Use this command with care, because once you discard a variable expression, you can restore it only by reentering it from the Expression editor.

Finally, if you want to edit the definition of a variable — to change, for example, an existing variable definition such as X=X*2 to X=X*3 — you can just double-click on it in the Variables requester's Variable List window. This causes the Expression editor to appear with the selected variable expression in the input field, ready to be changed. Make the changes and click on OK, and the new version of the expression will automatically replace the old.

The Output Icon

The Output command icon is used to send formatted output to a printer or a disk file. It can function as a Sibling, Child, or Partner icon, but not as a Parent icon. While you can use the Output icon to generate printed copy of database information, you can also use it to produce other printed material. When you direct your output to a disk file, the Output icon generates an ASCII-format text document, which almost all word processors and text editors can read. Another important possible use of the Output icon is to produce disk-based ASCII files of ARexx script instructions, which AmigaVision's Execute command icon can execute.

The Output icon sends a text stream to either the printer or a disk file. The text sequence is output on one line only. It can be a maximum of 132 characters in length; if it is longer, the excess is ignored. To use the Output icon, drag it from the Database icon menu into the Flow window and position it in your program at the proper point. Then double-click on the Output icon, and the Output requester appears (see Figure 8-7). Output defaults to the printer.

By clicking on the multistate gadget, you can toggle output between the printer and disk settings. If you choose disk output, you must specify a file name (including the full path name) for the file. (You can also use the Directory requester to specify the output file name.) If the file you name does not yet exist, AmigaVision will create it when you direct output to it. If the file does exist, AmigaVision will append the line of output to the end of the file.

At the bottom of the requester are the three main control features. These are the Column button, the Var button, and the Text field. These three functions allow you to define and position the text that will be output to the printer or disk file.

When using the Output Icon, it helps if you have a firm understanding of what is meant by the terms column and row. Think of the paper your printer will output to as a grid or matrix, composed of cells, each the size of a single character. The total number of lines you can place on the page is the number of rows the imaginary grid on your paper will support. The columns are the vertical lines of characters running from the first row at the top of the paper to the last row at the bottom of the paper. Every space on a line of text defines a column, and depending on the type of printer and the font you use, there can be anywhere from 40 to 256 columns on a sheet. (Most printers support a maximum of 80 columns, and AmigaVision supports a maximum of 132 columns.) Understanding this concept of a grid composed of rows and columns may be helpful when you use the formatting features of the Output Icon.

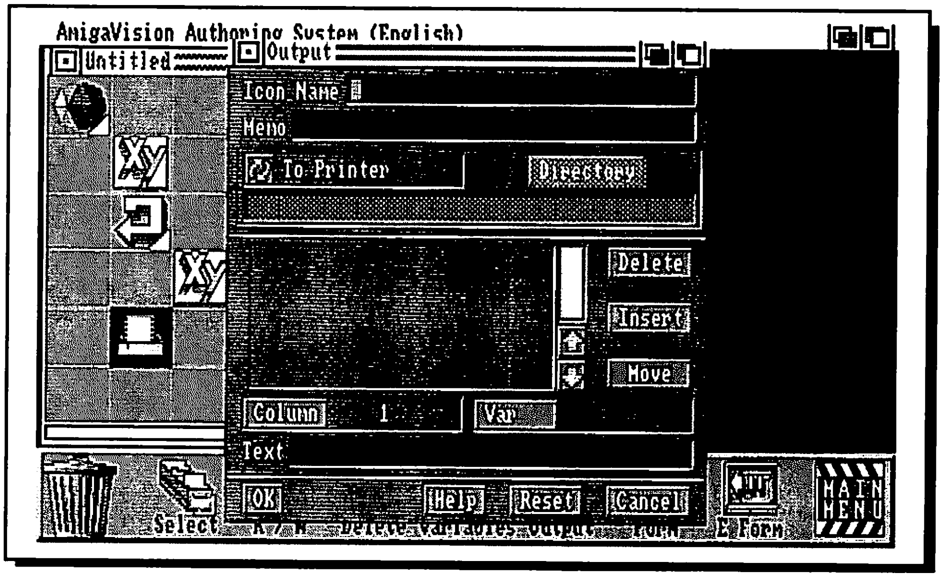


Figure 8-7
The Output requester appears when the Output icon is double-clicked

When you click on the Column button, a Specify Value requester appears (see Figure 8-8). This requester allows you to specify (by number) the column in which you want the data defined in the Text field or in the selected variable to begin. The Specify Value requester allows numeric input only; it does not accept variables. That means you must know the exact spot on the line that you intend to place this particular output. Your choices are columns 1 through 132. However, keep in mind that if the sum of the data characters being output and the column number exceeds 132, the extra data will be truncated. In other words, if your text is 30 characters long and you specify column 112 to begin the output, only the first 20 characters of the text will actually be output to the disk file or the printer.

If you want to output the value contained in a variable, click on the Var button. This brings up the Specify Variable requester (see Figure 8-9), from which you can pick any variable currently defined. When the Output icon is executed in your application, the current value of the variable is sent to the designated printer device or disk file.

If, on the other hand, you want to output a constant text string, you can simply enter the phrase into the Text input field. The contents of this field will then be sent to the disk or printer at the specified column position.

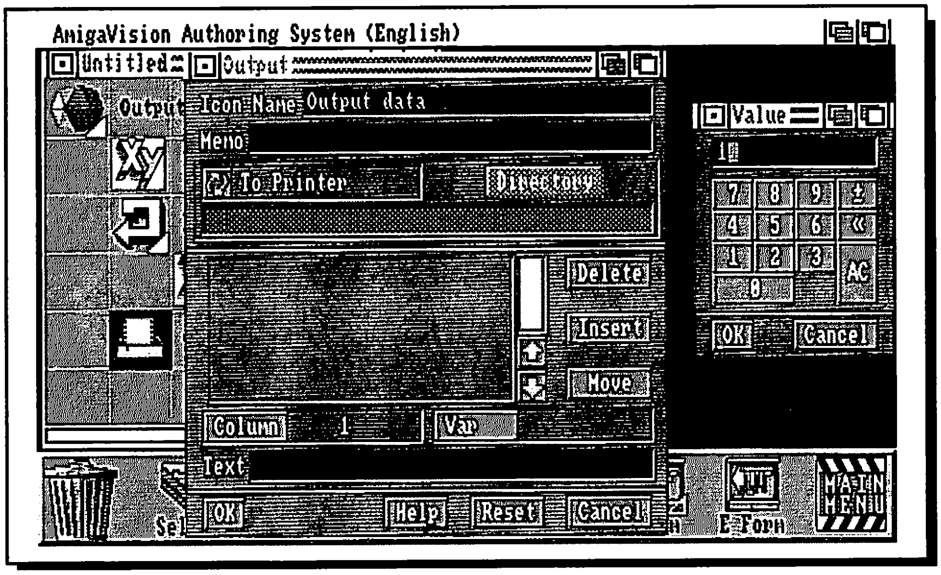


Figure 8-8
The Specify Value requester lets you specify the column in which you want the data defined

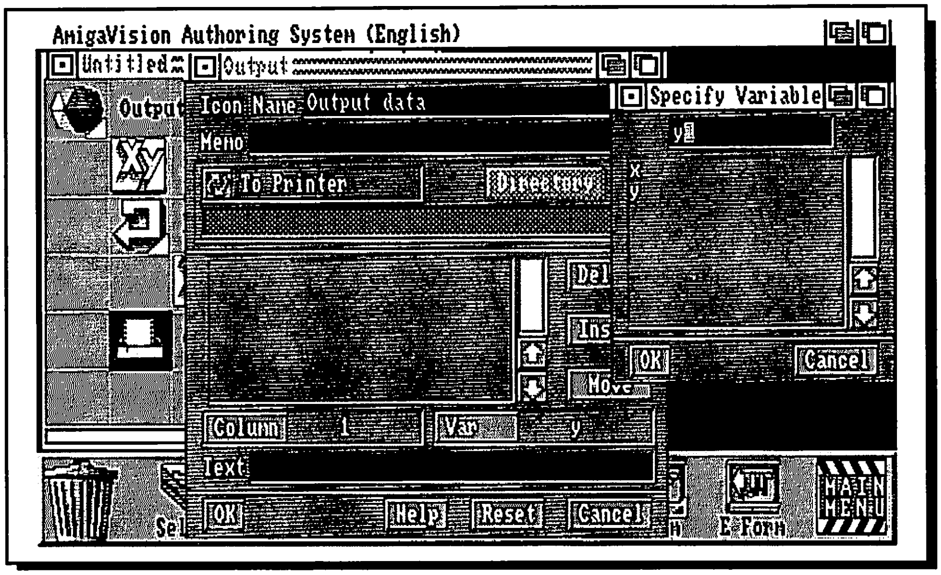


Figure 8-9
The Specify Variable requester appears when the Var button is clicked

Once you have defined the contents of your desired output (either as a literal text string or as a variable) and the column position, you must click on the Insert button to actually place the data into the Output icon's command list. When you select the Insert button, the designated information is placed into the Output requester's List Window at the current position (see Figure 8-10).

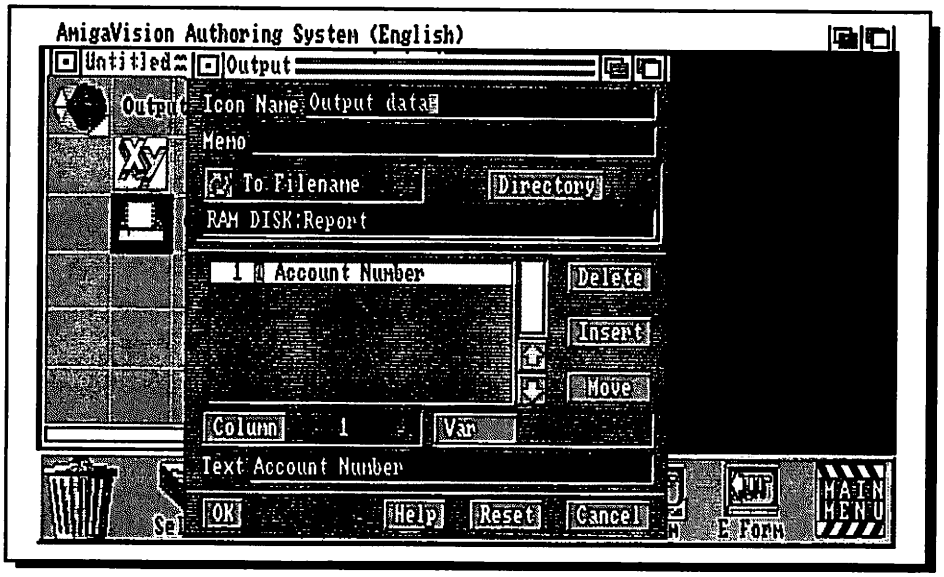


Figure 8-10
The Output icon's List window

The Output icon is more versatile than you might think. It not only lets you output a single string of data, but also allows you to place multiple streams of information on the same line. In other words, you can output multiple data items — four 30-character strings, for instance — on a single line. The trick is to keep track of all the positions, and make sure none of the data items gets truncated or overwritten.

To generate a report sheet with multiple columns of information, you would insert multiple items into the same Output icon's data list. Figure 8-11 demonstrates one such approach, sending the value of a sequence of variables every twenty characters on the line. This example assumes that each field of information will be no longer than twenty characters. If any is, it will either be overwritten by the next string in the sequence, or, in the case of the last string on the line, truncated.

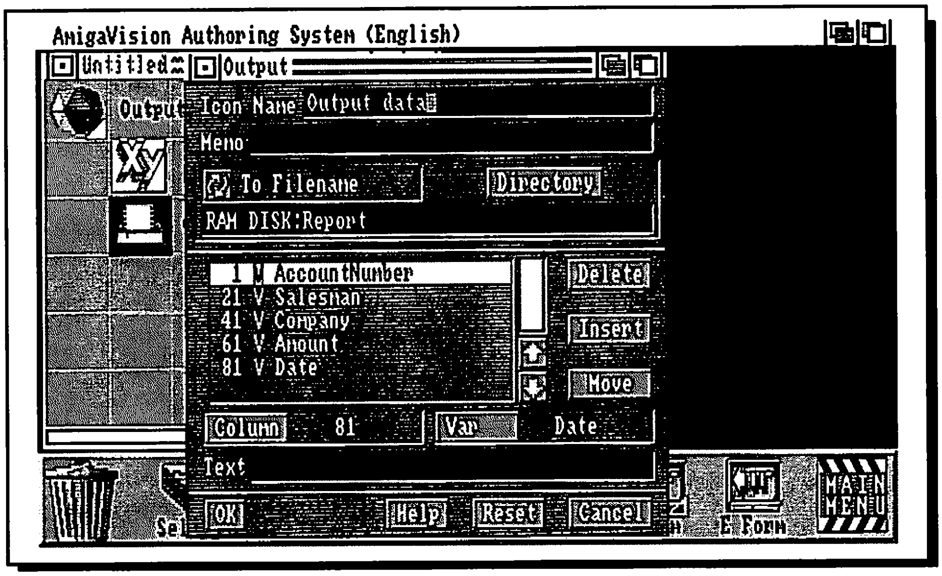


Figure 8-11
Multiple data elements for a single line of output

In cases like this, it is a very good idea to use the strlen() function to find the length of the text string, and if it is too long, to truncate it to the appropriate length using the substr() function. These two functions are discussed in detail in Chapter 13.

Like the Variables requester, the Output icon requester has a List window, which shows the defined components of the line, their type (text or variable), and their column position. You can move an item within the List window by clicking once on it to highlight it, and then clicking on the Move button. Your mouse pointer will change, showing a double-tipped arrow at the bottom, to indicate you are in Move mode. Click on the item you want to switch with the first, and the two will exchange places.

You can use the Delete button to delete the currently highlighted line item. If you merely want to edit the item, double-click on it and the values it represents will appear in the Column, Text, and Var fields. You can then edit the values and insert the new version of the expression into the List window. When you insert an edited expression, the new version does not replace the old; the old version remains, and you must select it and use the Delete button to remove it from the list.

The Select Record Icon

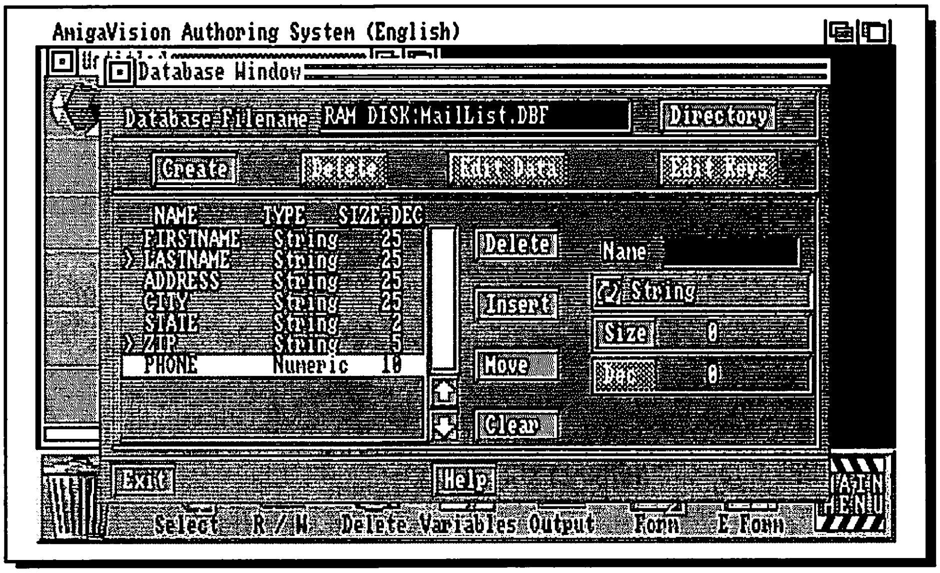
The Select Record icon (labeled Select on the Database Command icon menu) is used for two purposes. The first is to open an existing database file (created with AmigaVision's Database editor, Ashton-Tate's dBASE III, Precision's Superbase Professional, or VersaSoft's dBMAN) in order to access the records in the file. The second purpose is to link data fields within the database to variables created previously, the results of which you can then read into memory or write to the database using the Read/Write Record command.

You can have up to ten different databases open at the same time within an AmigaVision application, but you must use a separate Select Record icon to bring up each one. There is no Close File command — once you open a database, it remains open until the application is finished. When your application quits running, all open databases are automatically closed.

The Select Record icon can function as a sibling or a child of another icon, as the companion of a Partner icon, and also as a Parent icon itself. When used as a parent, the Select Record icon can work in much the same manner as a Counted Loop icon, looping through all the records in an open database sequentially. By properly using defined variables linked to specific key fields, you can also use the Select Record icon to read from specific records in a database.

Figure 8-12 shows the Database editor in action — a mailing list is being defined within it. In this example seven fields have been created, and two of them, Lastname and Zip, are designated as key fields (a key field can be identified by the (>) character next to its name). These key fields are the fields that AmigaVision uses to index or sort the database records. Because Lastname is the first key in the list, it is the primary key index. Zip, then, is the secondary key field. This means that when AmigaVision sorts the records, it sorts them first by the contents of the Lastname field. If there are multiple records with the same value in the Lastname field (if, for example, you have 25 Smiths), the program will sort within this group using the contents of the Zip field. Records having zip codes that begin with lower numbers will be placed before those that begin with larger numbers.

Before you add a Select Record icon to your program, you must have defined one or more variables (using either a Variables icon or the Module icon) that the Select Record icon can use as a link to the fields in each record. In many cases there will be one variable for every field in the database, but this is not always true (see Figure 8-13).

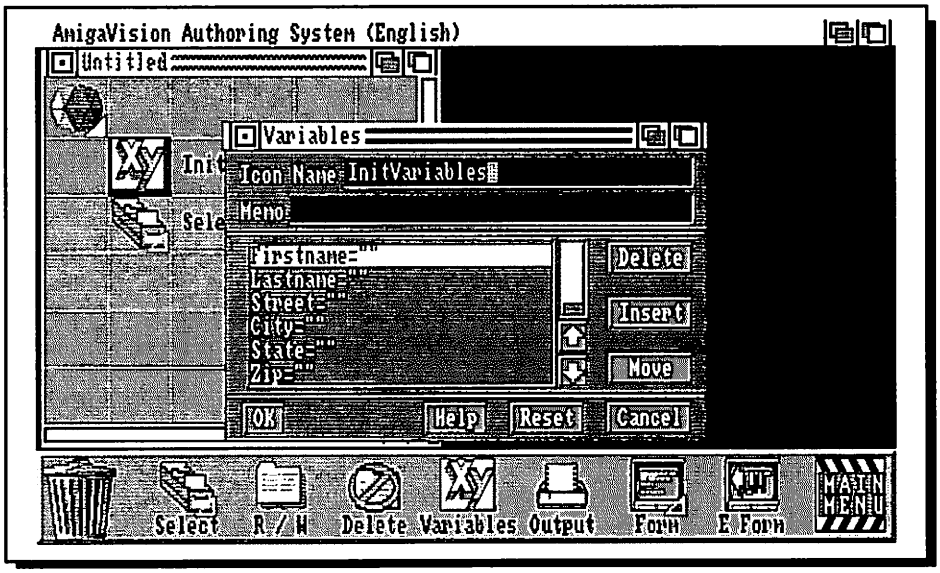


Database Filename: RAM DISK:MailList.DBF

NAME	TYPE	SIZE	DEC
FIRSTNAME	String	25	
LASTNAME	String	25	
ADDRESS	String	25	
CITY	String	25	
STATE	String	2	
ZIP	String	5	
PHONE	Numeric	10	

Name
String
Size 0
Dec 0

Figure 8-12
Creation of the mailing list database



Icon Name: InitVariables
Memo:

```
Firstname=""
Lastname=""
Street=""
City=""
State=""
Zip=""
```

Figure 8-13
Initializing variables to use with a database

Once you have dragged the Select Record icon into the Flow window to the desired position within your application, double-click on it to bring up the Select Record requester. Using the Directory gadget, select the database you want to open and the Select Record requester will display, within the List window, all of that database's defined fields (see Figure 8-14).

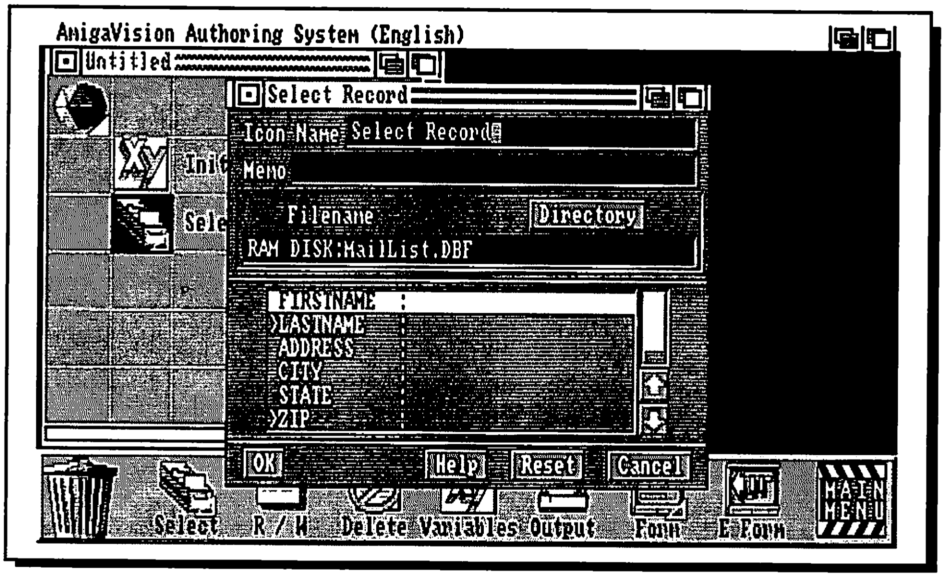


Figure 8-14
The Select Record requester appears when a database from the Directory gadget is selected

The next step is to link the fields in the Select Record requester with some of the defined variables. Click on one of the field names in the List window, and in response, the Specify Variable requester will appear (see Figure 8-15), displaying all defined variables currently available for use.

Click on the name of the variable that you want to link to the selected field. Then click on the OK button and you will return to the Select Record requester. You will see that the variable name has been added to the right of the field name in the List window. Repeat this process of linking field names with variable names until you have selected all the database fields you intend to use, and have placed the appropriate variable names to the right of them in the Select Record requester's List window (see Figure 8-16).

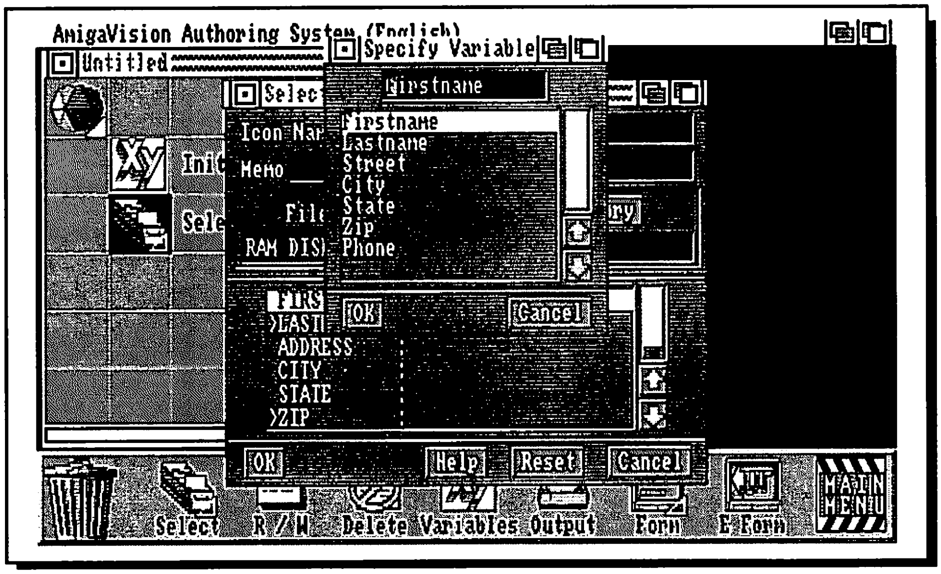
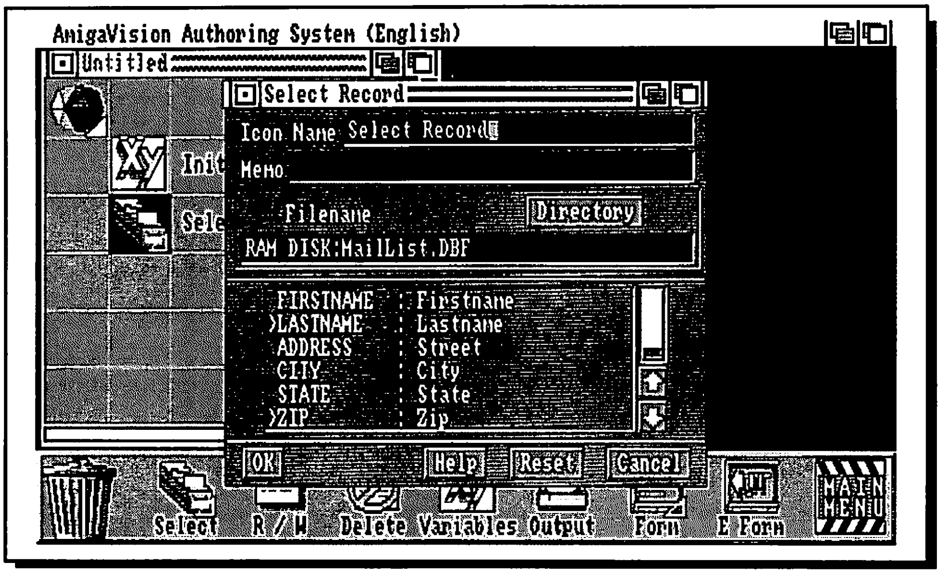


Figure 8-15
Linking variables to database fields



Icon Name Select Record
Memo
Filename Directory
RAM DISK:MailList.DBF

```
 FIRSTNAME  : Firstname
>LASTNAME   : Lastname
 ADDRESS    : Street
 CITY       : City
 STATE      : State
>ZIP        : Zip
```

Figure 8-16
A linked set of database fields and variable names

At this point the database file has been opened (using the Select Record command) and we have linked variables to be used for database input and output to specific fields within it. However, the definitions of the variables prior to linking can have a major impact on which record of the database is selected with the Select Record command.

As mentioned earlier, you can designate some fields as key fields, which you can then use as sort indexes in the database file. AmigaVision examines each new record as you add it to the database and positions the record within the database index on the basis of what is contained in the key fields. If necessary, other records are shifted to make room for the new record. The index is updated to show the position of this new record in the database.

In our mailing list example, there are two key fields, the surname (LASTNAME) and the zip-code (ZIP) fields. When we linked these fields with the variables Lastname and Zip, both had been initialized with a null string value, that is, Lastname="" (with nothing between the two sets of quotes). This had the effect of selecting the very first record in the database. However, if we had defined Lastname as some specific value (such as Lastname="Smith") when the Select Record icon was executed, AmigaVision would have used the key index to find the first record with an entry in the Lastname field equal to "Smith".

Likewise, if we had defined the Lastname variable as Lastname="Smith" and the Zip variable as Zip="03458" when AmigaVision executed the Select Record icon, it would have opened the database and looked for the first record containing the word "Smith" in the Lastname field and the value "03458" in the Zip field.

If the search is successful, any children of the Select Record icon are executed. If no matches are found, however, AmigaVision ignores the children of the Select Record icon and continues on to the next sibling of Select Record.

When using a defined variable as a selection criteria to search for a specific record, keep in mind that the search is case sensitive. In other words, if the content of the field is "Wallace", and you had defined your variable with the value "wallace", no match would be found because the uppercase W is treated as a different character than the lowercase w.

Also, the Select Record command treats the defined variable string as a wildcard, matching everything that starts with the designated characters. For example, if your search criteria is "Lou", the Select Record command would find "Louis" and "Louise" just as readily as it would "Lou". If you want to match only the defined character string, end it with the vertical bar (|) character. So, to find "Lou" and omit "Louis" or "Louise", search using the character string "Lou|".

Finally, you do not have to search only by key fields. AmigaVision's Select Record command allows you to search non-key fields for a specified string (linked to the field via a defined variable containing the string), but this search process is much slower than when a key index is involved.

The Read/Write Record Icon

The Read/Write Record icon (labeled R/W on the Database Command icon menu), can be used to read data from a selected record in the database, to change the contents of an existing record, or to insert an entirely new record into the database. In reading and writing data, AmigaVision addresses the record currently indicated by the Select Record icon. When adding a new record, AmigaVision inserts it into the database at a position specified by the key field indexing; if there is no key field, the record is inserted at the end of the database.

The Read/Write icon can act as a sibling to other icons, as the right-hand companion to a Partner icon, or as the child of a Parent icon. It cannot be a parent to other icons.

To use the Read/Write icon, position it into your flow chart at the appropriate place. (In Figure 8-17, I have made it the child of a Select Record icon.) Then double-click on the icon to bring up the Read/Write Record requester. This requester looks very much like the Select Record requester, except that it has a multistate gadget at the bottom, which cycles between three options: Read Record, Insert Record, and Update Record.

Choose the database you want to access by clicking on the Directory gadget and selecting a name from the AmigaVision file requester. (Remember that if the database has not been opened previously using the Select Record command, the Read/Write commands will have no effect.) Once you have chosen a database, the List window in the Read/Write requester displays the names of the fields in the database in exactly the same manner as the Select Record requester did. Just as with the Select Record requester, you must click on each field name to bring up the Specify Variable requester and select the variable from the list with which you want to link the field. While you can use the same variables as those used in the Select Record requester, it is a better programming practice to use a different set of variables for the Read/Write icon (see Figure 8-18).

In Read Data mode, the Read/Write command reads the data in the selected fields of the current record into the designated variables as it executes. If all the fields are linked to variables, the contents of the entire record will now be in your variables; if you have linked only some of the fields to variables, the unlinked fields are ignored. This allows you to selectively access any part of the record.

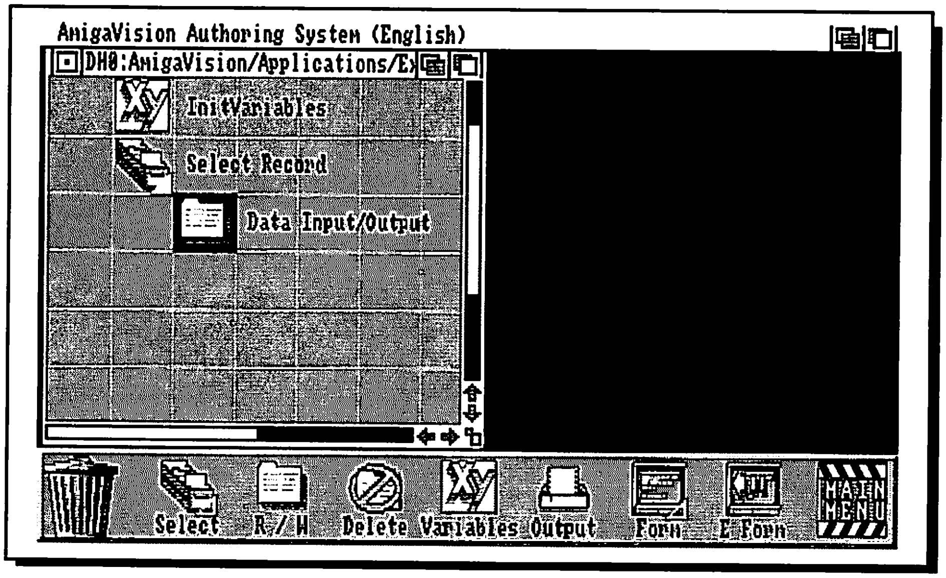
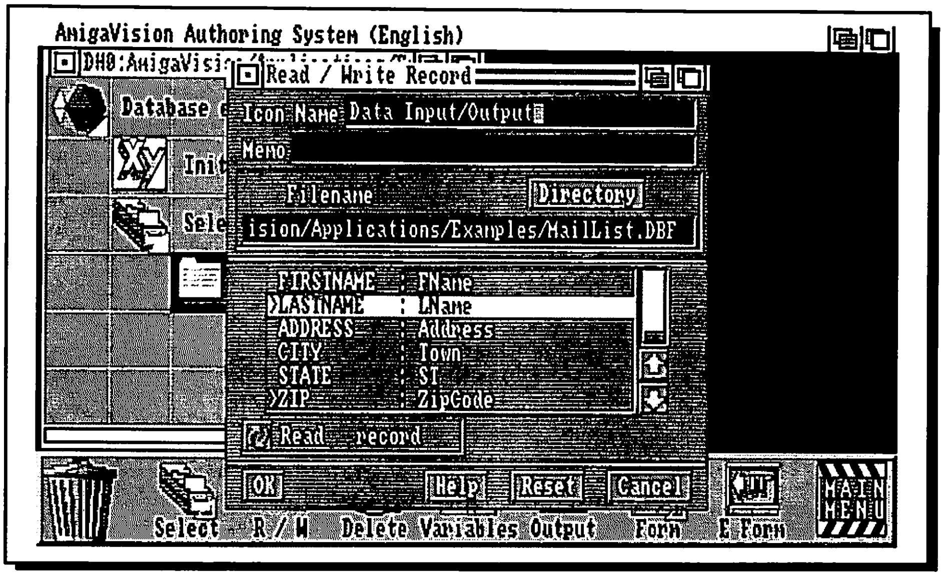


Figure 8-17
Positioning the Read/Write icon as a child of the Select Record icon



Icon Name Data Input/Output
Memo
Filename
ision/Applications/Examples/MailList.DBF

```
 FIRSTNAME : FName
>LASTNAME  : LName
 ADDRESS   : Address
 CITY      : Town
 STATE     : ST
>ZIP       : ZipCode
```

Figure 8-18
Defining the Read/Write Record requester

Once you have the information in variables, you can use it in your program. The types of data that you can store and retrieve are character strings, numbers (including floating-point values), calendar dates, and Boolean (true/false, yes/no) information. In many cases you may only want to review what is in the database, while in other cases you will want to modify the contents of the records. Using AmigaVision's string and mathematical functions, you can edit fields or calculate new values based on the contents of the existing fields, depending on your program's specific needs.

For example, suppose the user of your program discovers an incorrect entry (perhaps a misspelled name) in the mailing list database. The user could delete the old record and insert a new record to replace it, or, if you had added an Edit option to your program, he or she could simply correct the spelling. Using the Form icon's Object editor, you can generate a record-editing screen with input fields corresponding to each field in the record. This would allow the user of your AmigaVision program to make changes; on exiting the Object editor, these changes would be contained in variables.

To allow the user to update an existing record with new information, all you need to do is add another Read/Write command to your flow chart, this time placing it in Update mode by clicking on the requester's multistate gadget until it reads Update Record. Then when AmigaVision executes the Read/Write icon, the contents of the variables linked to the record are placed in the appropriate fields of the current record, replacing what was already there.

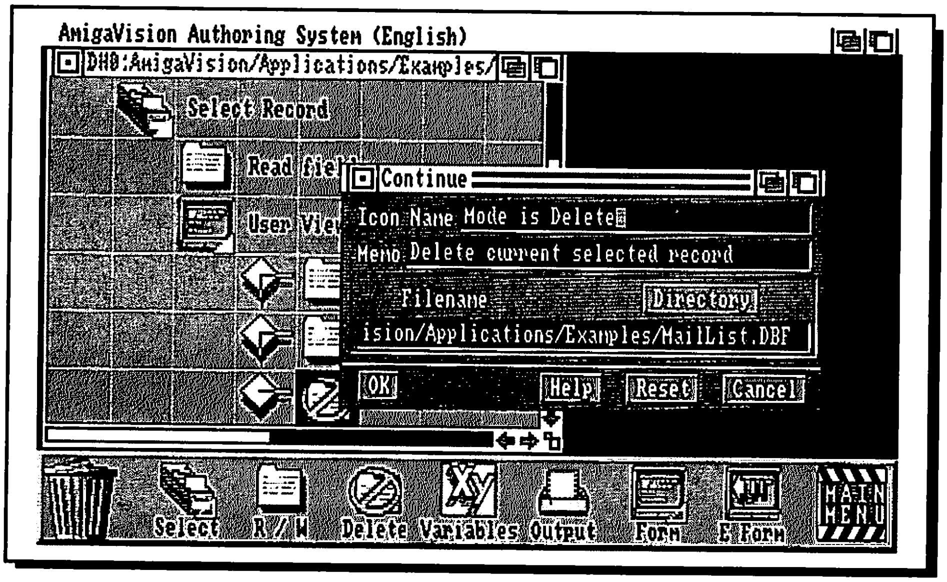
If you want your program's database to be able to accept new information, you can add an option for inserting a new record. Then, your user can easily add information into the input fields of your Forms icon. If you also include an option for saving the new information to the database (this time setting the multistate gadget to Insert Record mode), a Read/Write command will execute when the user selects the Save button. (No matter which mode you choose for the Read/Write command, you must link the fields in the records to a set of variables using the Read/Write requester.)

Because you have already offered the user the ability to view, edit, and add new records, it stands to reason you might want to offer the option of deleting the whole record from the database. For that you will need another icon: the Delete Record icon.

The Delete Record Icon

The Delete Record icon (labeled Delete on the Database Command icon menu) has one function — to delete unwanted records from an existing database. It can function as a sibling, as the companion to a Partner icon, or as a child of a Parent icon; it cannot, however, be a parent itself.

To use the Delete Record icon, place it into position in your flow chart and double-click on it to bring up the Delete Record requester (see Figure 8-19). (The Delete Record requester will be labeled "Continue" — an error that should be fixed in subsequent versions.)



Icon Name: Mode is Delete
Memo: Delete current selected record
ision/Applications/Examples/MailList.DBF

Figure 8-19
The Delete Record requester appears when the Delete Record icon is double-clicked

When executed, the Delete Record command completely deletes, or removes, the currently selected record from the designated database. Like the Read/Write Record command, the Delete Record command works only if you have opened the database using the Select Record command.

Use this command with care, because once deleted, a record (and all the information contained in it) is gone and cannot be recovered. When combined with a Select Record command that uses a defined selection criteria, the Delete Record icon can delete multiple records automatically. For example, if you set the selection criteria to be the name "Jones" in the Lastname field (by entering Jonesl), and if you place the Delete Record command as a child of a Select Record command, AmigaVision will delete every record containing Jones in the Lastname field when it executes those icons.

In most cases it is preferable to allow the user to delete only a single, specified record at a time. In our mailing list example, along with the options for updating an existing record and inserting a new one, we could add an option for deleting the current record. In Figure 8-20, that option has been implemented using an If-Then-Else decision structure. If the user selects the Update mode, the record is updated; if the selected mode is Insert, a new record is inserted; finally, if the selected mode is Delete, the current record is deleted. If the user selects any other option, no change is made to the database and the program moves on to the next record.

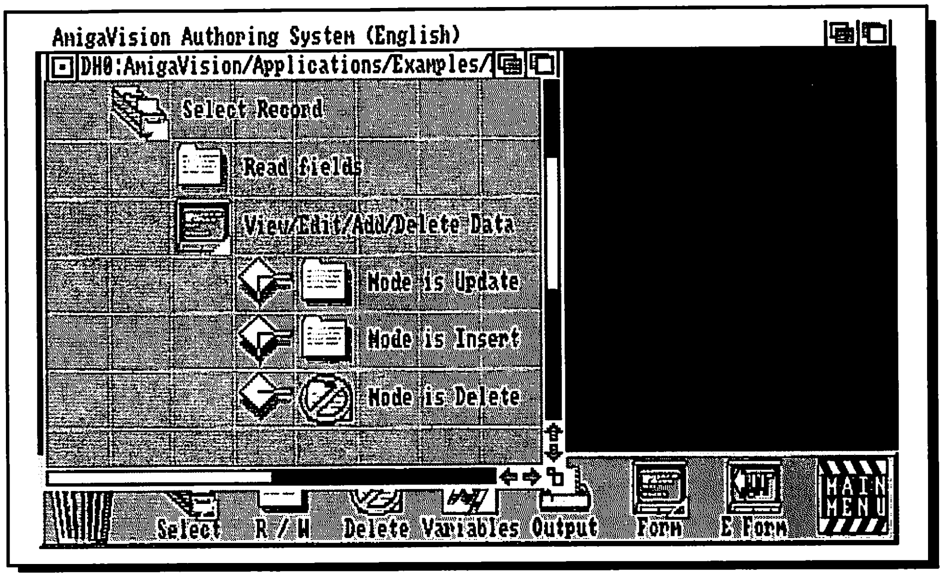


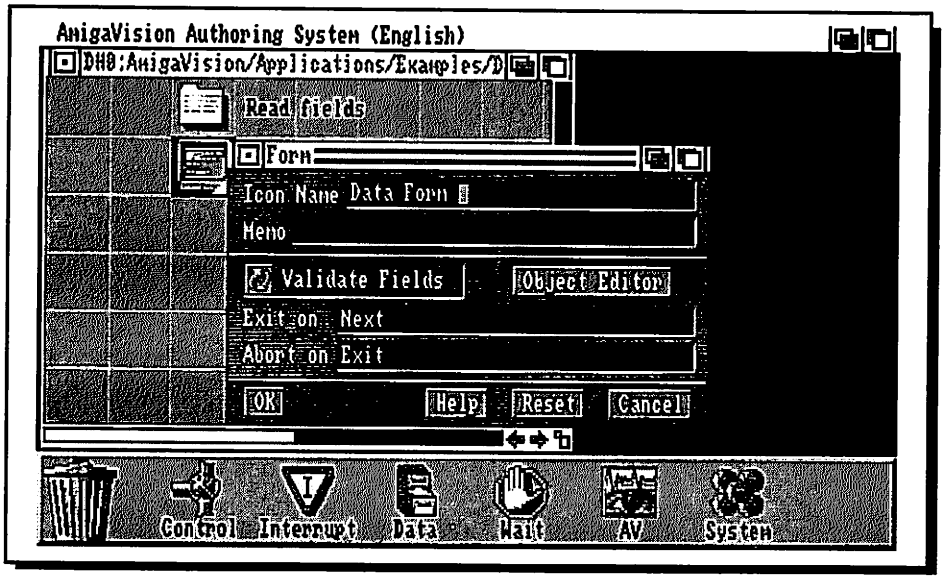
Figure 8-20
Multiple database functions

The Data Form Icon

The Data Form icon (labeled Form on the Database Command icon menu) is designed to take user input from the keyboard. By using the Object editor, you can create templates that allow the user of the application to type in numbers, text strings, dates, or logical responses, which are then automatically transferred to variables for further manipulation by the program. This information can also be stored in a database, and in fact, the Data Form icon is the only means of allowing the user to enter data (outside of single-character input) directly into an Amiga-Vision application. Because data input is critical to the functionality of a database, the Data Form icon plays a key role in database design and implementation.

The Data Form icon can act as a sibling, a child, a companion to a partner, or as a Parent icon. You cannot nest one Data Form icon under another Data Form icon, however, as the outer icon takes precedence over any inner ones. By using the Data Form icon as a parent, you can have icons that are executed until Amiga-Vision exits the form, which you can limit to specific circumstances.

Drag the Data Form icon into your flow chart, placing it at the appropriate position in the program. Double-click on the icon, and the Data Form requester will appear (see Figure 8-21). This requester offers the standard Name and Memo fields, and along the bottom, the usual complement of OK, Help, Reset, and Cancel buttons.


AmigaVision Authoring System (English)

DH0:AmigaVision/Applications/Examples/D

Form

Icon Name Data Form

Memo

Validate Fields

Exit on Next

Abort on Exit

Figure 8-21

The Data Form requester appears when the Data Form icon is double-clicked

The Data Form requester contains one multistate gadget, which allows you to toggle between the Validate Fields and Validate On Exit modes. It also provides two text fields, one labeled Exit On and the other labeled Abort On. There is also an Object Editor command button, which allows you to transfer to the Object editor, wherein you further define the Data Form icon.

When accessed from the Data Form icon, the Object editor works as it does from other icons — with one exception. In this case, there is one extra object you can add to the display: the input field (see Figure 8-22). (When using the Object editor from other icons, this option is normally ghosted, or unavailable.) The input field allows the user to type in a line of text, a number, or some other data type. Each input field is linked to a variable that you select, and when the user finishes typing and presses the Return key, the designated variable is updated with the information found in the field.

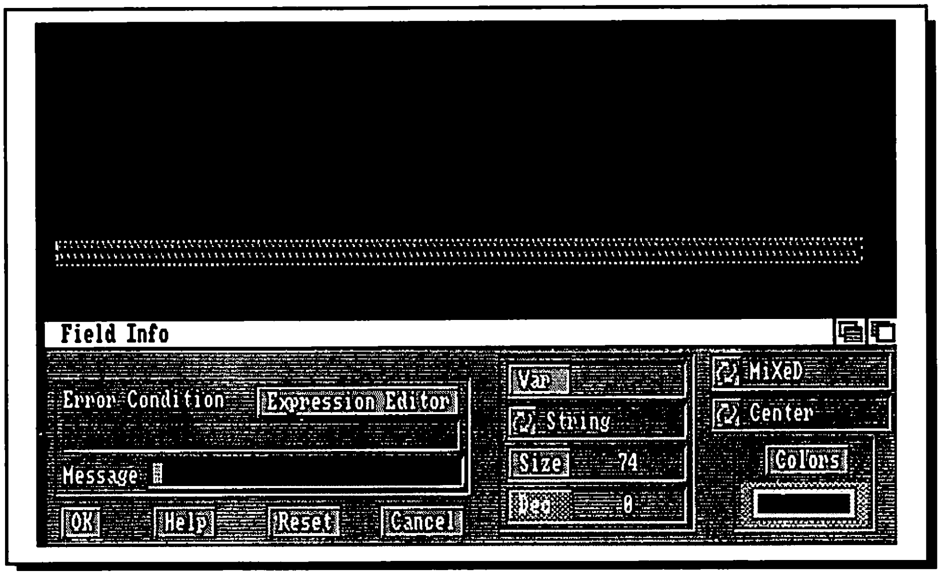


Figure 8-22
Object editor options

If you are designing a database (or other application) that requires user input, you can create an interface containing a series of input fields into which the user can enter information. Because the display can contain other objects as well (rectangles, circles, ellipses, polygons, text, and brushes), all of which can return a defined response to the program when the user clicks on them, you can create an entire user interface complete with hot spots, buttons, text, sounds, and, of course, input fields. Figure 8-23 is an example of a database interface that I put together quickly using the standard Object Editor drawing commands. While it certainly wouldn't win any design awards, it did help me accomplish my main goal of programming a small database. By using the default graphic objects, I was able to create a functional database program in one session without having to interrupt my programming to generate complex graphics.

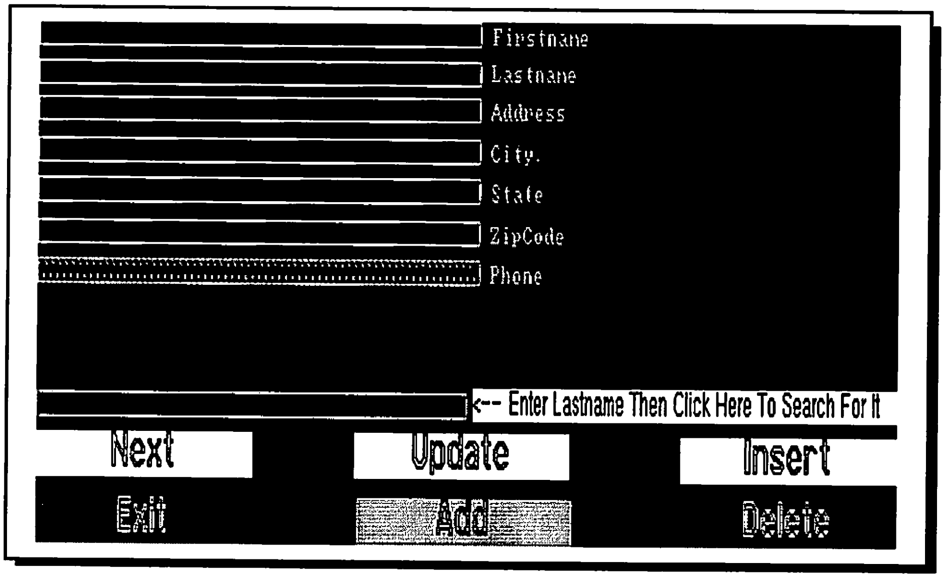


Figure 8-23
A simple interface created in the Data Form icon's Object editor

The user interface in Figure 8-23 consists of eight input fields and a series of command buttons. The seven input fields at the top left of the screen are linked directly to variables used with the Read/Write Record command. When a user enters information into the fields and presses the Return key, the variables are updated. The database itself does not change, however, until the user explicitly selects Update or Insert; choosing either of these options causes the program to then write the information in the linked variable fields to the database file using a Read/Write Record icon.

The eighth input field is used to define a search string. When the user enters a string and clicks on the Search hot spot, the program instructs AmigaVision to perform a search (using the Select Record command) for the first occurrence of that string in the Lastname field, which I had previously defined as a key field when creating the database. This input field is also linked to a variable, but in this case, the variable is not tied to those already linked to the database fields.

To link a variable to an input field, select the Input Field option from the Add submenu of the Objects pull-down menu in the Object editor. This gives you a cross-hair cursor that moves with the mouse. Position this pointer on the screen where you want to place the upper-left corner of the input field and, while holding the left mouse button down, drag open a box that is long enough to hold the longest string you might need. Release the left mouse button and you will see a rectangular box one character high. This is your new input area.

Once you have created an input field you must define it. Double-click on the input field and a new requester, the Field Info requester, will appear (see Figure 8-24). The Field Info requester allows you to define the characteristics of the input field, link specific variables to it, and select the type of data and the format it will accept.

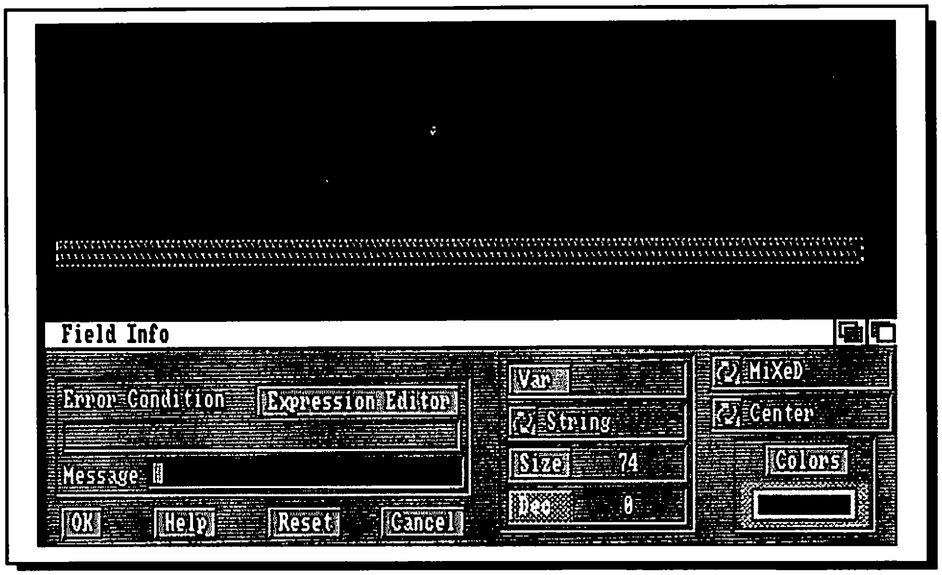


Figure 8-24
The Field Info requester lets you define an input field

As you can see in Figure 8-24, there are quite a few options available in the Field Info requester. The first thing to do here is to link the input field to a variable. Click on the command button labeled Var to bring up a Specify Variable requester (Figure 8-25). Select one of the variables from the list, and then click on the OK button to return to the Field Info requester.

Once you have selected a variable, use the multistate gadgets in the requester to select the data type (string, numeric, date, or Boolean), the type of text (uppercase, lowercase, or mixed) and how it is to be placed in the input field (flush left, flush right, or centered). You can also adjust the length (in characters) of the input you will allow by clicking on the Size button. This brings up another Specify Value

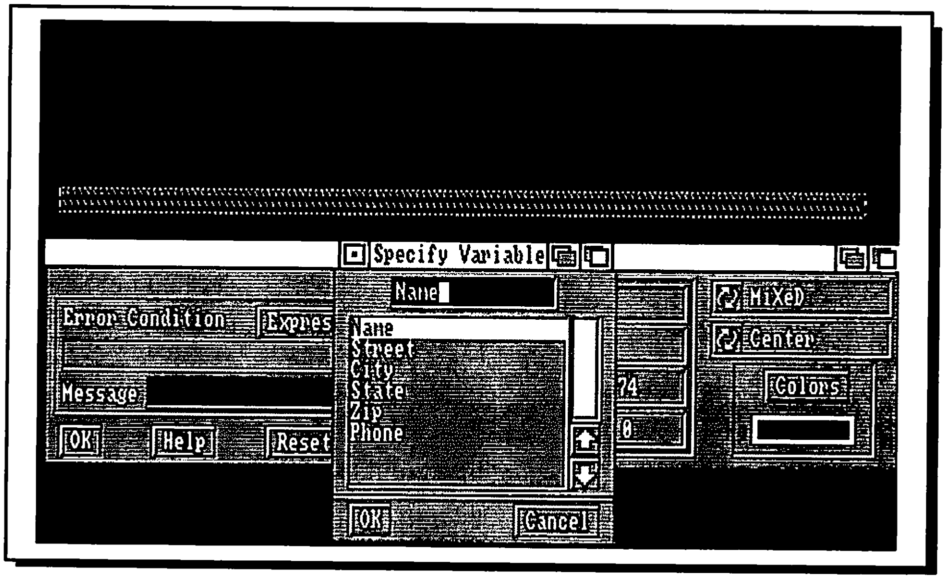
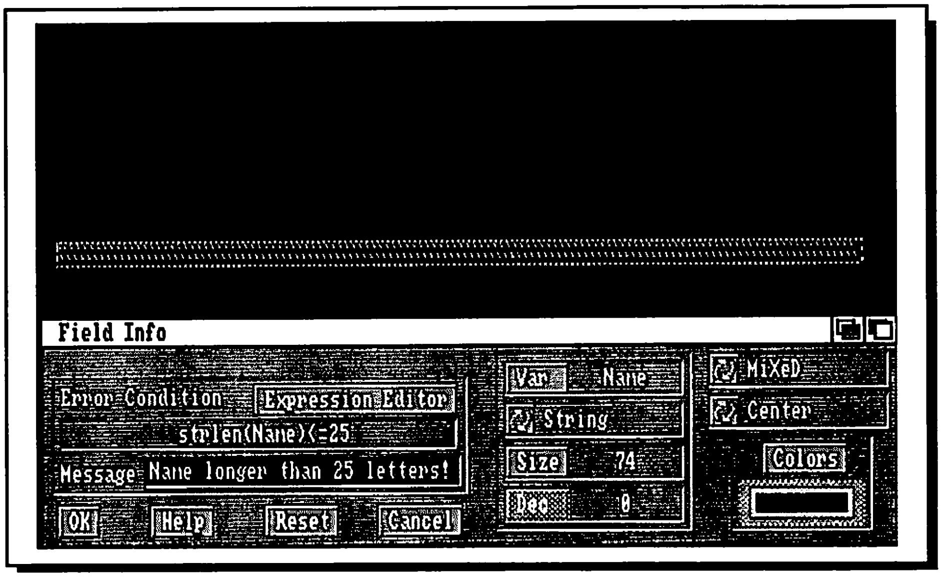


Figure 8-25
Selecting a variable to link to the input field

requester into which you enter the maximum number of characters (up to 255). If your data type is numerical, you can also select the Dec button to determine how many decimal places (up to 19) will be allowed. Using the Color gadget, you can determine the color of the border surrounding the input field, or even specify that it have no border. Unfortunately, you cannot alter the color of the text displayed in the input field.

Another very important option is the Error Condition. This allows you to place restrictions on what the user enters into the input field. For example, if the input field is being used for numeric data, you can set upper and lower ranges for the numeric values. To use the Error Condition option, click on the Expression Editor button and (using the supplied functions) create an expression to be used in the evaluation. Then, when the user enters information into the input field and presses the Return key, the information is evaluated based on the expression you defined. If the results evaluate as true — that is, if the user's input fits into your defined limitations — everything is fine and the cursor moves on to the next field (if there is one). If, on the other hand, the results evaluate as false — that is, if the user types in an entry that does not fit into your range — the cursor stays put and, if you had entered a phrase into the Message Text field below the Error Condition option, that text is then displayed on screen. By defining specific Error Conditions and setting up a custom error message in the Message Text field, you can trap most unwanted user input and inform the user of the mistake she or he has made. An example of a fully defined Field Info requester is shown in Figure 8-26.


Field Info

Error Condition [Expression Editor]
strlen(Name)<=25
Message Name longer than 25 letters!
[OK] [Help] [Reset] [Cancel]

Var Name
String
Size 74
Dec 0

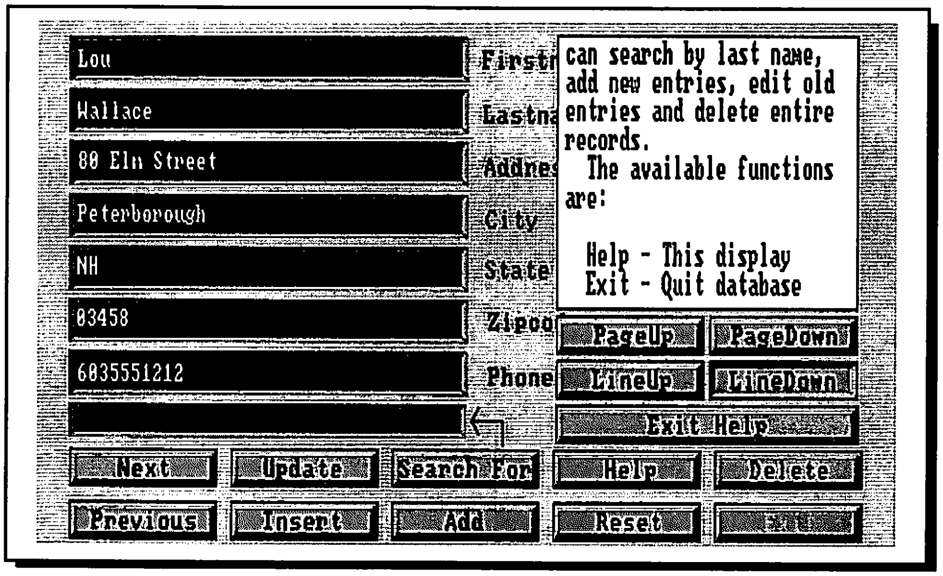
MiXeD
Center
Colors

Figure 8-26
A defined Field Info requester

To explore the ways in which hot spots can be used to control the Data Form icon, let's return to our example database. In this case, if a user wishes to exit the database, he or she must click on the Exit button. That's because in the Data Form requester, I had entered specific conditions on what would allow the program to exit the Data Form: I had indicated a response string of Exit in the Data Form requester's Exit On text field. Until that response is detected, the program stays at the Data Form input screen. As you might have guessed, the Exit button is a hot spot that returns a response of "Exit" when the user clicks on it. Each of the other buttons on the screen are also hot spots that return defined responses to the program and thus cause some action when they are clicked upon.

As I mentioned earlier, I decided to use the standard built-in graphic objects for the database form while I was in the process of actually programming the database. After I finished the programming task, including testing and debugging, I could have gone back to the Object editor and taken a little more time drawing and implementing the display. Instead, I decided to use a professional graphics program (DeluxePaint III, by Electronic Arts) because that program offers a much better set of drawing and rendering tools. With a little patience and work, I created a reasonably attractive bitmapped display for the input screen, complete with brushes that look like three-dimensional beveled-style buttons that change when the user clicks on them to appear depressed. Going back to the Data Form's Object editor, I then redesigned my user interface, adding the bitmapped graphics I created with DeluxePaint III (see Figure 8-27). The result was a program that worked almost exactly like the original but had a more polished, professional appearance, including an on-line Help facility that pops up when the user clicks on the Help button.

The whole program development process went much more smoothly because I used the default graphics routines available in the Data Form icon's Object editor while programming. Only when the program was finished and working did I bother to create custom graphics. This ability to design prototype interfaces while programming is a valuable feature of AmigaVision.


Lou Firstn
Wallace Lastna
88 Elm Street Addres
Peterborough City
NH State
03458 Zipco
6035551212 Phone

can search by last name, add new entries, edit old entries and delete entire records.
The available functions are:

Help - This display
Exit - Quit database

Figure 8-27
The database interface redesigned

Now that we have a better idea of what an input field is and how it works, let's return to the Data Form requester and discuss the significance of the remaining gadgets and fields.

The Data Form requester's multistate gadget switches the commands mode from Validate On Exit to Validate Fields. These options allow you to determine exactly when the program will check the error conditions set in the Field Info requester. If you select Validate Fields, the program will check each field as soon as the user presses the Return key. If you choose the Validate On Exit mode, however, the program does not check the field's contents against the error condition until the user exits the form.

When and how a user can exit a form is up to you. In my example, I have opted to allow exit from the form when the action of the user returns a response string of "Exit". I determined this by entering that string into the Data Form requester's Exit On field (you can use almost any response as long as you define it there). If you do not enter anything in the Exit On field, the form is exited as soon as the user finishes with the last input field. In other words, if you have no Exit On response defined, and your data form has six input fields, then the person using your program will exit the form as soon as he or she presses the Return key after making an entry in the sixth field.

You can also define a response to let the user out of the form without changing the variables linked to the input fields. To do this, specify an Abort On response. Like Exit On, this option returns a response to the program when the user clicks on a hot spot that you defined within the Object editor. When the user does click on the hot spot, any changes made to the variables are ignored; the program will then consider the variables equal to the values they had when the user first entered the Data Form.

The Form Exit Icon

AmigaVision offers another method of exiting a Data Form icon. This is the Form Exit icon command, (labeled E Form on the Database Command icon menu), which works much like the Loop Exit command. You can use Form Exit to program alternate conditions of exiting a Data Form besides those contained in the Exit On and Abort On fields.

The Form Exit icon *must* be a child of a Data Form icon. It can function as the right-hand companion of a partner-requiring icon that is also a child of the Data Form icon, or as a sibling of other children of the Data Form icon.

As a child of a Data Form icon, the Form Exit icon causes an unconditional exit from the Data Form icon (see Figure 8-28). In this example, an If-Then command evaluates some expression. If the expression proves to be true, the partner is executed. In this case the partner is a Form Exit icon.

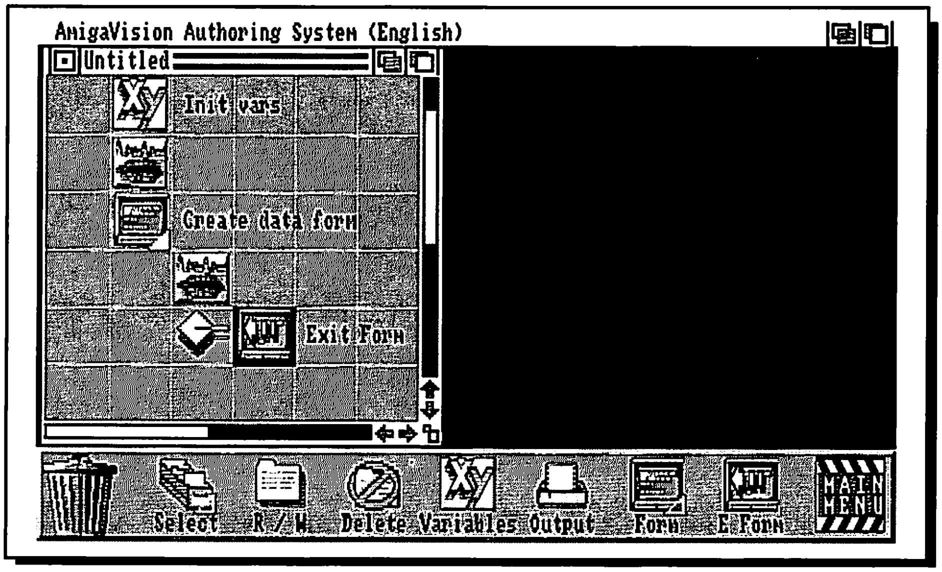
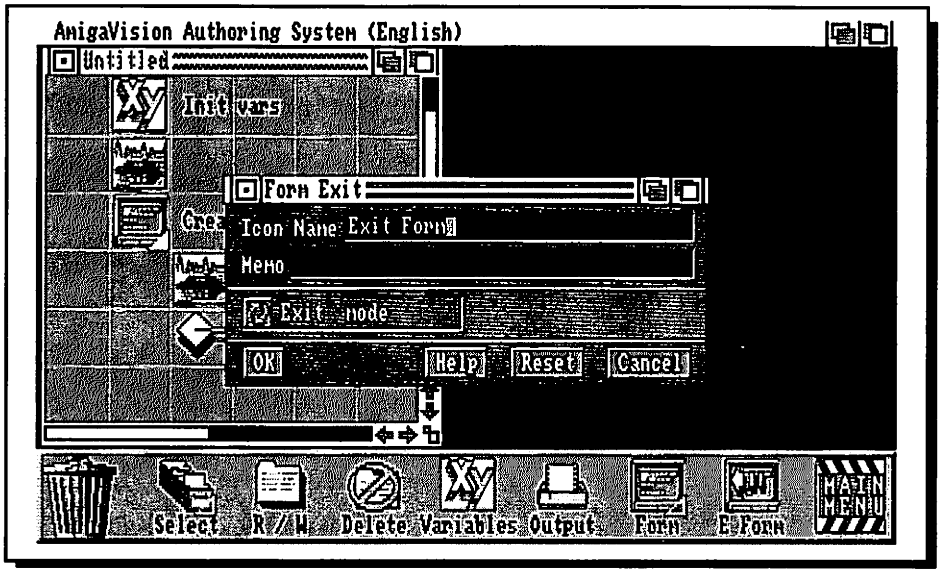


Figure 8-28
Using the Form Exit icon

The Form Exit icon has its own requester, which appears when you double-click on it (see Figure 8-29). Its only significant option is a multistate gadget that toggles the exit mode between Exit and Abort.

These two modes work in much the same manner as the Exit On and Abort On options in the Data Form requester. Exit mode exits the Data Form icon, and any changes made to the variables in it are maintained. Abort mode also exits the Data Form, but any changes made to variables within it are ignored.



Icon Name: Exit Form
Memo

Figure 8-29
The Form Exit requester appears when the Form Exit icon is double-clicked

– 9 –
The Wait Command Menu

One of the most important aspects of AmigaVision is the high degree of user interactivity it allows in the applications it generates. The key to that interactivity is the set of five icon commands found under the Wait menu. (The Wait menu appears on the Main icon menu as an upraised hand.)

The Wait Command Icons

The five command icons that make up the Wait menu are Delay, Condition, Mouse, Keyboard, and Grouped Wait (see Figure 9-1). Two of these icons allow the user of your application to select options and control events using both the keyboard and the mouse. The remainder cause the program to pause until a certain amount of time has passed, until some specific condition exists, or until the combination of several specific events occur. Deciding which of these commands to use in a given situation is the key to successfully implementing interactive (and time- and event-based) applications.

The Delay Icon

The simplest of the Wait menu icons is the Delay icon. This icon is used to pause program execution for a defined amount of time, measured in seconds. Just drag the Delay icon from the Wait menu into the flow window and position it in your program as a Sibling icon, as a child, or as the companion of a Partner icon. The Delay icon itself cannot support children.

Despite what your AmigaVision manual states, the Delay icon should *not* be used to pause the program while a user reads informational text or examines a graphic

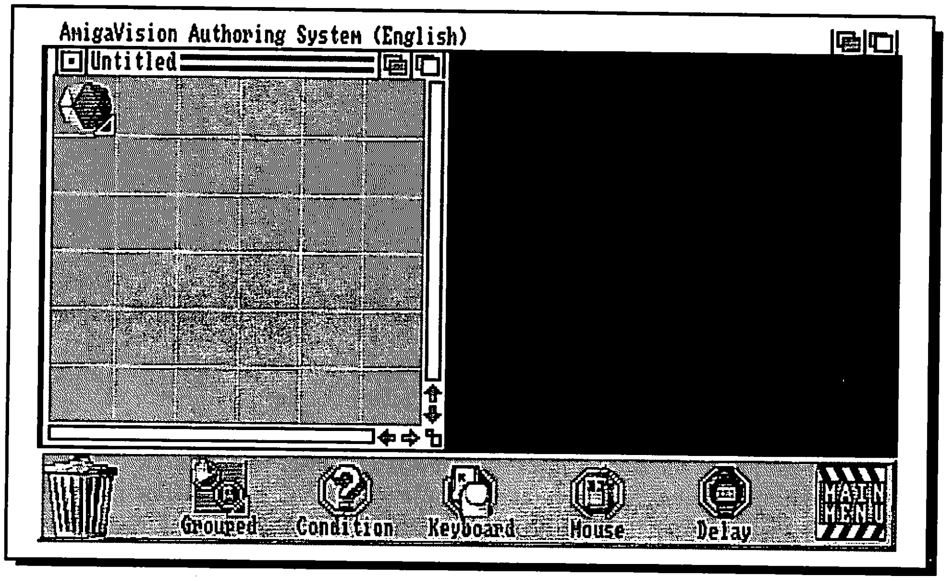


Figure 9-1
The five command icons that make up the Wait menu

screen. This is an extremely poor technique to use for this purpose, because different users read at different speeds; no matter which value you choose, some will be forced to wait longer than they need while others will not have enough time. A better technique for these situations is to use a Wait Mouse or Wait Keyboard command (or both via the Grouped Wait), so the user can pause and then move on as soon as she or he is ready. This is especially important for software that one person will run multiple times; as the user becomes familiar with the Help screens and other instructions, he or she will want to pause only briefly and immediately move on to the main program. A good example of an appropriate use for the Delay icon is a self-running slide show or other demo that does not require user interaction.

To use the Delay icon, double-click on it to open the Delay requester (see Figure 9-2). This requester has only one functional option beyond the standard Name and Memo fields, and the OK, Help, Reset, and Cancel buttons. That feature is the Delay command button.

Click on the delay button and a Specify Value requester appears. You can specify the length of delay here by entering numbers. This requester reads units as seconds. So, to define a delay of one minute, or 60 seconds, you would enter a value of 60; to create a ten-minute delay, you would enter 600.

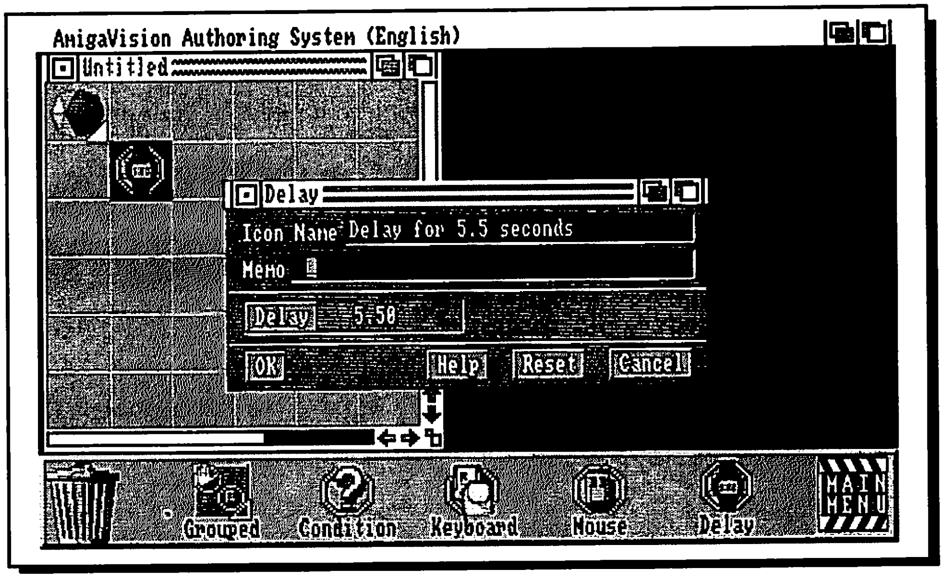


Figure 9-2
The Delay requester appears when the Delay icon is double-clicked

You are not limited to integer values because the Specify Value requester allows decimal fractions. Only two decimal places are significant, so as far as Amiga-Vision knows, a request to wait for 0.309 seconds is no different than a request to pause for 0.30 seconds. If you enter more than two digits after the decimal point, AmigaVision will simply truncate the extras.

Note that on versions of AmigaVision earlier than 1.53G, it is very important not to use zero as a value in the Delay icon. Those early versions of AmigaVision treat that value as infinity, so if your program encounters a solitary zero in the Delay command, it will literally pause forever (or until the user of your program turns the machine off!) Beginning with version 1.53G, AmigaVision treats a zero value correctly, which is to say that it reads zero as no delay at all.

Wait Condition Icon

The next Wait command we will examine is the Wait Condition icon. This command is used to pause program execution until a certain defined condition occurs. You can define that condition using the Expression editor.

The Wait Condition command can be a Sibling icon, a child of another icon, or the companion of a Partner icon. It cannot be a parent.

To use the Wait Condition icon, drag it into the Flow window and position it in your program. Then double-click on the icon to bring up the Wait Condition requester (see Figure 9-3). Beyond the features standard on most requesters, this one offers two command buttons that allow you to define an expression to look for and a maximum time to wait for that expression to appear.

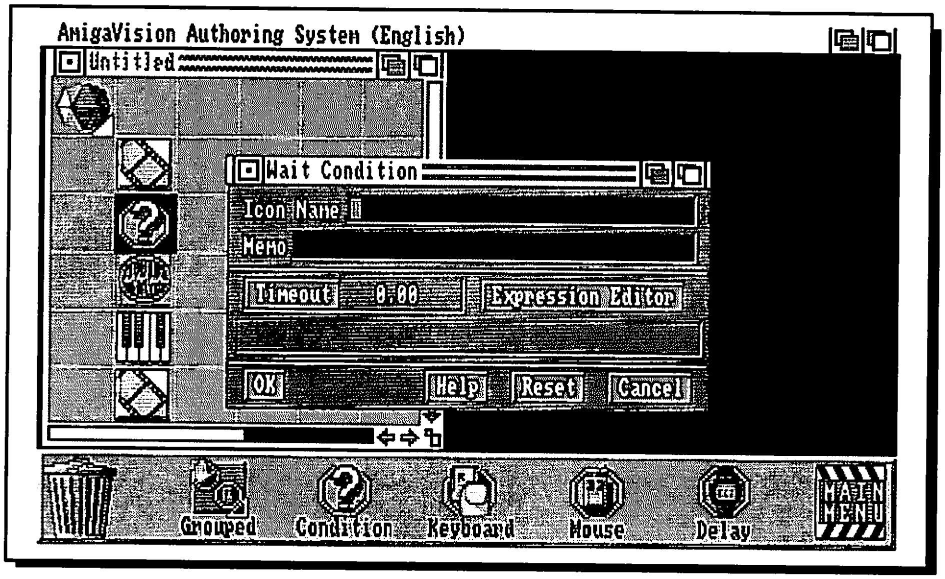


Figure 9-3
The Wait Condition requester appears when the Wait Condition icon is double-clicked

Clicking the Expression Editor button brings up the Expression Editor requester. Here you create the logical expression that the Wait Condition icon evaluates. As long as this expression evaluates as false, program execution remains halted at the Wait Condition icon.

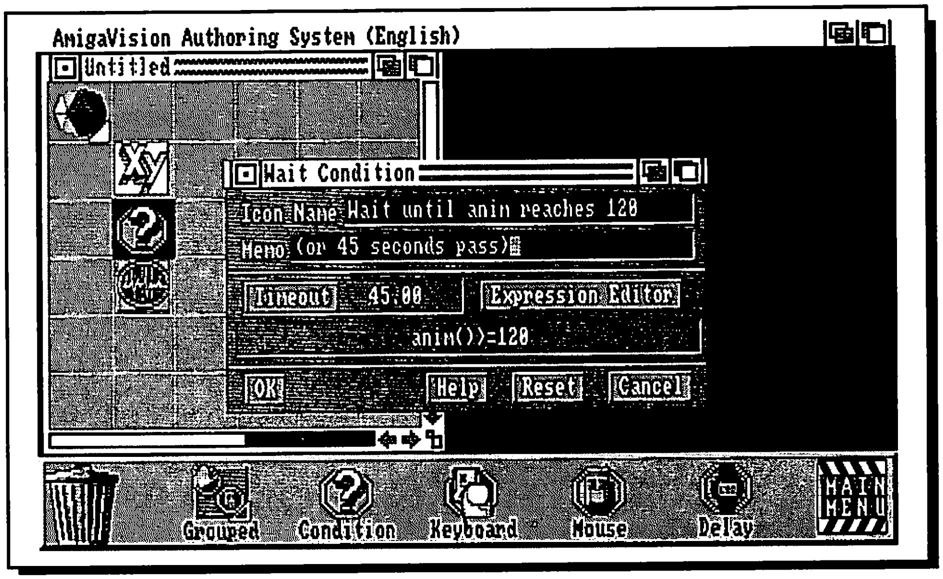
What kinds of expressions can (or should) you use to pause the program? While there is an almost infinite number of possible situations for its use, I will mention a couple of examples. One example is a situation where you want to wait until the program reaches a certain frame of an animation or video. Using an animation as an example, you might enter this expression:

```
Anim() >= 120
```

Entering this expression in the Wait Condition icon's Expression editor would cause the program execution to halt until the animation had reached or exceeded frame 120 (see Figure 9-4). Entering the following expression when using a videodisc system would have a similar effect:

Video() >= 25100

This would pause program execution until the video disc had played to at least video frame 25100.



Icon Name: Wait until anim reaches 120
Memo: (or 45 seconds pass)
Timeout: 45:00
anim()>=120

Figure 9-4
A defined Wait Condition requester

The trick to using the Wait Condition command is to select an expression that *can* become true. It is easy to create some that will never become true. For example, consider this one.

ANIM() < 0

This expression will never evaluate as true because no animation can ever reach a frame number that is less than zero!

A more common error would be one where some variable is being monitored, for example:

X == 1

Note that this logical expression uses the double equals sign, which asks whether one side equals the other. This expression could not become true, because it is waiting for the value of variable X to change to 1. When program execution is halted, it is impossible for the value of a variable to change, unless an interrupt event occurs that modifies the variable. Thus, when you use the Wait Condition icon command keep in mind that unless the expression you use can become true sometime, your program will just hang when it comes to this command.

A way out of these situations is to define a maximum time period for the command to wait. To do this, click on the Timeout command button. A Specify Value requester will appear, allowing you to enter the number of seconds to wait. If this amount of time elapses and the condition has not yet evaluated as true, the pause is aborted and program execution continues.

Even in AmigaVision version 1.53G, you should not set the Timeout value to zero (which will cause an infinite wait to occur in the Wait Condition command itself) unless you are absolutely sure that the defined expression will become true at some point.

Wait Keyboard Icon

The Wait Keyboard icon command allows you to create truly interactive applications. It is used to pause a program until the user presses a key, and gives you the option of selecting the specific key or keys to wait for. In addition, you can access the Object editor and all its graphic functions from this icon.

The Wait Keyboard icon can be a Sibling icon, a child of a parent, or the companion to a Partner icon. It cannot act as a Parent icon itself.

To use it, drag the Wait Keyboard icon into your Flow window and position it at the desired location in your program. Double-click on the icon and the Wait Keyboard requester will appear. Besides the common buttons and fields found in nearly all requesters, this requester contains six features that allow you to specify exactly which keys you want to check for, and how long to monitor them. Also, this requester provides an option to display additional graphics while waiting for the user input (see Figure 9-5).

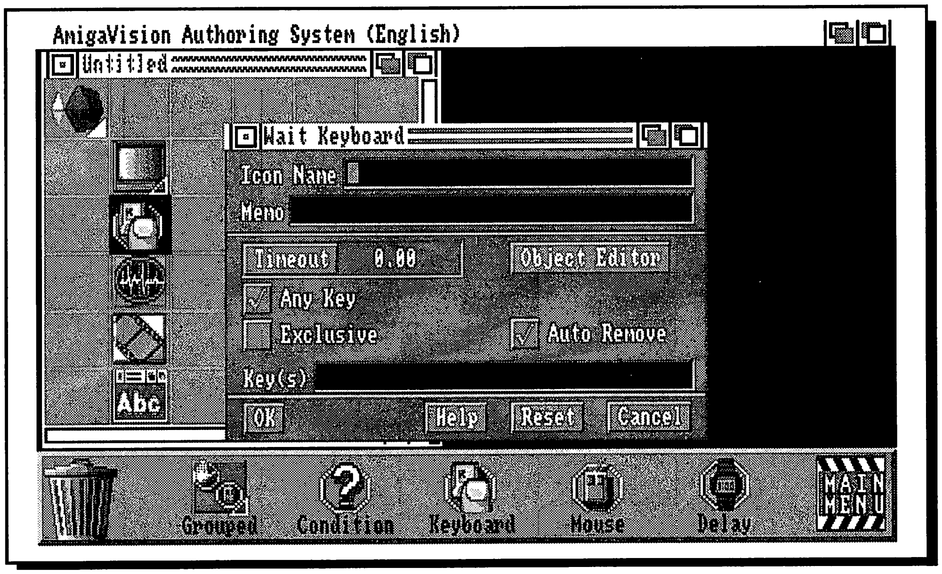
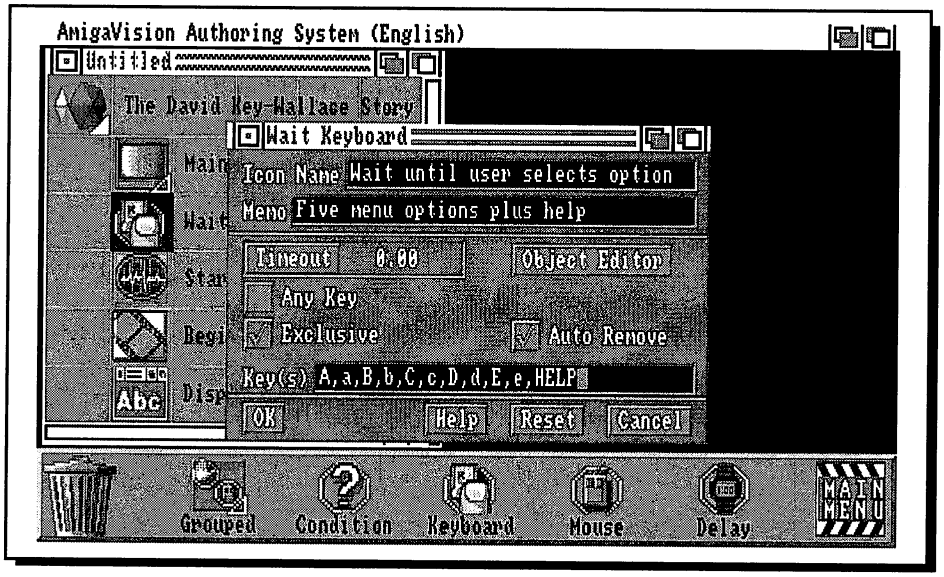


Figure 9-5
The Wait Keyboard requester appears when the Wait Keyboard icon is double-clicked

Two of these options are represented by check-box gadgets, which allow you to toggle these features on and off by clicking with the mouse. These two options are Any Key and Exclusive. They are not mutually exclusive; in other words, you can have both set to the "on" position at the same time. When either of these options is enabled (on), its button displays its characteristic check mark; when disabled, the button appears as a blank box.

Any Key indicates that the program should wait until the user presses a key — any key — on the keyboard. This is useful when there are no specific selections to be made and you just want the user to indicate that she or he is ready to continue. With this option enabled, once AmigaVision executes the Wait Keyboard icon, the program just waits until someone presses a key, and then continues on its merry way.

A similar option is the Exclusive feature, which allows you to indicate *specific* keys that will allow program execution to continue. Figure 9-6 shows a Wait Keyboard requester that has been defined to use a specific set of keys. Notice that both lowercase and uppercase letters have been entered. This was done this way so that when the user presses a correct letter, the proper action will occur, regardless of whether the Shift key is also depressed. To define a specific key just enter it into the Key(s) text field.



Icon Name: Wait until user selects option
Memo: Five menu options plus help
Timeout: 0.00
Key(s): A,a,B,b,C,c,D,d,E,e,HELP

Figure 9-6
A defined Wait Keyboard requester

Suppose you specify a key in a Wait Keyboard icon and you do not remove it (by using the Auto Remove feature). This specified key will still be active during your next Wait Keyboard icon, even though you didn't actually list it in the current Key(s) field.

When a Wait Keyboard icon has Exclusive turned ON, only the keys listed in its Key(s) field are active. Keys specified previously, even in Keyboard Interrupts, are not active.

Besides the alphanumeric keys, you can also indicate some of the special keys on the keyboard. These include the Function keys (F1-F10), the Escape (Esc), Help (Help), Tab (Tab), and Delete (Del) keys, the Cursor keys (Up, Down, Right, Left), and the Return key (Return). To use these keys, just spell out their names (as indicated in parentheses) in the Key(s) field.

Note that you are not limited to waiting for just one key. If you need to watch for a series of keys, just enter them into the Key(s) field, separating them with commas. This includes the special keys as well. The limit to the number of keys that can be monitored is strictly a function of the number of characters allowed in the Key(s) field. This field allows 79 characters, so if you are using single keys, you can define a total of 40 keys separated by commas. Of course when you use any of the nonalphanumeric keys, all of which require a definition of more than a single character, you reduce the total number of keys that will fit in the Key(s) field.

Because the two selectable Wait conditions are not exclusive, you can enable both the Any Key and the Exclusive options simultaneously. This would result in a situation wherein the user could select one of the defined keys, but if he or she presses another key, program execution would continue anyway.

With multiple possible keyboard responses, you must allow the program to determine which key was pressed. To do so, use the Response() function in the Expression editor. One technique of using this function is to assign the response to a variable using the Variable icon, then check the value of the variable in a comparison statement and execute a section of code accordingly.

```
UserKey = Response()
IF UserKey == "A" OR UserKey == "a"
```

You can also use the Response() function directly in your evaluation. For example, you could use the If-Then icon to check the user's response, and if the evaluation is true, execute its partner.

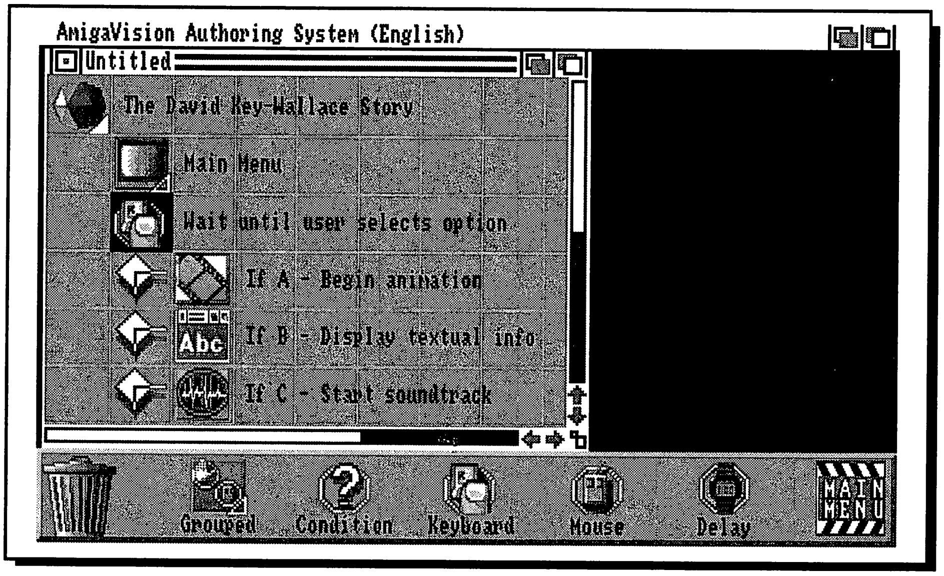
```
IF Response() == "A" OR Response() == "a"
```

Note that in the two statements above that incorporate the word IF, the IF represents the If-Then icon, and is not actually part of the expression. Figure 9-7 shows a section of code that evaluates possible answers and executes code based on the response.

An important consideration when using the Exclusive Key option is the effect it has on keyboard interrupts. If Exclusive Key is enabled, any previously defined and active keyboard interrupt is effectively disabled while the Wait Keyboard icon is in use. This includes mouse-interrupt hot spots as well.

By accessing the Object editor from the Wait Keyboard icon, you can display graphic elements, text information, or even variables on the screen while waiting for the keyboard input. The only two objects you cannot use for such an interim display are the Input field (used by the Data Form icon) and the Text window (used by the Text icon).

Related to the Object editor is the Auto Remove gadget. This gadget lets you determine how the graphic objects generated in the Wait Keyboard icon's Object editor are treated when the icon finishes executing. If Auto Remove is enabled (a check mark in the gadget box indicates that it is "on"), all the objects you had added in the Object editor from the Wait Keyboard icon are removed, and the



The David Key-Wallace Story
Main Menu
Wait until user selects option
If A - Begin animation
If B - Display textual info
If C - Start soundtrack

Figure 9-7
Evaluating a keyboard response

screen is restored to the state it was in before the graphics were placed on the screen. If Auto Remove is disabled (its gadget is not checked), the objects added via the Object editor remain in place until all the siblings of the Wait Keyboard icon are finished executing.

The final option on the Wait Keyboard icon is the Timeout command. Clicking on this button brings up a Specify Value requester that allows you to enter a maximum delay (in seconds) that you wish the Wait Keyboard icon to wait for the user to press a key. It allows fractions of a second (to two decimal places), and as with other Timeout commands, entering a value of zero will cause an infinite wait to occur. In this case, using a zero value for the Timeout command is often desirable, as you will want the user to indicate when to go on and not limit the time she or he has to decide.

Wait Mouse Icon

Because use of the mouse is so popular on the Amiga, the Wait Mouse icon will likely be the device you employ most often to control your program's interaction with the user. Like the other Wait icons, Wait Mouse causes program execution to pause until the user clicks the left mouse button. You can designate a number of specific targets for the click, or opt to accept any mouse click, regardless of the mouse pointer's position. It should be noted that the Wait Mouse command responds to clicking with the *left* mouse button. Although the AmigaVision manual incorrectly indicates that AmigaVision does not check for right mouse-button activity, if your application is started from the Workbench, all buttons are detected by the Wait Mouse icon. Clicking of the right mouse button can be detected using a function found in the Expression editor, which is discussed in detail in Chapter 13.

The Wait Mouse icon can be used as a Sibling icon, a child of another icon, or even a companion of a partner-requiring icon. It cannot be a Parent icon.

To use it, drag the Wait Mouse icon into the Flow window and position it in your program wherever you want program execution paused until you have some mouse input. Then double-click on the Wait Mouse icon to bring up the Wait Mouse requester. This requester is very similar to the Wait Keyboard requester (see Figure 9-8) but it does not have the Key(s) field.

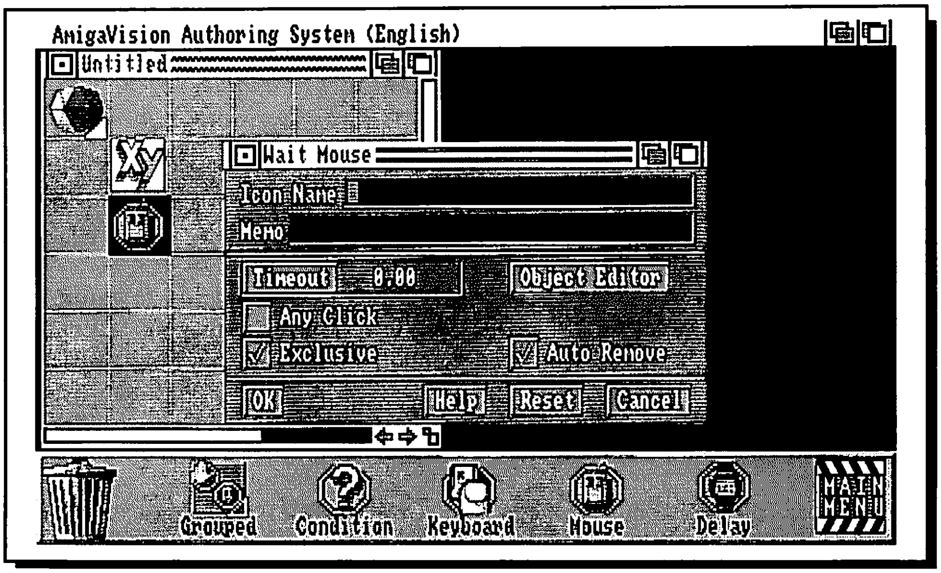


Figure 9-8
The Wait Mouse requester appears when the Wait Mouse icon is double-clicked

Five gadgets on this requester allow you to define how the Wait Mouse icon is to work in a given situation. Two of these, the Any Click and Exclusive options, are check-box gadgets that let you determine exactly what kind of mouse click will prompt the continuation of program execution. If the Any Click option is enabled (if a check mark appears on its gadget), a click of the left mouse button anywhere on the screen will cause program execution to continue. If the Any Click option is disabled, it will take a response generated by clicking on a specific hot spot (created within the Object editor) to continue program execution.

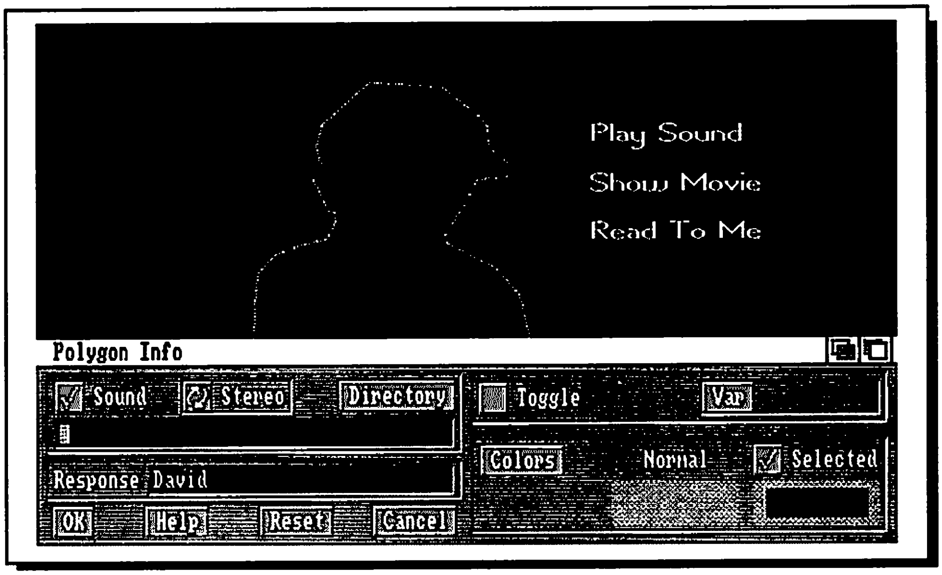
Just as the Wait Keyboard icon's Exclusive Key option affects keyboard interrupts, the Exclusive mouse-click option also disables any previously defined and active mouse interrupts while the Wait Mouse icon is active. If you want to keep your mouse interrupts active at all times, make sure you do not enable the Exclusive option on the Wait Mouse requester.

The Object Editor command button brings up the Object Editor screen, and it is here that you further define your interactive mouse displays. By using the available graphic and text objects (the Input field and Text window are not available here) you can create very impressive menus and displays that can return a number of responses to the main program when clicked.

Once you are in the Object editor, you can place graphic or text elements on screen, define their colors (you can even cause a color change to occur when the object is clicked in order to give the user a visual feedback), and specify a digitized sound to play when the object is clicked. More importantly, you can specify a text string to be used as a response, which is passed on to other commands in the program when the object is clicked. You can determine this response with the Expression editor's Response() function, and, using a variety of decision-making icons and commands, you can make the program react according to the user's response.

Figure 9-9 shows an interactive scene being defined from within the Object editor. Here several elements have been added to the display, including three text objects and one polygon object. Once you have created and positioned an object in the Object editor, you can double-click on the object to bring up a requester specific to it. (Each of the different graphic and text objects have a requester specific to the definition needs of that type of object.)

In Figure 9-9, you can see the Polygon Info requester, which was accessed by double-clicking on the polygon shape. In this requester, I defined which colors are to be used for the polygon object. Notice that in this case, the default colors — the colors that define the object in its usual state (when it has not been clicked) — are transparent, meaning they are invisible, while the selected colors — the colors that appear in response to a user's mouse click — are defined. The results of this definition is an invisible hot spot that, when clicked, momentarily takes on the selected color. The reason this hot spot has been defined to be normally invisible is it is overlaid on top of a screen image of a small child. When the user clicks on the



Response David

Figure 9-9
Defining hot spots in the Object editor

child the hot spot flashes and a response is returned to the program. In other situations you would want to have both the normal and the selected colors visible, especially in cases where the graphic object is being used as a button or menu item.

Notice that in the Response field in Figure 9-9 the text string "David" has been entered. This is the response that the Response() function would detect if the user clicked on this hot spot. Because the Response field can accommodate up to 255 characters, you can have a virtually unlimited number of responses. It is a good policy to make your response strings completely unambiguous so that you will be able to easily tell what they are for.

Another feature of the Object editor's hot spots is their ability to use digitized sound as a feedback response to confirm that a hot spot has been clicked. When defining an object via its requester, you can set audio feedback simply by selecting the Sound gadget. If you engage Sound mode but leave it undefined, the sound that plays in response to a mouse click will be the default feedback sound designated in the AmigaVision Defaults menu (see Chapter 4). You can, however, specify a different sound by clicking on the Directory gadget and selecting a file from your 8SVX directory. When you click on the name of the sound file in the requester to select it, you can choose to direct it to either both sound channels (for stereo sound) or the left or the right speaker only.

One thing that should be noted here (and it is equally true for the other Wait functions), is that it is often preferable to add the graphic hot spots on top of a previously loaded display. Using an Audio Visual command icon such as Screen or Video, you can display an image, then overlay it with graphics, text, or even invisible hot spots generated from the Object editor. In Figure 9-10, the figure of the child is a hot spot. Once you have used the Screen icon to define an image, you can specify that the image be displayed when you enter the Object editor from the Wait Mouse (or Wait Keyboard) icons. In the case that an image has been defined in a previous icon, AmigaVision will put up a requester asking if you want to have the image displayed in the Object editor. In Figure 9-9, I had indicated I did not want the screen image to be displayed in the Object editor, but later, when I was making some adjustments to the hot spots, I decided to load the image within the Object editor so I could position the graphic elements more precisely. With that in mind, I entered the Object editor and clicked on the OK button when AmigaVision asked if I wanted to load the background screen (which had been defined previously in the Screen icon). A scene demonstrating this from within the Object editor screen is shown in Figure 9-10.

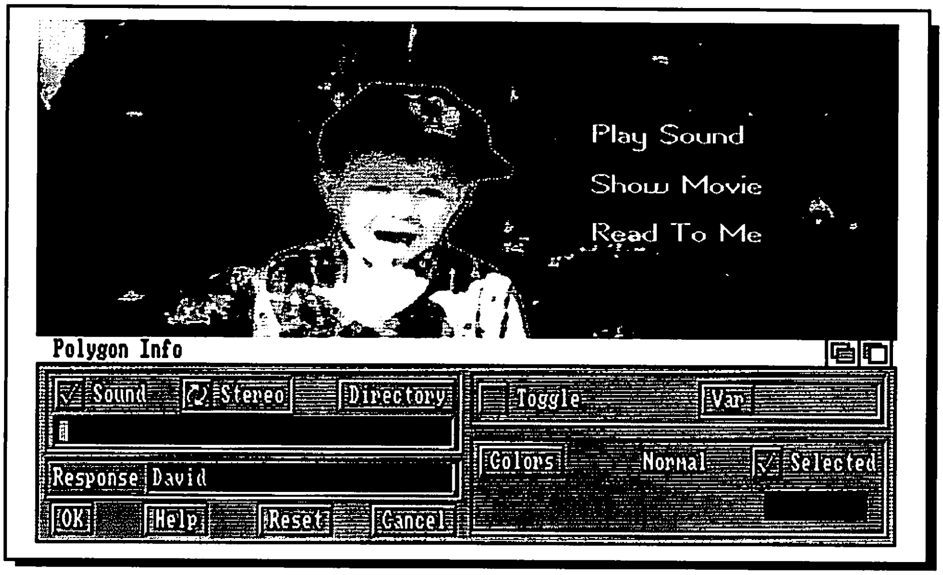


Figure 9-10
A composite image within the Object editor

The Wait Mouse requester has a check-box gadget labeled Auto Remove. When this gadget is enabled, any elements defined in the Object editor to be placed on screen by the Wait Mouse command will be automatically removed from the display when program execution moves passed the Wait Mouse icon. If this gadget is not enabled, those objects will remain on the screen until all the siblings of the Wait Mouse icon have been executed.

Like the other Wait icon requesters, the Wait Mouse requester contains a Timeout button. When clicked it brings up a Specify Value requester in which you can specify the amount of time (in seconds) the Wait Mouse icon should pause before it continues on. If you enter zero it will wait forever. If you enter a positive value, the program will pause for exactly that number of seconds and if, at that point, there has been no user input, program execution will continue. If you decide to add a time limit to the Wait Mouse icon, be sure to take into account the possibility that the user may not respond within the allotted time. For this reason you should plan some default action. Remember that the time length is measured in *seconds*, (not minutes, hours, jiffies, or other time units) and that AmigaVision is accurate to two decimal places.

Grouped Wait Icon

The Grouped Wait icon is used to combine several different types of Wait icons into one, more complex (but also more flexible) Wait icon. Generally, the Wait icons you will group are the Wait Keyboard and Wait Mouse icons, but you can also include the Wait Condition icon. You can use two or more Wait icons in a grouped wait structure.

The Grouped Wait icon can be a sibling, a child, or a partner of other icons. It always acts as a parent to other Wait icons (no other icon types can be its children), although it cannot be the parent of another Grouped Wait icon.

To use the Grouped Wait icon, drag it from the Wait menu into the Flow window and position it in your program. Then, select the other Wait commands you want to include in the group and position them as children of the Grouped Wait icon in the Flow window. Figure 9-11 shows a set of three Wait icons placed under one Grouped Wait icon.

Each child of the Grouped Wait icon can be defined individually. The Wait Keyboard icon can be programmed to wait for the press of any key or for the press of one or more specific keys. Meanwhile, the Wait Mouse can be set to wait for any mouse click or for a click on a specific hot spot, and the Wait Condition icon can be programmed to wait for some defined conditioned to become true. None of the Timeout lengths of the individual children of the Grouped Wait icon will have any effect, however. The Grouped Wait icon has its own Timeout value, which takes priority over all the others.

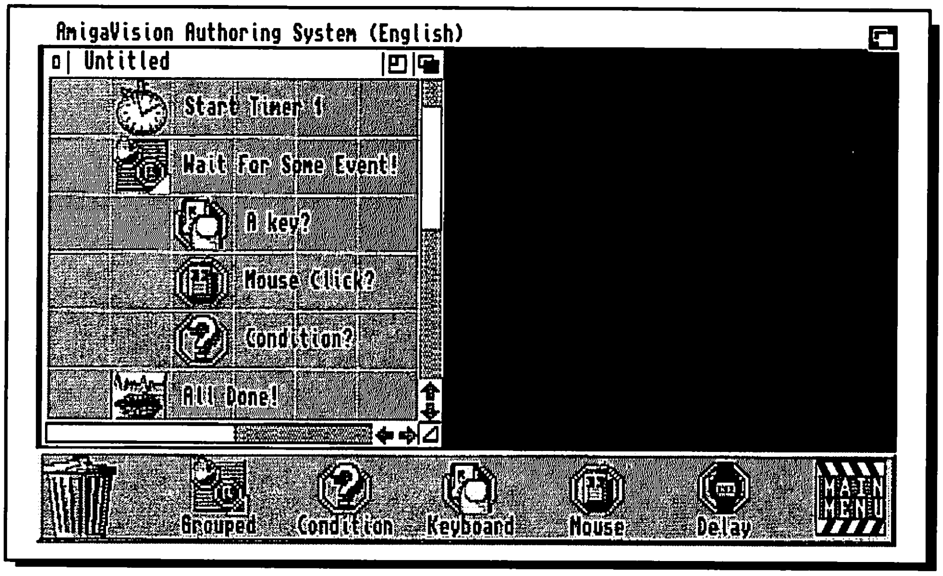


Figure 9-11
A Grouped Wait icon family at work

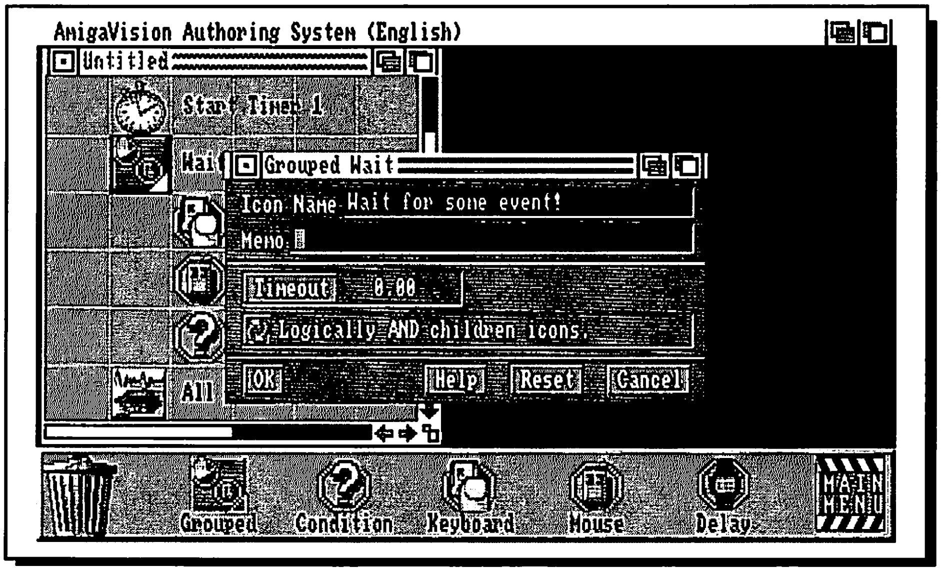
The Grouped Wait icon can be set to respond in several ways. Double-click on it and the Grouped Wait requester appears (see Figure 9-12). This requester has two major options. One is the Timeout command which, like all other Timeout controls, allows you to specify a maximum time (in seconds) that the Grouped Wait icon (and, in this case, its Child icons) will wait for the defined events to occur.

The other option is a multistate gadget that toggles the operation of all the commands in the group between the "Logically OR children icons" and the "Logically AND children icons" modes. These modes logically determine how AmigaVision responds to the child Wait icons.

If the mode is Logically OR, AmigaVision pauses until at least one of the Wait icons has been satisfied. If just one is satisfied, program execution continues. In other words, the logic works like this:

IF Icon1 is true OR Icon2 is true OR Icon3 is true THEN continue

In order for one of the icons to evaluate as true, a condition defined in one of them must be met — the Wait Keyboard icon must detect a key press, the Wait Mouse icon must detect a mouse click, or the Wait Condition icon's expression must evaluate as true.



Icon Name Wait for some event!

Memo:

Timeout 0:00

Logically AND children icons.

Figure 9-12
The Grouped Wait requester appears when the Grouped Wait icon is double-clicked

If the mode is Logically AND, all the conditions of all the children of the Grouped Wait icon must be met. In this case the logic works this way:

IF Icon1 is true AND Icon2 is true AND Icon3 is true THEN continue

All the conditions must be met before execution continues. They do not have to be satisfied in any particular order, but all must be satisfied before the program can move on.

You can employ the Grouped Wait icon when you want to allow the user of your program the option of responding via either the mouse or the keyboard. To do so, just use both Wait icons as children of the Grouped Wait icon and set the mode to Logically OR.

– 10 –

The Audio Visual Command Menu

Contained within the AV, or Audio Visual, menu are the commands that allow AmigaVision to display images, animations and video, and to play sounds, speech, and music — that is, to incorporate all the things that the word multimedia brings to mind. Of course, these are also things that the word Amiga brings to mind, because in these areas the Amiga shines.

The Audio Visual Command Icons

Click on the AV icon on the Main menu and AmigaVision displays the Audio Visual menu of icons (see Figure 10-1). These nine commands will quickly become the most important tools of any AmigaVision programs you create. If the other commands of AmigaVision are thought of as the skeleton of your program, it is the AV icons that are its heart and soul.

Several of the AV commands require files created by programs outside of AmigaVision. For example, the Screen icon is normally used to display pictures created with one of the many Amiga graphics programs. The Animation icon uses animation files in the ANIM5 format, which are also created with programs other than AmigaVision. The Sound command plays digitized sound samples, which must originate from a sound sampler. Each of the AV commands has its own specific data requirements (we will discuss these requirements separately in relation to each particular command). The point is, that as an author, you will have to be able to either create or acquire the pictures, sound effects, music, and animations needed for these commands to be effective.

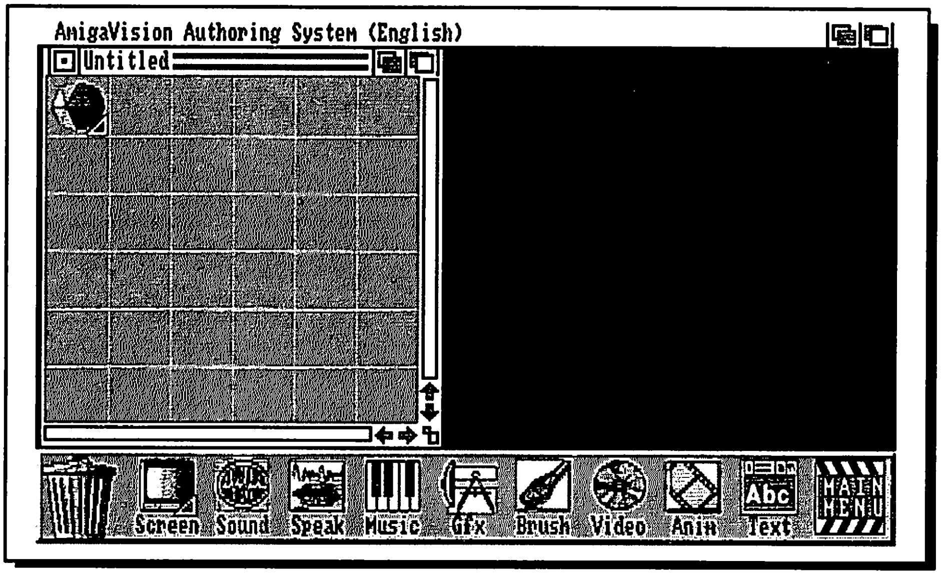


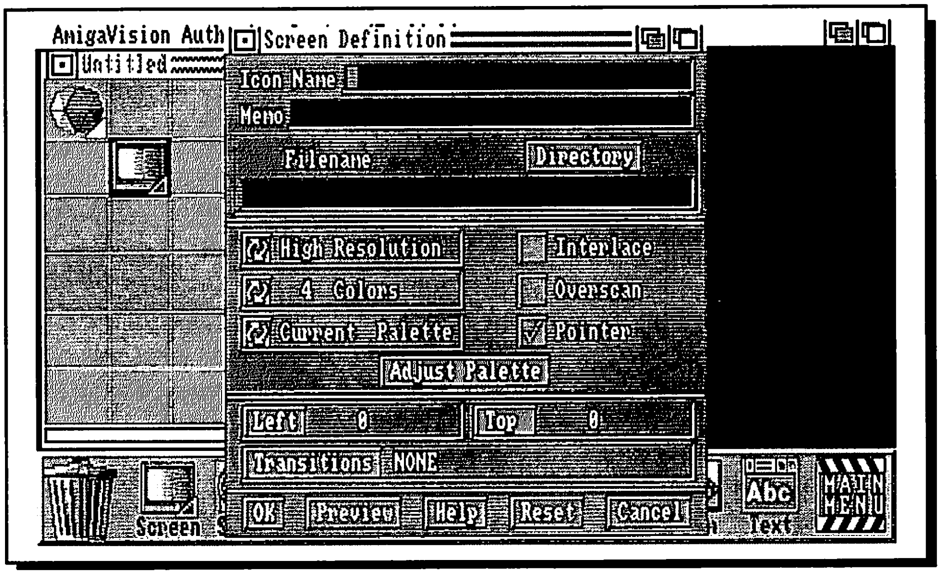
Figure 10-1
The Audio Visual command icons are the most important tools of program creation

The Screen Icon

We begin our study of the Audio Visual icons with the Screen icon. This command has several important functions, the most important being its ability to display Amiga IFF graphics images of any resolution and size, including such special modes as Extra_Halfbrite (64-color mode), Hold and Modify (HAM, or 4096-color mode), and overscanned displays. Once a picture has been loaded, the rest of the AmigaVision commands work in and on it — until another graphics icon changes the display.

The Screen icon can be used as a Sibling icon, a child of a Parent icon, the companion to a partner-requiring icon, or as a parent to other Audio Visual icons, including other Screen icons.

To use the Screen icon, drag it from the AV menu and position it in your program's Flow window. Then double-click on the Screen icon to bring up the Screen Definition requester (see Figure 10-2).



Icon Name:
Memo:
High Resolution
4 Colors
Current Palette
Interlace
Overscan
✓ Pointer
Left 0
Top 0
Transitions NONE

Figure 10-2
The Screen Definition requester appears when the Screen icon is double-clicked

As you can see from Figure 10-2, the Screen Definition requester has many options. Besides the usual Icon Name and Memo fields and the OK, Help, Reset, and Cancel buttons, there are another dozen options that you can use to modify the screen and the appearance of your AmigaVision programs.

Let's start with the three multistate gadgets on the left side of the requester. The top gadget is the Display Mode gadget. If you click on it, you can step through all the modes the Amiga supports. The selections are: Low Resolution (320 pixels wide), High Resolution (640 pixels wide), Extra_Halfbrite (320 pixels wide and 64 colors, 32 of which are half-intensity shades of the first 32), and Hold & Modify (HAM mode, a 320-pixel-wide, 4096-color display). From this gadget you can also select Current (which indicates that AmigaVision should use whatever mode and colors are currently on display) and File Defined. This last option is handy when you are using a variable that contains the name of the picture you want displayed. Because AmigaVision has no way of telling ahead of time what that variable will contain, it cannot know what the file characteristics of the image will be. By selecting File Defined you are instructing AmigaVision to set the display mode according to the characteristics of the picture contained in the variable.

The middle multistate gadget allows you to change the number of colors on the basis of the display mode. For example, if you have selected Low Resolution, it will allow you to choose from 2, 4, 8, 16, or 32 colors. High Resolution mode allows only 2, 4, 8, or 16 colors at a time. Extra_Halfbrite and HAM are special cases, each allowing only 64 and 4096, respectively. If you are using Low or High Resolution modes, the Color gadget will cycle through the allowable number of colors for that mode. In Extra_Halfbrite and HAM modes this gadget is ghosted, or unavailable, although it does display the number of colors that the mode supports.

The bottom multistate gadget allows you to choose how the color palette is to be handled. The three options are Original, Current, and Modified. Original indicates that the selected picture should be displayed using the palette contained within it. If you select Current, AmigaVision will display the picture using whatever colors are on screen currently, ignoring the palette information in the file itself. Choosing Modified means you want to use the palette contained within the picture, but to modify it based on any adjustments made with the Adjust Palette command.

To the right of the multistate gadgets are three check-box gadgets you can use to further define how the screen will be displayed. The first toggles Interlaced (400-line) mode, and can be enabled in any of the Amiga's display modes. The Interlaced option doubles vertical resolution and, unless you are using an Amiga 3000 or your system is equipped with a flickerFixer (Microway) or other display-enhancing device, it will cause some degree of flickering on your monitor.

The second check-box gadget is used to turn Overscan mode on and off. Overscanning allows you to extend the display of your computer images to fill the screen's border. This mode is often used in video work to eliminate ugly screen borders.

The third check-box gadget lets you determine whether the mouse pointer is to be displayed. If your application is an interactive program that requires the user to make selections using the mouse, you will certainly want the pointer to be visible. If your program is controlled by the keyboard, however, or is merely showing some screens, the mouse pointer might distract the user. For situations such as these, you can elect to turn it off.

Beneath the array of multistate and check-box gadgets is a command button labeled Adjust Palette. Clicking this button brings up a Palette requester that can be used to define the colors in your screens or to modify the colors in an existing display. Figure 10-3 shows how the Screen Palette requester works. (If you have already defined an image earlier in your program, AmigaVision loads that image and displays the requester on top of it.) You can alter any color by clicking on it in the palette box (on the right) and using the sliding gadgets (on the left) to adjust its Red, Green, and Blue values. Depending on your display mode and the number of colors you have selected for that mode, the palette box will contain 2 to 32 colors. (While HAM can display 4096 colors, you can adjust only the first 16 colors. Likewise, Halfbrite mode displays 64 colors, but the palette shows only the first 32.) Once you have defined a modified palette this way, AmigaVision will use this new palette instead of the default in displaying any image for which you have selected Modified Palette from the Screen Definition requester.

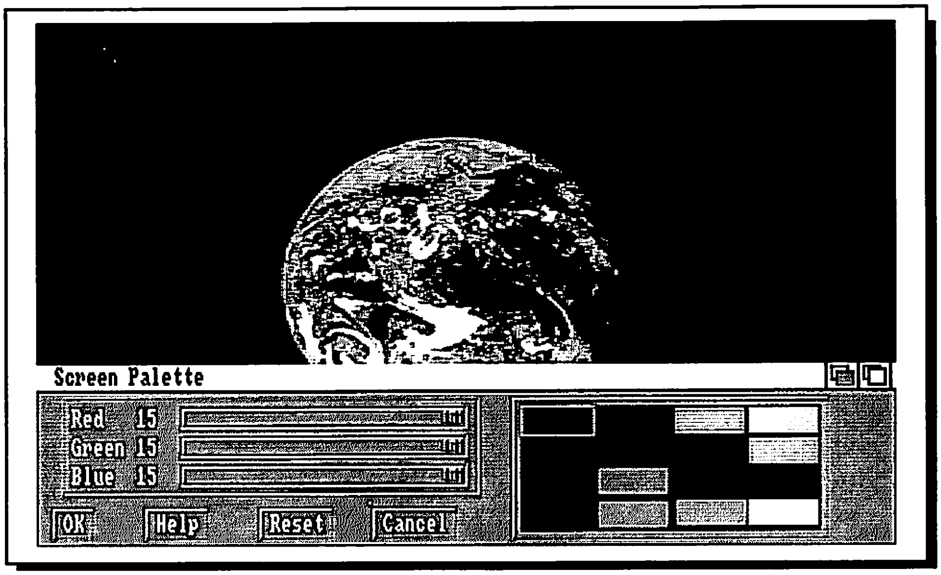
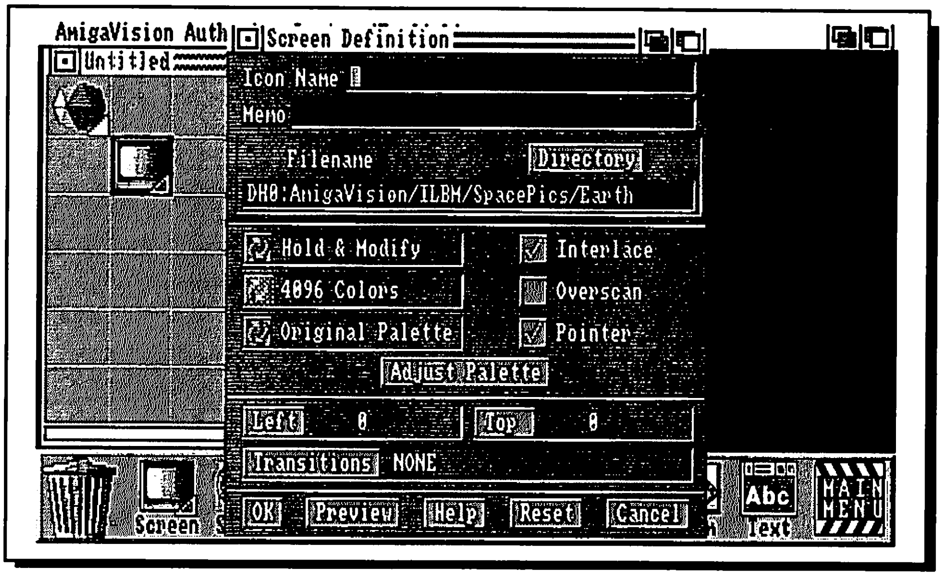


Figure 10-3
The Screen Palette requester is used to define the colors in a screen

If you want all these changes to apply to a defined image, click on the Directory button to bring up the AmigaVision file requester. From here you can select any IFF-compatible images, including pictures, brushes, or ANIM5-format animations (if you select an animation, only its first frame will be displayed). Once you have made your selection from the file requester, click on the OK button to accept it (or just double-click on the file name in the list to achieve the same effect). The file requester will disappear, returning you to the Screen Definition requester which now reflects the characteristics of the screen selected. The information corresponding to these characteristics is found in the screen file itself and is read by AmigaVision when you select the file name.

Take a look at Figure 10-4. Here I have selected a picture named 'Earth' from the ILBM image directory. The Screen Definition requester contains all the information needed to display this picture correctly. For example, look at the three multistate gadgets on the requester.



Icon Name
Memo
Filename
DH0:AmigaVision/ILBM/SpacePics/Earth
Hold & Modify
4096 Colors
Original Palette
Interlace ✓
Overscan
Pointer ✓
Left 0
Top 0
Transitions NONE

Figure 10-4
A defined Screen Definition requester containing display information

The top multistate gadget reflects the graphics mode of the picture, which in this case is Hold & Modify (HAM), a 4096-color display mode. The middle multistate gadget indicates the number of colors available in this mode and shows that this is indeed a 4096-color display. (Note that the button itself is ghosted, which indicates it is not possible to change the number of colors. While some display modes allow a variable number of colors, HAM by definition is a 4096-color display, so there is no option to increase or decrease the number of colors.) The bottom multistate gadget indicates how the palette is to be treated. Here I have chosen the default selection, which is the original palette contained within the picture itself.

To the right of the multistate gadgets are the three check-box buttons. When you select a picture, the top two will reflect the information found in that picture's file. The first gadget, Interlace, is checked, showing that the Earth file is to be displayed in interlaced (400-line) mode. The second gadget bears no check mark, which means that the file is to be shown without using Overscan mode. Nothing in this picture file tells AmigaVision how the pointer is to be treated, so the Pointer button is unaltered.

Below these gadgets are two buttons labeled Left and Top, each with a value displayed to its right. These commands are used to determine an offset position for displaying the screen.

You can enter both positive and negative values for Left (which corresponds to the X axis) and Top (which corresponds to the Y axis). If you use a positive Top offset, the image will be moved down the screen the specified amount. A positive Left offset shifts the image to the right. In the case of a 640-by-400 picture, using a Left offset value of 320 and a Top value of 200 would mean that when the image was displayed, the pixel defining the upper-left corner of the image would appear exactly in the middle of the screen. As a result, only the upper-left quarter of the image would show, and everything to the right of and below the intersection of pixels 320 and 200 would be off the screen.

If you specified the same values as negatives (-320 and -200) for Left and Top, the image would be shifted left and up in the display, and would show only the lower-right quarter of the image in the upper-left corner of the monitor.

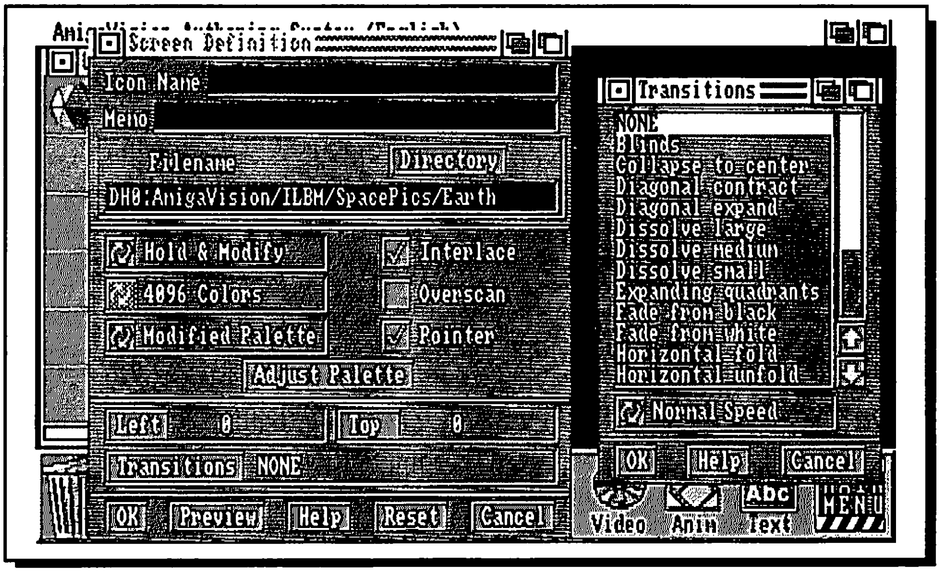
To set a screen-offset position, click on either the Left or Top buttons to summon a Specify Value requester. Enter the value you want, and then click OK. On exiting the Specify Value requester, you will see the new value displayed to the right of the button you selected. When you then display the corresponding image, it will be shifted accordingly.

In the Specify Value requester you can indicate variables as well as constants. By using a variable for positioning the image, you are able to perform some interesting effects not directly supported by AmigaVision. For example, if the picture you want to display is a virtual image — an image larger than can be displayed on a single screen — you can use the Left and Top commands to scroll around and see the whole thing.

In most cases, you will want to position full-screen images at the default locations of 0,0, and will ignore the screen-positioning features. However, in the case of partial-screen images such as brushes, these positioning features become very important. Surprisingly, they can also be used to move animations, often with very useful results.

The next feature of the Screen icon is the Transitions option, which offers 18 ways of changing from one screen image to another. (Actually the option offers 19 transition effects if you count the None selection, which causes an abrupt change of displays as in an old-fashioned slide show. In video terms, however, such a change is referred to as a cut and not a transition.)

When you click on the Transitions button, the Transitions requester (see Figure 10-5) appears. From here you can select the transition you want to use. The window that displays the list of transitions offers the usual scrolling features so that you can select items that do not initially show in the window. There is also a multistate gadget that has three possible states, Slow Speed, Normal Speed, and Fast Speed. These speed settings allow you to determine the rate at which the transition occurs. The rate you select will depend upon the transition you use as well as your personal taste.


Screen Definition

Icon Name

Memo

Filename: DH0:AmigaVision/ILBM/SpacePics/Earth

Hold & Modify ✓ Interlace

4096 Colors Overscan

Modified Palette ✓ Pointer

Left 0 Top 0

Transitions NONE

Transitions

NONE
Blinds
Collapse to center
Diagonal contract
Diagonal expand
Dissolve large
Dissolve medium
Dissolve small
Expanding quadrants
Fade from black
Fade from white
Horizontal fold
Horizontal unfold

Normal Speed

Figure 10-5
The Transitions requester appears when the Transactions button is clicked

As you are creating your programs, it is important to keep in mind that in most cases, a transition can occur only between screens of the same resolution and mode. The exceptions to this rule are the Fade From Black and Fade From White transitions. Also, if you select None as a transition you can move instantly from one screen to another, regardless of the resolution or palette differences. If you try to make an illegal transition (from one resolution to another), AmigaVision will display an Error requester (during the presentation of the program), informing you of your mistake.

It is possible to make a transition from a HAM screen to another screen as long as the horizontal and vertical resolutions are the same. The trick is to display the non-HAM screen in HAM mode by using the multistate gadget on the Screen Definition requester to change the display mode of the non-HAM image. It will then work, although the second screen will retain the color palette of the first screen for a moment (this generally happens anyway between two HAM images). You can minimize this effect by doing some preprocessing of your images using a program like PIXmate (Progressive Peripherals & Software), Butcher (Eagle Tree), or the Colors module of Deluxe PhotoLab (Electronic Arts). With any of these programs, you can try to incorporate the same base palette of 16 colors in all of the HAM images you use. If all your HAM images do use the same base palette, there will be no color-change problem in the AmigaVision transitions.

Your AmigaVision manual states that it is a good practice to use the same mode and resolution for all the screens in your application. If you can live with that, fine; but don't restrict your programs just because of a minor limitation in Amiga-Visions's transitions. Go ahead and experiment. Sometimes you can find solutions that even the AmigaVision programmers did not know were possible.

At the bottom of the Screen Definition requester is an extra command button — the Preview button. It does just what its name suggests, it allows you to preview the screen and the selected transition.

When you select Preview, AmigaVision looks back through your flow chart and finds the preceding Screen icon (if there is one). It then loads this image (or sets the display mode if there is no prior image) and begins the transition to the new screen.

Preview mode makes it very easy to tell if the effect you created is what you had intended and if the speed is right. If the results are not satisfactory, simply change the speed or choose another transition and preview it again. You can try as many combinations as you like. The beauty of the Preview mode is you do not have to work your way through the whole AmigaVision application to see a specific transition. Instead, you can check each one as you program.

You can use the Screen icon merely to clear the screen by placing it in your program and leaving it undefined. You can also use it to set the display mode for other parts of your AmigaVision application. In fact, it is a very good practice to place a Screen icon very early in your AmigaVision programs, setting the resolution, number of colors, and the interlace flags to what you want to work in. Later, when you add objects via the Object editor from other icons, AmigaVision will automatically use these settings.

The Brush Icon

While similar to the Screen icon, the Brush icon behaves somewhat differently and is most often used in conjunction with an existing picture loaded by the Screen icon. It displays images that are not full-screen size.

The Brush icon can be a Sibling icon, a companion to a Partner icon, or the child of a Parent icon. It cannot be a parent itself.

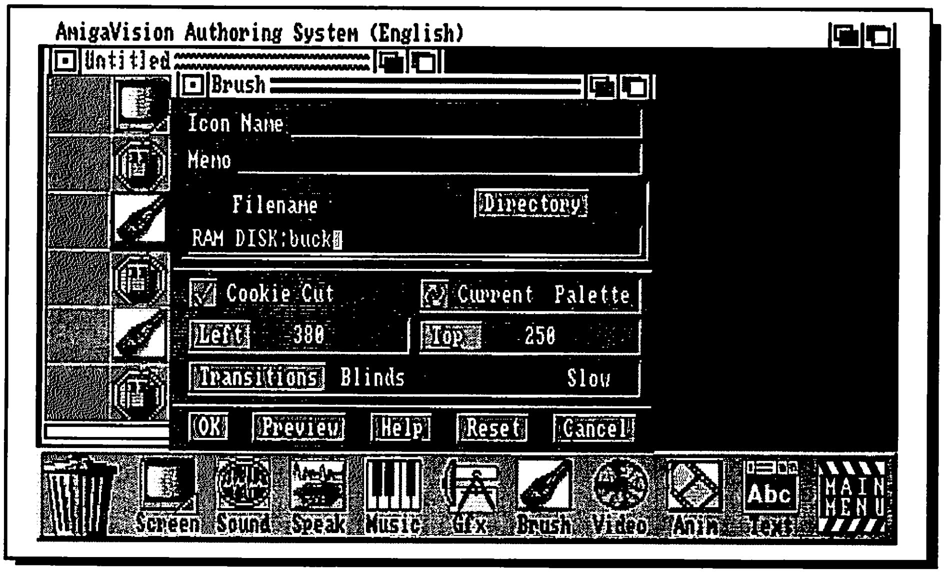
The word "brush" was first popularized as a computer-graphics term by Electronic Arts' DeluxePaint. That program applies the term to the small pieces of bitmap it allows you to cut from an image and paste down or use as a paint brush. Brushes — that is, images saved in brush format — are now generated by nearly every Amiga paint program, and disks of clip art in brush format are available from many companies.

Unlike the Screen icon, which replaces an existing screen image with a new one, the Brush icon can overlay a smaller piece of bitmap onto an existing screen. It has no provisions for changing the mode or resolution of the screen; instead, it works in whatever mode it finds itself in. For that reason, it is generally a good idea to use brushes in the same display modes with which they were generated. Otherwise, you can end up with brush images that appear elongated or squashed, or that do not display their color information correctly.

To use the Brush icon, drag it into the Flow window and place it into your program. Then double-click on it to open the Brush requester (see Figure 10-6). As you can see, several of this requester's features are very similar to those of the Screen requester. The Brush requester does, however, offer some additional options.

Selecting a graphics brush is very similar to selecting a picture with the Screen icon. Just click on the Directory gadget and choose the brush-image file from the requester that appears, changing the device and directory if needed.

The Left and Top gadgets work here just as they do in the Screen icon. By clicking on them, you summon the Specify Value requester, wherein you can enter a specific screen location or choose a variable. Although these positional commands are of limited use in relation to the Screen icon, they are of great importance to the Brush icon. Rarely will you want to display a brush at the top left of the screen, which is where the default values of 0,0 place it. Generally, you will need to position these brushes according to custom specifications, and unless you know the exact coordinates of where they are to go, you will need to do some experimenting to get them right. (Lucky for us, the Preview button makes it easy to test out different positions.)


AmigaVision Authoring System (English)

Icon Name

Memo

Filename

RAM DISK:buck

[✓] Cookie Cut

[✓] Current Palette

Left 380

Top 250

Transitions Blinds Slow

Figure 10-6
The Brush requester appears when the Brush icon is double-clicked

The Transition option also works in relation to the Brush icon exactly as does its counterpart in the Screen icon. Clicking on it takes you to the Transition requester, which gives you the same list of options. While the effects offered here work just as they do with full screens, they are limited to the area of the screen that the brush overlaps. The only exception to this is the two fades (Fade From Black and Fade From White), which make the whole screen fade to black or white, then fade back in with the brush displayed. As with the other transitions, these effects do not erase the old screen when used in Brush mode. Because the transitions are confined to the area of the brush, some effects that seem too slow for a full screen work quite nicely for smaller brushes. By the same token, some transitions that work fine for full screens appear far too fast — even set to the slowest transition speed — when used with a small brush. To check the effect of a transition, use the Preview button at the bottom of the requester, which works just as it does on the Screen Definition requester.

The single multistate gadget on the Brush requester lets you determine where AmigaVision gets its palette information. Choosing Current Palette means that you want the brush to use the same palette as the screen upon which it is being deposited. Override Palette instructs AmigaVision to use the palette information contained in the brush. This has the effect of changing the palette used by the entire display — not just the area of the screen the brush occupies. If all your screens and brushes use the same palette, this will not present a problem.

The requester's single check-box gadget, labeled Cookie Cut, indicates to AmigaVision that you want the brush placed on the screen. Without the Cookie Cut option enabled, AmigaVision will stamp the brush down on the screen using a rectangle that is as large as necessary to encase the whole brush. The entire rectangle will be stamped down, thus overwriting anything under it.

If Cookie Cut is enabled (a check mark appears next to it), any area of the brush rectangle that is background color will be treated as invisible. When placed on the screen in this mode, only the actual image will replace what was already on the screen, and wherever there was background color in the brush, the original screen image will show through.

Keep in mind that when you place these brushes onto the screen, you are making them part of the existing screen. This differs greatly from placing brushes on the screen from within the Object editor: these are removed from the screen (at your option) when AmigaVision finishes executing their corresponding icon. Brush-icon images do not go away until they themselves are overwritten with another brush or until you load another screen.

The Graphics Icon

The Graphics icon (labeled GFx on the Audio Visual icon menu) is something of a hybrid, bridging the gap between the other graphics-oriented icons (Screen and Brush) and the Object editor. Through its Object editor access, you can place a variety of text and graphics on the screen, which can either be removed when the icon and its siblings are finished, or be left permanently stamped into the current screen display (as with the Brush icon). In addition, the Graphics icon offers a feature that allows a very simple means of generating animation: a technique called color cycling.

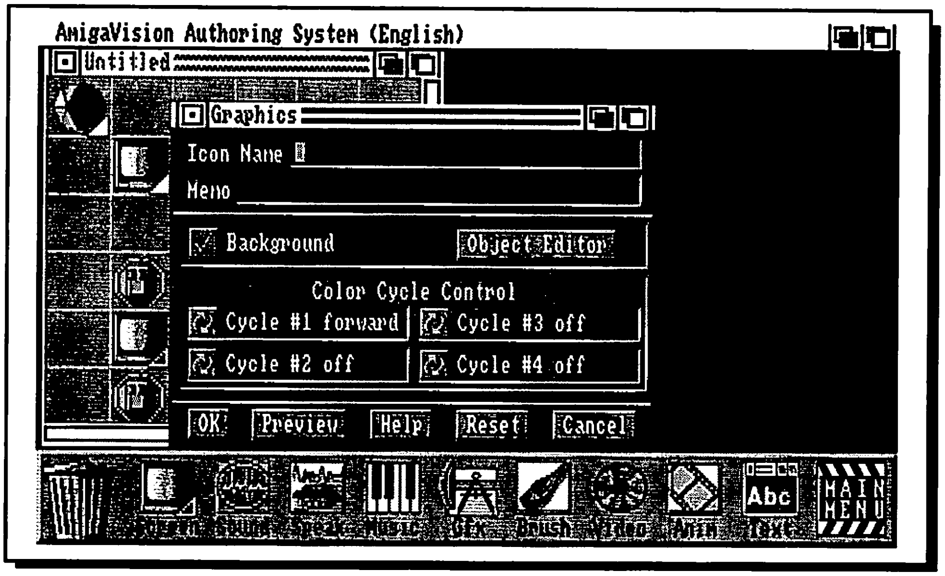
The Graphics icon can be used as a sibling to other icons, as a child of a Parent icon (it is especially useful as a child of a Screen icon), and as a companion to a Partner icon. It cannot be a Parent icon itself.

Drag the Graphics icon into the Flow window and position it into your program. Because of the way the Graphics icon works, it is especially important to think carefully about exactly where the icon is placed.

Normally, the effects of the Graphics icon (such as the placement of objects on the display from within the Object editor), remain only until the last sibling of the

Graphics icon is finished executing. However, once color cycling has been enabled, it will continue even after the Graphics icon and its siblings have finished, until a new screen has been loaded or enabled.

Double-click on the Graphics icon to open the Graphics requester (see Figure 10-7). This requester contains six major options as well as the usual complement of buttons and fields found in other AmigaVision requesters.



Icon Name
Memo
✓ Background
Object Editor
Color Cycle Control
Cycle #1 forward
Cycle #3 off
Cycle #2 off
Cycle #4 off

Figure 10-7
The Graphics Icon requester appears when the Graphics icon is double-clicked

Clicking on the Object Editor button brings up the usual Object editor, complete with its selection of graphic objects (rectangles, lines, circles, ellipses, polygons, brushes, and text). You can place these on screen and specify that they change colors and return a response when the user clicks on them.

Graphics icon objects are normally removed when the icon and its siblings finish executing. If you enable the Graphics requester's Background gadget, however, these objects will remain on screen after the Graphics icon finishes executing; until a new screen is displayed.

Regardless of how the Background gadget is set, color cycling remains until either a new screen is displayed or until another Graphics icon explicitly shuts it off. Color-cycling control is handled by the four multistate gadgets on the Graphics requester. Each gadget controls a different set of colors, and can switch between four modes: Cycle Forward, Cycle Reverse, Cycle Off, and No Change (N/C). The No Change option is useful when you want to alter some of the cycling ranges but want the remainder unaffected.

AmigaVision itself cannot create color-cycling effects or control their speed; these things must be done within the paint program you use to create the graphics. AmigaVision supports only four sets of color ranges for cycling. Keep this in mind when using such programs as DeluxePaint (Electronic Arts) that can save up to six ranges. Regardless of how many color-cycling ranges your paint program supports, AmigaVision will work with only the first four. If you use a range higher than that, you will not be able to use it from within AmigaVision. The speed at which the colors are cycled is also set from within the paint program and not from within AmigaVision, so you should select the speed you want before saving the picture.

The Animation Icon

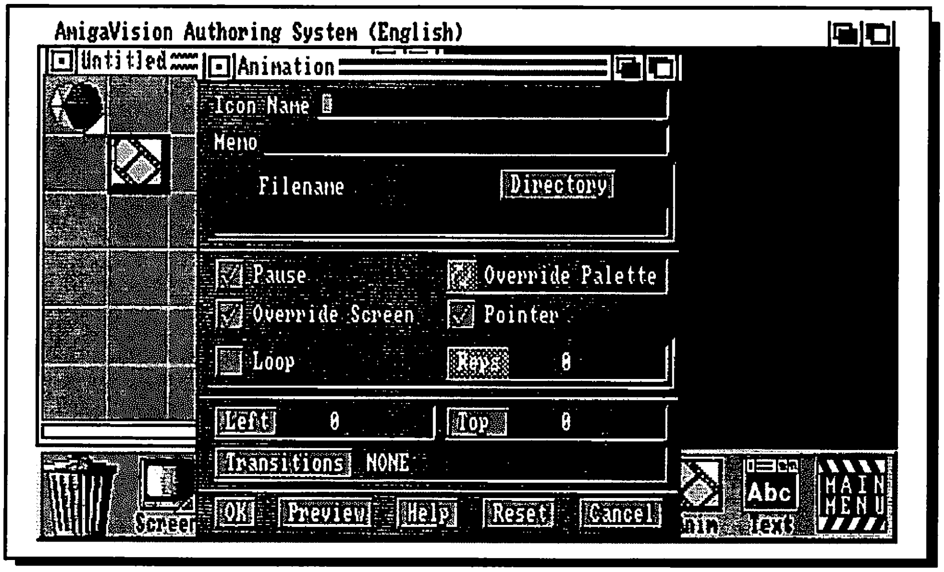
The Animation icon is used to play standard Amiga animations that have been stored in the IFF-ANIM5 format. This includes animations made in any Amiga display mode, although it does not, unfortunately, include animated brushes such as those created by Electronic Arts' DeluxePaint III. Animations, like pictures displayed via the Screen icon, overwrite the image currently on screen unless you explicitly instruct them not to.

The Animation icon can be used as a sibling to other icons, as a child of a Parent icon, or as the companion of a Partner icon. It cannot be used as a Parent icon itself, however.

Place the Animation icon into the Flow window by dragging it from the Audio Visual menu and releasing it at the desired position in your program. Then double-click on it to bring up the Animation requester (see Figure 10-8).

The Animation requester has some options common to other graphics-image requesters. For example, to select an animation file, you can type the full path and name of the animation into the Filename text-input field or, easier yet, click on the Directory button and use the standard AmigaVision file requester to find and select the animation file you want to use.

The Animation requester also has the Left and Top screen-image positioning buttons found on both the Screen and Brush icon requesters. These allow you to precisely position the animation on the screen, shifting it right, left, up, or down



Icon Name
Memo
Filename
✓ Pause
✓ Override Palette
✓ Override Screen
✓ Pointer
☐ Loop
Reps 0
Left 0
Top 0
Transitions NONE

Figure 10-8
The Animation requester appears when the Animation icon is double-clicked

by some defined offset. Although the current version of AmigaVision does not support animated brushes (animations smaller than full-screen size), you can use this option to shift the animation and achieve a similar effect — showing only part of an animation on the screen. And, contrary to what the *AmigaVision User's Manual* states, the Left and Top commands *do* accept negative values, just as they do in the Screen icon.

Also found on the Animation requester is the Transitions option. You can initiate animations using any of the standard transitions available for screens or brushes. The same restrictions on resolutions and palettes that apply to screens also apply here.

The Animation icon offers a wonderful option found only on certain commands in the Audio Visual menu (the Animation, Sound, Speech, Music, and Video commands) and the System menu's Resource Control command. This powerhouse command is allowed via the Pause check-box gadget.

Pause, when enabled (a check mark appears next to the gadget), forces Amiga-Vision to wait until the current icon is finished executing before moving on to the next. There's nothing too unusual about that; this is how most AmigaVision commands work. If the Pause option is not enabled, however, something entirely different happens: the corresponding icon is started (in this case, the animation begins), *but program execution continues onward!*

This feature gives you the ability to execute other AmigaVision code while the animation is playing. For example, while the user watches the animation, you could present a text window to describe what is happening in the sequence, allow the database to perform a search, display video from a laser disc (via a genlock), or perhaps narrate the animation with a digitized voice.

When a programmed interrupt occurs, its effect on an animation that is playing depends on how the Pause option is set. If Pause is enabled, the animation stops while the interrupt code is executed. When the interrupt routine finishes executing its own code, control returns to the Animation icon and the animation starts playing again — from the beginning. It does not continue where it left off when the interrupt occurred. You should take this into consideration if you are combining animation with an active interrupt routine.

What your AmigaVision manual does not tell you is what happens if you start an animation without the Pause option enabled. With the Pause option off, an animation broken into by a programmed interrupt routine continues to play — even while the interrupt code is being executed!

This is not to say that the things included in the interrupt routine won't affect the animation — they will. If your interrupt routine uses another Animation, Screen, or Brush icon, for example, this will stop the animation from running. Then, unless you restart the original animation from the interrupt code (or from elsewhere in your program), it will not continue to play when the interrupt finishes.

Other commands can affect the animation screen. For example, hot spots added from the Object editor can leave marks on the screen, even when the Auto Remove option is enabled. To get around this problem, place Object-editor objects only in areas of the animation screen where no movement is occurring. As an alternative solution, draw your buttons directly onto the animation (in all frames), and then place invisible hot spots over them. Because the buttons and gadgets are part of the animation and the hot spots are invisible, the hot spots do not cause any damage to the image.

The multistate gadget on the Animation requester lets you determine where the color-palette information comes from. If this gadget is set to Override (the default setting), the palette comes from the animation. If it is set to Current, the color palette of the current screen is used instead.

Like the Palette gadget, the Override Screen check-box button lets you elect to use the screen resolution of the animation (the enabled setting is the default). If you disable this option (by clicking on the gadget), AmigaVision will use the current screen resolution to display the animation. If you do not use the default, you can use the Left and Top gadgets to set an animation on top of an existing screen picture and still let some of the old screen show through. (This works best if the screen and the animation use similar palettes and display modes.)

The Loop gadget triggers Looping Mode. If it is not enabled, the animation will play through just once, ending on the last frame. If it is enabled, you will be able to use its companion gadget (labeled Reps) to set the number of times to repeat the animation. If Reps is set to 0 and Loop is enabled, the animation will play over and over, indefinitely.

According to the AmigaVision manual there is no way to determine what frame an animation is on, so you cannot sync another action to a specified frame. In actuality, the Anim() function, which can tell you the current frame number, is one of a handful of functions added to the Expression editor in AmigaVision version 1.53G. The manual, however, does not reflect these additions.

The Anim() function returns the current frame of a playing animation to a variable. Let's assume that you have already created a variable called FrameNumber from within a Variable, Module, or Subroutine icon. To find the current frame, you would add the following expression to your program using the Variable icon:

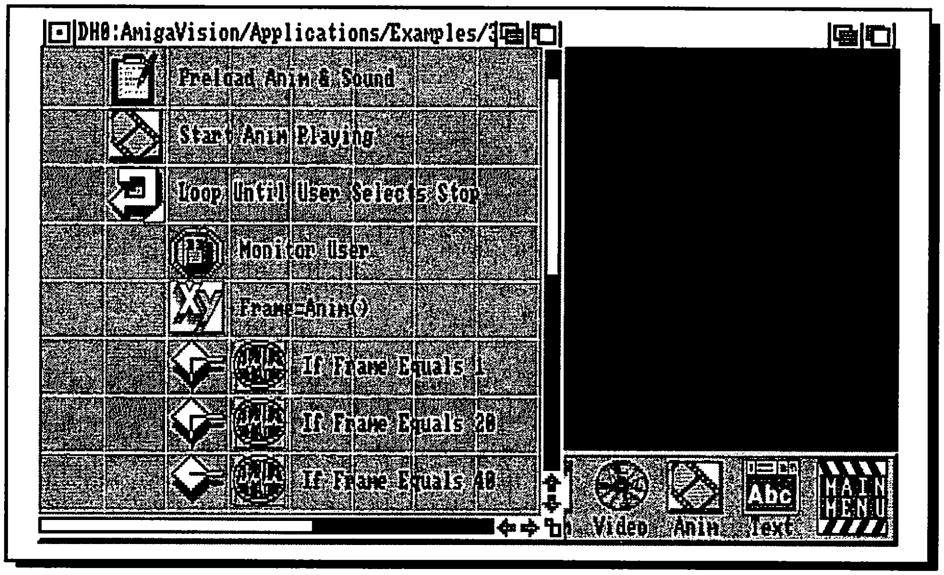
```
FrameNumber = Anim()
```

Once this expression is executed, the variable FrameNumber will contain the number corresponding to the animation frame that was being executed when you called the Anim() function. This number can then be used by a conditional icon (such as If-Then, If-Then-Else, or Conditional Goto) in order to execute some other commands at specific points in an animation. For an example of such a routine, see Figure 10-9.

If you want to play a certain sound every time an animation reaches a particular frame, just start the animation playing with the Pause option disabled, and then monitor the variable until it reaches the frame number (or perhaps a range of numbers) of interest. When it does, execute your sound code.

Another technique for syncing animation with another event involves using the Conditional Wait icon. Just specify an expression wherein the program waits for the condition to become true. For example, suppose you had entered this expression into the Conditional Wait icon's expression field:

```
(Anim() >= 20 AND Anim() <= 22)
```


DH0:AmigaVision/Applications/Examples/3

```
Preload Anim & Sound
Start Anim Playing
Loop Until User Selects Stop
    Monitor User
    Frame=Anim()
        If Frame Equals 1
        If Frame Equals 20
        If Frame Equals 40
```

Figure 10-9
Syncing animation frames with other events

In this scenario, when the Conditional Wait icon executes, program flow will pause until the expression evaluates as true — that is, when the animation plays frames 20, 21, or 22.

The Video Icon

The Video icon command allows you to incorporate full-motion prerecorded video from a laser disc into your AmigaVision programs. You can combine these video images with text, brushes, screens, or animations.

The Video icon can be used as a Sibling icon, a child of a Parent icon, or the companion of a Partner icon. It cannot be used as a parent itself.

There are several hardware requirements that you must meet before you can use the Video icon in your applications. The first requirement is a laser-disc player. The home laser-disc players that are becoming more common and much less expensive do not, unfortunately, provide the serial (RS-232) port through which AmigaVision sends its control commands. Only the more expensive industrial and professional-level players currently provide this type of port. (For a complete list of supported devices see your AmigaVision manual.)

Once you have a suitable laser-disc player, you will need to connect it to the Amiga's serial port via a special cable. (You need a null-modem cable if you are using a Sony, or a custom cable if you use Pioneer.) If you cannot find a source for such a cable, simply buy a null-modem adapter from an electronics store (such as Radio Shack) and use it with a standard serial cable.

You will also need a genlock device to combine the graphics from your computer and the video signals from the laser-disc player on your computer's monitor. There are a great many genlocks available, including a relatively inexpensive one from Commodore (the A2300) that works with A2000 and A3000 computers. While the A2300 certainly does not provide broadcast-quality output, it does a very good job of combining the computer and laser-disc video signals together for on-screen viewing.

The final requirement for using laser-disc-based video in AmigaVision is the laser disc itself. AmigaVision best supports the Standard or CAV (Constant Angular Velocity)-format laser disc, which allows full random access to any frame on the disk. Most consumer movie discs are produced in Extended or CLV (Constant Linear Velocity) formats, which *do not* allow random frame access. When shopping for disks for use with AmigaVision, make sure you get CAV format, as CLV disks do not offer full functionality with the authoring system.

To program laser-disc-based video in your applications with the Video icon, drag it from the Audio Visual menu and place it into position within your Flow window. Then double-click on the icon itself to bring up the Video requester (see Figure 10-10).

The Video requester is quite powerful and has a large number of options. It allows you to toggle the video signal, both audio channels of the laser disc, and the on-screen frame-number display. It also lets you access AmigaVision's software-based Videodisc Controller and issue over a dozen different laser-disc commands. (Note that not all laser-disc players support the full command set of AmigaVision. If you include a command in the Video icon that your laser-disc device does not recognize, the player will ignore the command.)

One of the gadgets on the Video requester is the Pause button. Like the Animation icon, the Video icon allows you to issue commands and then to either pause the application until the Video icon finishes, or to continue to execute other commands without waiting for it to finish. If the Pause button is enabled (its gadget is checked), AmigaVision will wait until the Video icon command finishes; if Pause is not enabled, the Video icon command is initiated but program execution continues.

Below the Pause button are four multistate gadgets that have three settings each: Off, On, and N/C (No Change).

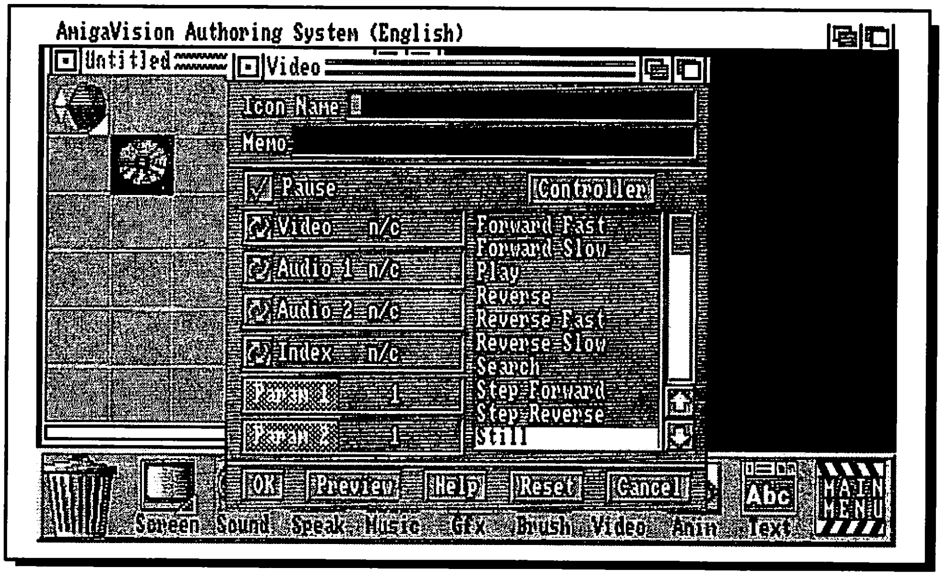


Figure 10-10
The Video requester lets you program laser-disc-based video into an application

The first gadget controls the video signal coming from the laser-disc player. Under program control, you can turn the video signal on or off. Alternately, if your command is aimed at a channel other than video (an audio channel, for instance), you can set the multistate gadget to N/C. In this case, there will be no change in the status of the video signal when the icon executes: if it is currently off it will remain off, and if it is on it will stay on.

The second and third multistate gadgets control the left and right audio channels on the laser disc. These controls can be very useful, as some instructional disks provide dialog on one channel and sound effects on another. You can elect to toggle either or both of the channels using these commands. Keep in mind that you need not use the video provided by a laser disc in order to use its audio. It is very possible, and may sometimes be desirable, to use the audio tracks from a laser disc instead of using the Amiga's sound channels. As with the video signal, each sound channel can be turned on or off independently of the other, or left unchanged using the N/C option.

The fourth multistate gadget controls the frame counter that some laser-disc players have. If the player you use offers such a counter, you will see the current frame numbers of the CAV-format disk somewhere on screen when this option is enabled. As the player advances, the frame numbers will change. Although you will seldom want to have this option enabled for the execution of your video-based applications, you will appreciate it during development. It is very handy for finding the exact sequences and frames you need for your programs. Like the preceding commands, it can be set to On, Off, or N/C (No Change).

Below the multistate gadgets are two buttons labelled Param 1 and Param 2. Clicking on either of these gadgets brings up a Specify Value requester into which you can enter numbers (or select variables) that represent frames on the video disc. If the command you are trying to execute requires less than two parameters, one or both of these gadgets will be ghosted, or unavailable. Actually, the only commands that need parameters are Play, which requires two; Search, which requires one; and AutoStop, which also requires just one parameter.

At the top of the List window, near the laser-disc commands, is the Controller button. Clicking on it brings up the Videodisc Controller requester. (The Videodisc Controller was mentioned in Chapter 4 in relation to the Tools menu and is detailed in Chapter 12.) Within the Video icon, you can use this requester to specify sequences you wish to locate on a laser disc. Then, when you exit the controller, the Video requester's Param fields will display the information you specified.

For example, if you select the Search command from the Video requester's list window and then click on the Controller button, you can use the controller requester to fill in the Param 1 field (only one parameter is required for a frame search). When the Videodisc Controller appears, you will find that it is already set to Search mode. If you had selected AutoStop or Play (the only other two commands that require parameters), you would find the controller was in the proper mode when it appeared.

Once you enter the Controller, move to the frame you want and then click on the appropriate button on the controller. As an example, if you had entered the controller with Play selected on the Video requester, at the bottom right of the Controller would be the word Play and two buttons labeled Start and Stop. Using the control buttons on the Controller, move to the desired starting frame. Then click on the Start button. A requester will appear with four options: Current Frame, Enter Frame, Memory 1, or Memory 2. If you click on Current Frame, the number corresponding to the frame that the laser disc is on will appear in the Start field. Advance the disk some frames ahead and click on the Stop button. When the requester appears again, select Current Frame as before. This time the Stop field gets the information. Now exit the requester by clicking on the OK button.

You will be returned to the Video requester and will find the Param 1 and Param 2 fields properly defined with the same values that were in the Start and Stop fields of the Controller requester.

If you had entered the Controller with Search or AutoStop selected instead of Start and Stop, you would have had only one field available: the Frame field. The procedure to define it is exactly the same as that described for Play, and when defined, its contents are returned to the Param 1 field in the Video requester.

AmigaVision supports a total of 13 laser-disc commands through the Video requester. Not all laser-disc players support all these commands, however. The following is a list of the AmigaVision laser-disc commands available from the Video requester. Those marked with an asterisk require one or more parameters.

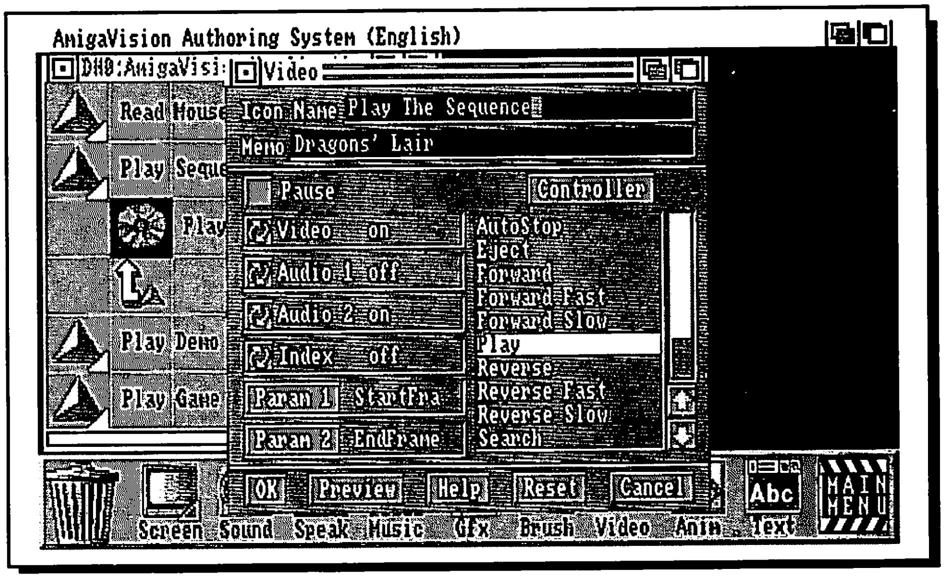
AutoStop *	- Stop disc at frame
Eject	- Open laser-disc player for disk removal
Forward	- Start playing forward from current frame
Forward Fast	- Start fast forward play from current frame
Forward Slow	- Start slow forward play from current frame
Play *	- Play disk from start frame to stop frame
Reverse	- Start disk in reverse direction from current frame
Reverse Fast	- Start fast reverse play from current frame
Reverse Slow	- Start slow reverse play from current frame
Search *	- Go to specified frame
Step Forward	- Move disc ahead one frame from current frame
Step Reverse	- Move disc back one frame from current frame
Still	- Pause at current frame

Figure 10-11 is an example of a fully defined Video requester. It is set to play a sequence of video, using variables for the starting and ending frames. The Video gadget and one sound channel are enabled; the other sound channel has been turned off. The Index counter is also disabled.

The Text File Icon

The Text File icon is used to place the contents of an ASCII-format text file onto the screen so that the user of your program can easily control them. It can present the file in a special window, in any font, using any color and in any screen mode or resolution. It also has its own string-search facility.

The Text File icon can be a Sibling icon, the child of a Parent icon, or the companion of a Partner icon. It cannot be a Parent icon. Only one Text icon can be used at a time, and it is capable of displaying only one window of text.


AmigaVision Authoring System (English)

Icon Name Play The Sequence

Menu Dragons' Lair

Pause

Video on

Audio 1 off

Audio 2 on

Index off

Param 1 StartFra

Param 2 EndFrame

Controller: Play

Figure 10-11
A defined Video requester set to play a sequence of video

To use the Text icon, you must have previously created a file using a text editor or word processor that can save in ASCII format. From the Audio Visual menu, grab the Text File icon, drag it into the Flow window, and position it in your program. Then double-click on the icon to bring up the Text File requester (see Figure 10-12).

The Text File requester offers only two unusual options: a String button and an Object Editor button. The other gadgets, common to most AmigaVision requesters, include Directory (used to select the text file), Timeout (used to specify the maximum time the text window will be displayed), Preview (used to take a look at your handiwork without having to execute the whole program), and the usual OK, Preview, Help, Reset, and Cancel buttons.

You create the text window and its control options from within the Object editor. Once you have clicked on the button and have been whisked off to the Object editor work screen, take a look at the Objects Add menu. At the bottom of the menu you will find that the Text window option is now available.

To place a Text window on the screen, you can either select the Text Window menu option or press the w key on the keyboard. Once you do, you will find that your pointer now sports a crosshair. This is the Text Window tool. To use it, position the cursor where you want one corner of the text window to be, press

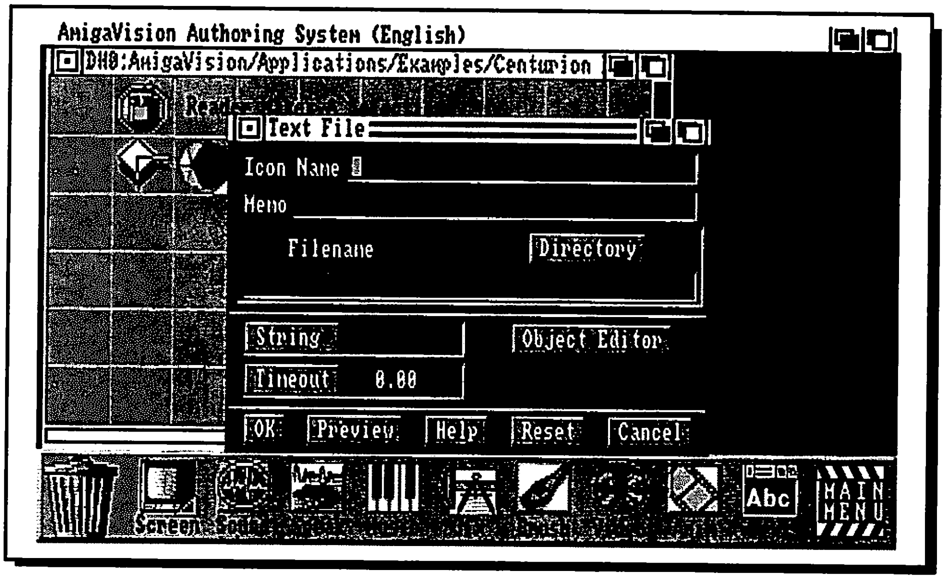


Figure 10-12
The Text File requester appears when the Text File icon is double-clicked

and hold down the left mouse button, and then drag the pointer to where you want the opposite corner. You will see that a box has formed. Once you have sized and positioned the box, release the left mouse button (you will be able to edit both the size and position later using the Object editor editing commands). Your newly created box will seem to pulse or vibrate around its outer edges, indicating it is the currently active object.

The next step is to define the visible characteristics of the Text window. To do so, double-click on the text box and a new requester, the Text Window Info requester, will appear (see Figure 10-13). From here you can specify the type style, the point size, and the colors of the text you wish to use within the window. You can also specify the window's background color, the border of the window, and the color of any highlighted text in the file. It should be noted that while you can select any font, only one font— a single type face in one point size — can be used in a window. In addition, you cannot select any special font softstyles (formatting such as bold, italic, and underlined text) from the Text Window Info requester.

Once you have defined a Text window, you must create gadgets to allow the user to control the window. There are five commands the Text window responds to: Lineup, Linedown, Pageup, Pagedown, and Quit. You can assign these commands to hot spots by entering them into the requester's Response fields (the command

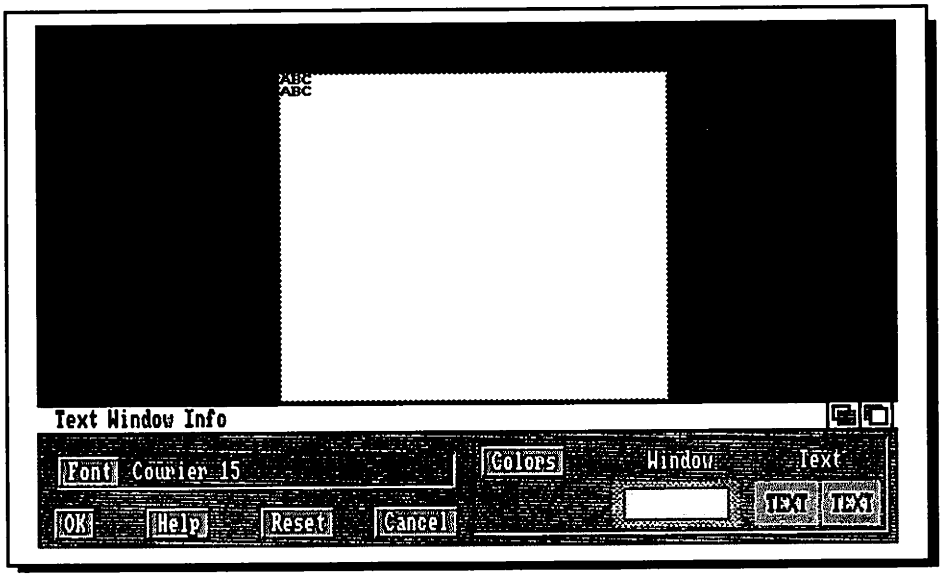
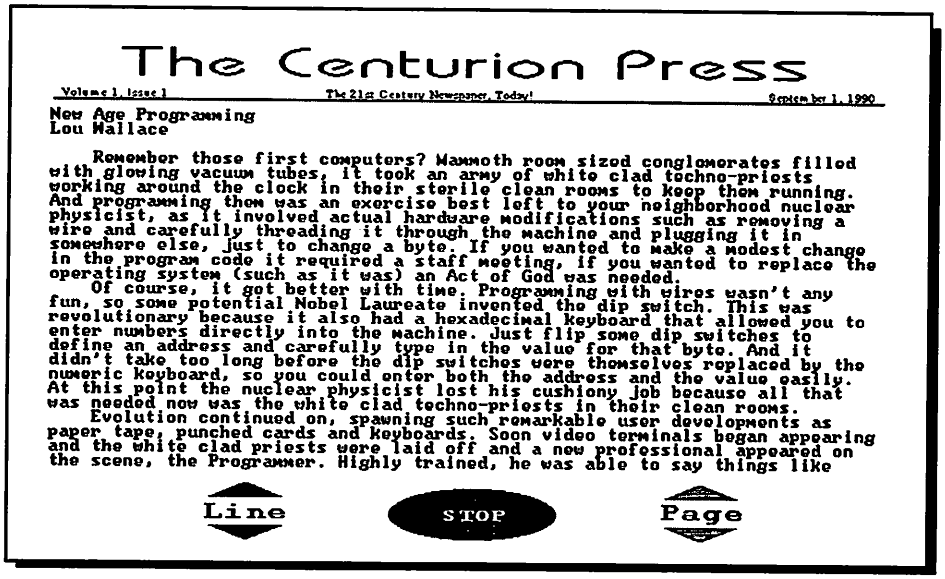


Figure 10-13
The Text Window Info requester defines visible characteristics of a Text window

strings can be all lowercase, all uppercase, or a mixture of both). If the body of the text is larger than can be displayed within a single text window, you must define gadgets that correspond to these responses so that the user is able to move through the text file.

The commands Lineup and Linedown scroll text up or down one line at a time. Pageup and Pagedown move the text one full window at a time. If the user reaches the top or bottom of the file and tries to scroll further, the commands will have no effect. The Quit command, which allows the user to get out of the Text File icon, is the only means for exit. Be sure to define a Quit hot spot; if you don't, the user will not be able to exit the Text File icon and continue with the program.

The simplest method of creating command gadgets for your text window is to draw the buttons using the Object editor. A button can be any type of object — as long as you enter the proper response into the object's response fields. I tend to sketch out my gadgets using the Object editor's graphics while I'm programming, but once the programming is completed, I generally go back and add more stylized bitmapped graphics created in a paint program such as DeluxePaint III (Electronic Arts) or Digi-Paint 3 (NewTek). (For examples of different approaches in the design of gadgets for text windows, see Figures 10-14 and 8-29.)



The Centurion Press

Volume 1, Issue 1 The 21st Century Newspaper, Today! September 1, 1990

New Age Programming
Lou Wallace

Remember those first computers? Mammoth room sized conglomerates filled with glowing vacuum tubes, it took an army of white clad techno-priests working around the clock in their sterile clean rooms to keep them running. And programming then was an exercise best left to your neighborhood nuclear physicist, as it involved actual hardware modifications such as removing a wire and carefully threading it through the machine and plugging it in somewhere else, just to change a byte. If you wanted to make a modest change in the program code it required a staff meeting, if you wanted to replace the operating system (such as it was) an Act of God was needed.

Of course, it got better with time. Programming with wires wasn't any fun, so some potential Nobel Laureate invented the dip switch. This was revolutionary because it also had a hexadecimal keyboard that allowed you to enter numbers directly into the machine. Just flip some dip switches to define an address and carefully type in the value for that byte. And it didn't take too long before the dip switches were themselves replaced by the numeric keyboard, so you could enter both the address and the value easily. At this point the nuclear physicist lost his cushiony job because all that was needed now was the white clad techno-priests in their clean rooms.

Evolution continued on, spawning such remarkable user developments as paper tape, punched cards and keyboards. Soon video terminals began appearing and the white clad priests were laid off and a new professional appeared on the scene, the Programmer. Highly trained, he was able to say things like

Figure 10-14
Custom control gadgets for a Text window

Because the Text window and its control gadgets are created within the Object editor, you also have access to other graphics and text objects while in the Text File icon. Text objects can display instructions, while graphics objects can be part of the screen design. Although you can set up any object to return a response, AmigaVision ignores responses other than Lineup, Linedown, Pageup, Pagedown, or Quit while in the Text File icon. Also, it allows the user to control these gadgets via the mouse only — there is no provision for using the keyboard.

Returning to the Text File icon requester, we find the String command button. Clicking on it brings up a Specify Variable requester wherein you can select a previously defined string variable. When you select a variable, AmigaVision searches through the text file until it finds a sequence matching your definition. When it does, it then displays the file in the window starting at the location in the file where it finds the sequence. This search is case sensitive.

In addition to being able to search for a given string in a file, you can embed control codes that affect the text's softstyles and formatting. These commands are all prefixed with the vertical bar (|). Here is a table of the formatting commands.

|B	- Bold toggle on/off
|I	- Italic toggle on/off
|U	- Underline toggle on/off

|H - Highlight toggle on/off
|FD - Normal word wrap using one carriage return
|FW - Normal word wrap using two carriage returns
|FN - No word wrap

Keep in mind that the styling and highlight commands are toggles: You must issue them once to turn them on and then reissue them to turn them off. For example, if you want to set the word "AmigaVision" in boldface in your text file, but want to leave the rest in regular roman type, use "|B" as both a prefix and suffix to the word, as in the following example:

"I found |BAmigaVision|B to be the best multimedia language in the galaxy!"

When displayed in the text window, the preceding statement would appear like this:

I found **AmigaVision** to be the best multimedia language in the galaxy!

The Speak Icon

The Speak icon (which the AmigaVision manual also refers to as the Synthesized Speech icon) is used to generate computer-synthesized speech using the Amiga's built-in speech software. It can be used as a Sibling icon, a child of a Parent icon, or as the companion to a Partner icon. It cannot be a parent.

Drag the Speak icon from the Audio Visual menu into the Flow window and position it in your program. Then double-click on it to bring up the Synthesized Speech requester (see Figure 10-15).

From the Synthesized Speech requester you can program your application to give voice to any text, from a single word or a simple text string to an entire text file. Below this requester's Memo field is a multistate gadget that toggles between Text String and Filename. When in Text String mode, you can type into the field below it a string of a maximum of 255 characters. In Filename mode, you can enter the file name and complete path in the field or use the Directory gadget (which is available in this mode) to select a text file. The file must contain pure ASCII text in order to work.

Below the text field is a series of gadgets. Those on the right are multistate gadgets that control the various voice parameters. The first toggles between Male and Female modes, giving you the choice of a low- or high-pitched voice. The second directs the sound to either the Left or Right speaker or switches the output to Stereo. The bottom gadget changes the voice from a natural, human sound to a robotic sound.

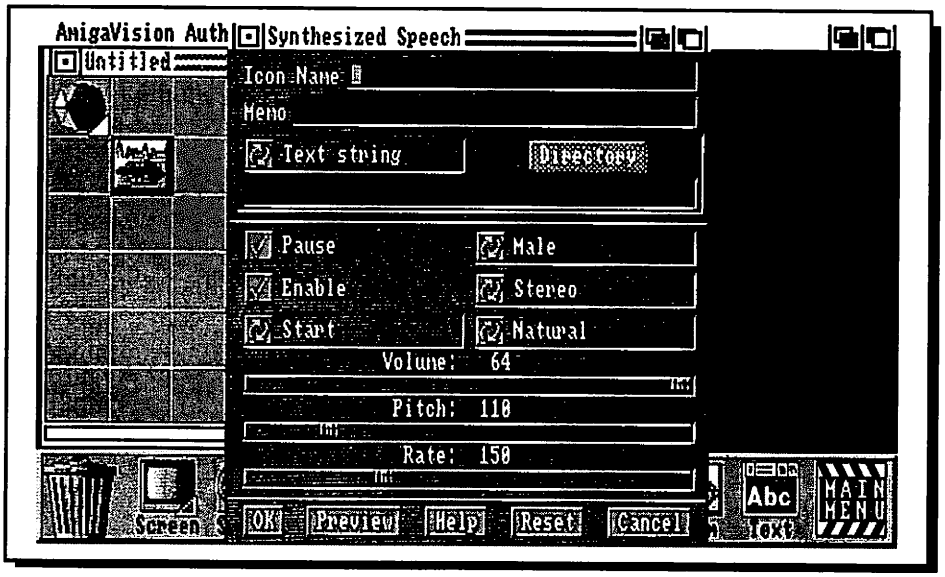


Figure 10-15
The Synthesized Speech requester appears when the Speak icon is double-clicked

On the top left is a Pause gadget. Like the Pause option found on the Animation and Video requesters, this command, when disabled, allows program execution to continue while the Amiga is executing the corresponding icon; in this case, while the Amiga's voice is "speaking." If Pause is enabled (its gadget displays a check mark), the program will pause at the Speak icon until the text string or ASCII file is completely finished.

Below Pause is a check-box gadget called Enable. When it is checked (enabled), AmigaVision uses all the settings on the Speech requester to define the way in which the speech is generated. If not checked, the speech is generated using the previous Speak icon's settings, and the current settings are ignored.

Another multistate gadget, this one below the Enable button, toggles between Start and Stop. You can use this gadget to allow the user to silence a Speak icon.

At the bottom of the requester is a set of three sliding gadgets that allow you to change the volume, pitch, and rate of the speech. These attributes are controllable via the sliding gadgets only and cannot be directly controlled through variables.

Just as the Text File icon allows special control codes in its text files, so does the Speak icon. You can include these commands within the text file and change the voice parameters 'on the fly.' The commands must be entered in the correct order (rate, pitch, mode, sex, volume) and enclosed by vertical bars, as in the following example:

| Rate, Pitch, Mode, Sex, Volume |

Each parameter has a specific range of values, equal to what you can set from the Synthesized Speech requester. The ranges for the various parameters are as follows:

Rate	40-400
Pitch	65-320
Mode	0-1 (0 is natural, 1 is robotic)
Sex	M-F
Volume	0-64

Besides allowing you the flexibility to change voice attributes within your text files, these parameters compensate for the lack of variable support for the Speak icon. By creating a string variable that contains the parameters you want, you can use the Database Output icon to write the control string to a file (usually a temporary file in the RAM disk). You can also set up the Speak icon to configure its voice according to the parameters defined in your variable string. Finally, you can "speak" any string or file (with Enable off) using these parameters. When you need to change the voice parameters, just create a new string, save it to the RAM disk, and start over!

Even though you can output the speech to a specific sound channel and turn off the Pause option so that speech can continue while other icons are executing, only one Speak icon can execute at a time. If you attempt to run more than one simultaneously, you will get an error message, followed by a requester asking whether you want to continue or cancel the presentation.

The Digitized Sound Icon

The Digitized Sound icon is used to play digitized sound files in standard Amiga 8SVX format.

The Digitized Sound icon can be a Sibling icon, a child of a parent, or the companion of a Partner icon. It cannot be a parent itself.

To use the Sound icon, drag it from the Audio Visual menu into the Flow window and position it in your program. Then double-click on the icon to bring up the Digitized Sound requester (see Figure 10-16).

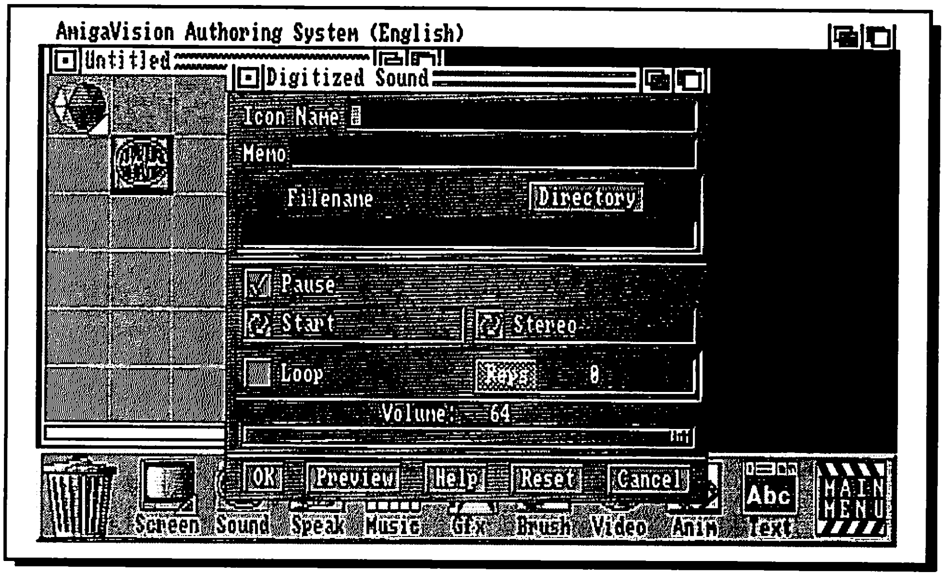


Figure 10-16
The Digitized Sound requester appears when the Digitized Sound icon is double-clicked

The Digitized Sound requester is relatively simple to use. Using the Directory command, select the 8SVX file you want to play. Then, depending on your needs, either enable or disable the Pause option.

If you are just starting the sound, set the multistate gadget to Start. If you want to stop a sound in progress, make sure its name is in the Filename field and then set the multistate gadget to Stop. The multistate gadget also has an option called Stop All. Select this and all the sounds that are playing will cease; you need not specify any file names to use this option.

If you want to play a sound more than once, click on the Loop check-box gadget to enable it, and then use the Reps button to set the number of times to play the sound. Remember that if set to zero, the sound will loop indefinitely.

The volume level for sound playback ranges from 0 (off) to 64 (full volume). You control the volume via the sliding gadget at the bottom of the Digitized Sound requester.

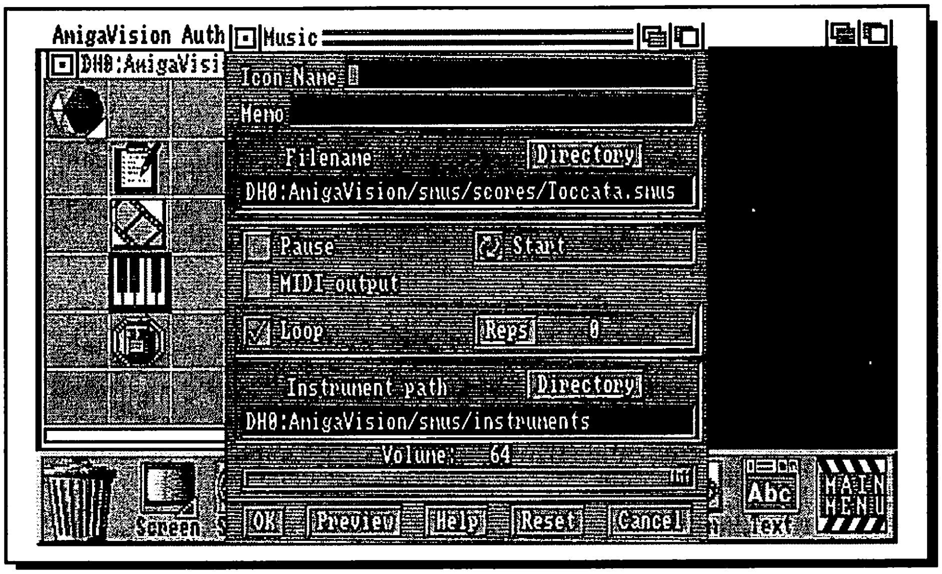
You can specify the channel(s) you want the sound to play through using the multistate gadget on the right of the requester. If you want dual-channel sound, leave this gadget in its default setting of Stereo. If you prefer to send your sound out through one channel only, select either Left or Right. With the Digitized Sound icon, it is quite possible to have different sounds coming out of the different speakers simultaneously. If all your sounds are monophonic, you can play four (two per sound channel) at once. Because all stereo samples require two channels, you can play only two of these (or one along with two mono samples) at the same time. The Sound icon is quite flexible, but be careful to not use more than four channels at once.

If you want to hear your sound sample play, click on the Preview button at the bottom of the requester.

The Music Icon

The Music icon allows you to play musical scores and instruments in the SMUS format used by Electronic Arts' Deluxe Music Construction Set (DMCS).

The Music icon can be used as a Sibling icon, the child of a parent, or as the companion to a Partner icon. It cannot be a parent. To use the Music icon, drag it into the flow chart and position it in your program. Then double-click on it to bring up the Music requester (see Figure 10-17).



Icon Name
Memo
Filename Directory
DH0:AmigaVision/smus/scores/Toccata.smus
Pause ✓ Start
MIDI output
✓ Loop Reps 0
Instrument path Directory
DH0:AmigaVision/smus/instruments
Volume: 64
OK Preview Help Reset Cancel

Figure 10-17
The Music requester appears when the Music icon is double-clicked

You can enter the path and file name of the music score you wish to use directly into the Music requester. Alternately, you can use the Directory gadget to bring up the standard file requester and select the SMUS music file with your mouse.

Like the requesters corresponding to the other sound commands, the Music requester has a Pause button that you can enable if you want AmigaVision to wait until the music has finished before continuing with the rest of the program. If Pause is not enabled (not checked), the music file is started and AmigaVision then goes on to the next icon. This feature allows you to play music while running animations or video, or to add a soothing background score to a database or educational application.

You can use the Start/Stop multistate gadget in this requester to initiate a specified music file (this is the default), or to stop a song that is currently playing. (To stop the file you must explicitly designate its file name in the Filename field, just as you did when you started it.)

Keep in mind that SMUS files use all the available sound channels on the Amiga. This means that, while music is playing, you cannot play digitized sounds with the Digitized Sound icon or speech with the Speak icon. It also means that while music is playing you cannot also use a digitized sound as feedback in the hot spots created with the Object editor. If you try to, you will overload the sound channels.

AmigaVision supports MIDI output via the Music icon. If you have a MIDI interface properly connected to a keyboard and want to use it to generate music for your application, enable (check mark) the MIDI Output option on the requester. Beware, however, that MIDI support is limited to four channels, at least in the current version of AmigaVision. You can output a SMUS file to a MIDI device, but you cannot send a MIDI file to a MIDI device.

The Loop gadget lets you instruct AmigaVision to play the music file more than once. When it is enabled, you can click on the Reps button next to it, which brings up a Specify Value requester. Here, you can enter the number of times to repeat the song. A value of zero means the song will play continuously.

At the bottom of the requester is the standard Volume slide control gadget. You can choose from 0 (sound is off) to 64 (full volume). As with the other sound icons, there is no way to control music volume using a variable; volume must be explicitly set here on the requester.

If you have specified an Instruments directory in the Defaults requester (see Chapter 4), the Instrument field will contain that path. If not, you must use the Directory command next to the Instrument Path field to specify the path AmigaVision needs to find the instruments. Please note that any instruments you use in your AmigaVision applications must be in the same format as those used by Electronic Arts' DMCS. While other Amiga music packages support the SMUS format for the musical scores, not all use the same format for instruments.

– 11 –

The System Command Menu

As you have seen, AmigaVision's command icons are organized into groups based on function. Graphics and sound comprise one group, database functions are in another, and interrupts compose yet another group. This grouping by function is continued in this chapter on the System menu (called the Module menu in early versions of AmigaVision).

The System Command Icons

At first glance, the icons in this menu seem to be more different than similar. Here we find the Subroutine and Module icons, which allow you to differentiate the various parts of your programs into discrete components. We also find the Return and Quit icons, which give you an escape route from either a subroutine or the entire program. However, from the Resource icon's management of the data structures of sound and graphics to the Execute icon's ability to use other programs from within AmigaVision, we see two common threads. The first is a concern for the internal structure of AmigaVision programs, which is evident in such commands as Module and Subroutine (and their respective exit icons) and in the Timer icon (which measures elapsed time within sections of your applications). The second thread concerns the use of data generated outside of AmigaVision as well as the use of external applications that are controlled by AmigaVision.

To access the System icons, click on the icon labeled System in the Main menu. This will switch you to the System menu (see Figure 11-1). On this menu are the Module, Quit, Subroutine, Return, Timer, Resource, and Execute icons.

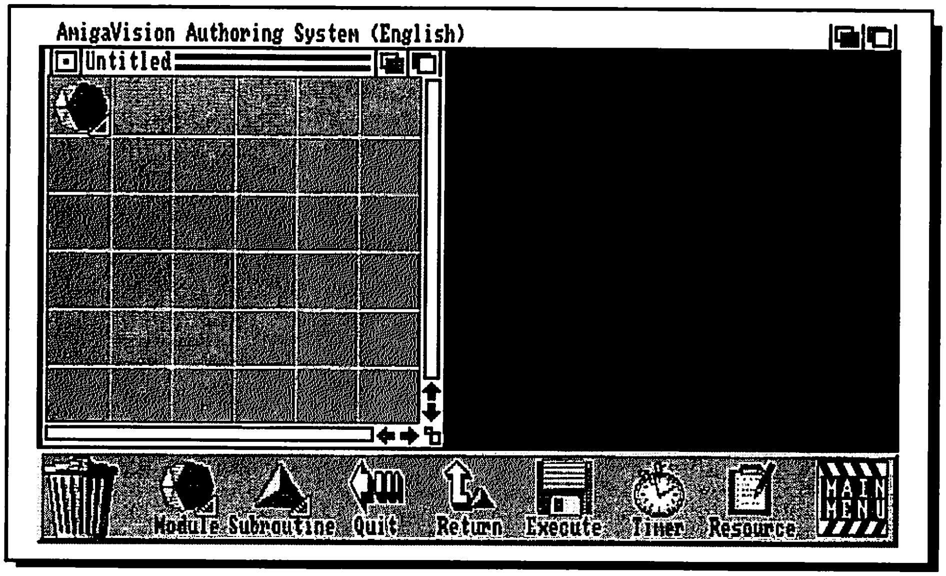


Figure 11-1
The System menu commands

The Module Icon

Every AmigaVision flow chart begins life with a Module icon in place. You can get rid of this Module icon by placing another icon under it and then dragging the Module icon into the trash, leaving only the icon you added. *But,* try to get rid of all the icons in a window by dragging them out of the window and into the Trashcan icon and what happens? AmigaVision replaces them with a new icon, and not just any icon — it specifically adds a Module icon. Why the Module icon? Because the Module icon is the ultimate AmigaVision organizer. As you become more proficient with AmigaVision programming, you will discover that the Module icon is essential to developing sophisticated software.

The Module icon can be a Sibling icon, a Parent icon, the child of a Parent icon (including another Module icon), or the companion to a Partner icon.

In almost all cases, your programs will begin with a Module icon as the parent of all the other icons. The Module icon is the ultimate Parent icon, allowing every other icon — except for the Subroutine icon — to be used as its children. You can use Module icons to separate the various parts of your programs into sections that can be executed one after another. They can create variables accessible by only the children of the Module icon. Finally, they can be the companions of Partner icons (such as the If-Then and If-Then-Else conditional commands) to extend the influence of the partner beyond a single command.

To add a Module icon to your program, drag it from the System menu into the flow chart and position it in your program. Because a Module icon can be used to create a new section of a program, there is no need to define it as there is with most other commands. If you wish to name the icon, however, (a highly recommended practice) or use it to create variables local to the children of the Module icon, double-click on the icon to bring up the Module requester (see Figure 11-2).

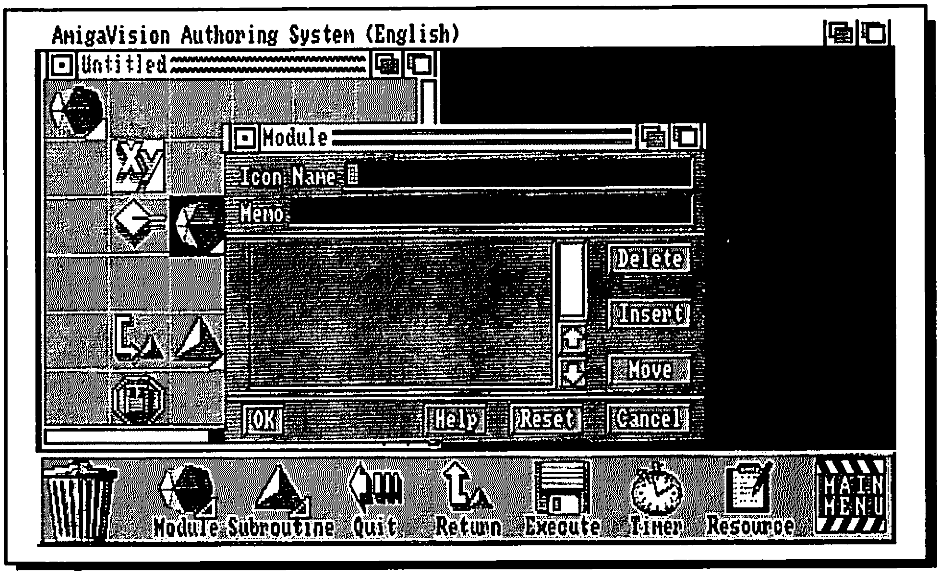


Figure 11-2
The Module requester is an exact duplicate of the Variable requester

Notice that the Module requester is not only very similar to the Variable requester, but is an exact duplicate. The reason for this is that one of the two major functions of the Module icon is to create variables for the use of its children. These variables are known as local variables because they are limited, or localized, in scope. They have meaning to only the children of the Module, and the children of its children.

To create a local variable, click on the button labeled Insert. Like the Insert gadget on the Variable requester, this button summons the Expression editor. Within the

Expression editor you create your variable using the names and initial values your program will require.

```
Firstname = "James"
Lastname = "Key-Wallace"
Name = ""
Price = 24.95
Distance = 1200
Today = Date()
Tax = 0.00
```

These variables are available for use by the children of the Module icon. They can be used as is or modified in a variety of ways. For example, you can combine several string variables into one single variable or use one numeric value to calculate another.

```
Name = Firstname + "z" + Lastname
Tax = Price * 0.05
```

If you need to have a variable with wider accessibility, you should use the Variable icon to create global variables, which are available from everywhere within your program. There are no restrictions on using global and local variables together in the same expression. If one of the children of the Module icon is a Variable icon, any variables created with the Variable icon are global. Only those made with the Module are local.

Module icons can serve as parents to the major sections of your program, and thus help delineate one section from another. Figure 11-3 shows a section of a program broken down into different levels, or sections. Each Module icon contains a large amount of AmigaVision code, which is needed to perform the actions of its respective level.

Because of the limitations posed by the fact that Partner icons are allowed only a single companion, AmigaVision programs should, by rights, be under some severe constraints. You can, however, extend the effectiveness of a Partner icon by using a Module icon as its companion. Then you can add any number of commands as children of the Module (see Figure 11-4).

The Quit Icon

The Quit icon is the command that allows you to exit from an AmigaVision program. You can program your application to stop in any number of ways and for

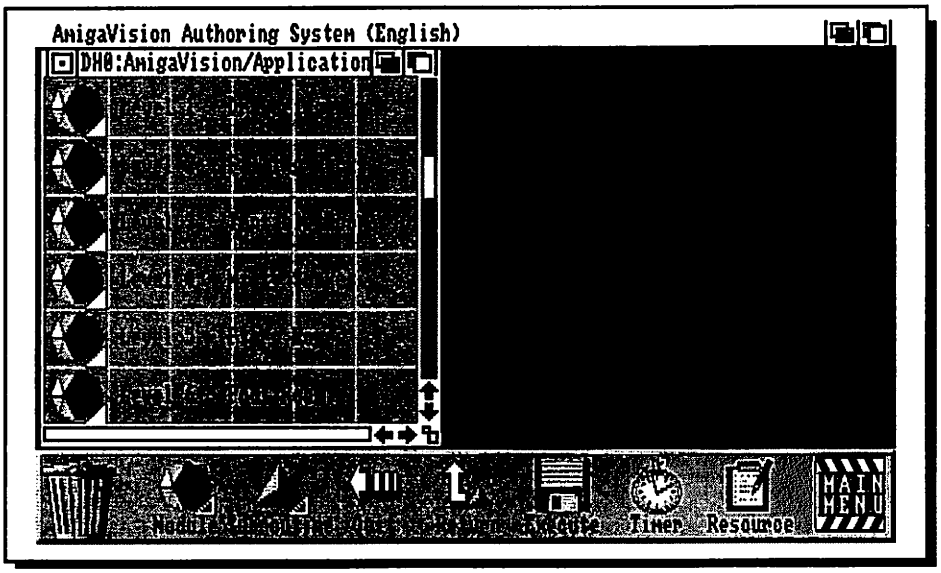

AmigaVision Authoring System (English)
DH0:AmigaVision/Application
Timer
Resource
MAIN MENU

Figure 11-3
The Module icon used to separate sections of a program

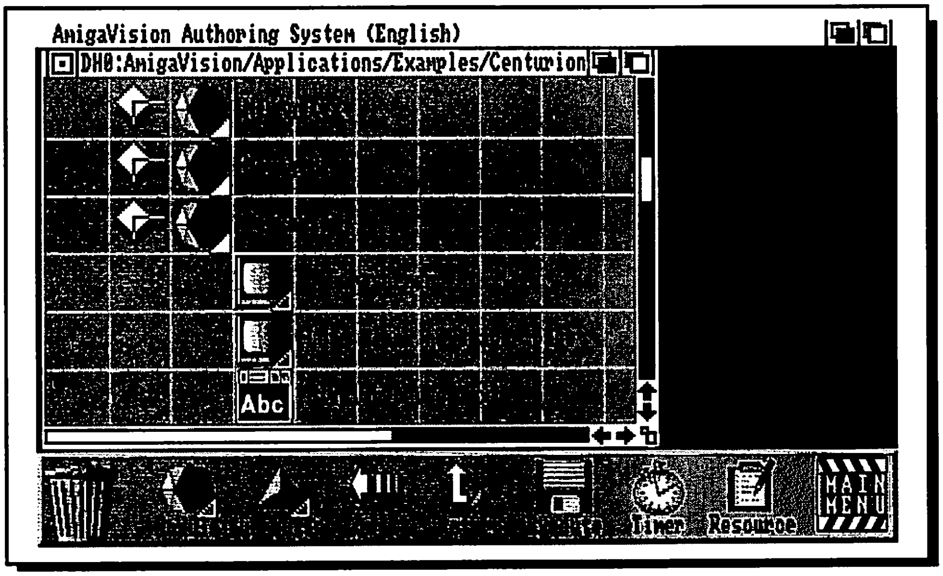
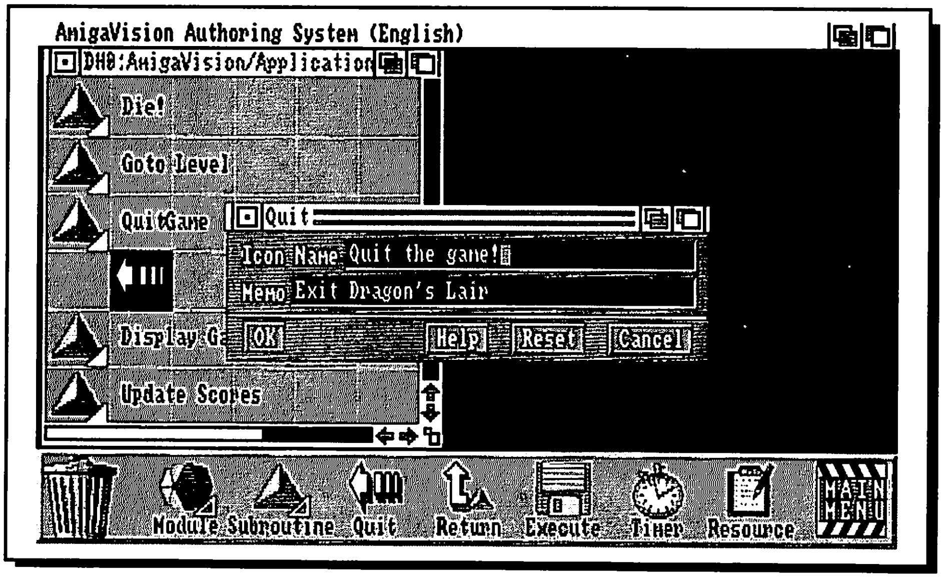

AmigaVision Authoring System (English)
DH0:AmigaVision/Applications/Examples/Centurion
Abc
Timer
Resource
MAIN MENU

Figure 11-4
The Module icon as a child, partner, and parent

any number of reasons. Your program can quit in response to a simple menu selection by the user, or in response to a lack of activity for some specified period of elapsed time. You can even set up the application to quit when it finishes the task it was designed to accomplish. Whatever the reason, there has to be a way out of any program, and the Quit icon is the command to allow it.

The Quit icon can be a Sibling icon, a Child icon, or the companion of a Partner icon. It cannot be a parent.

To use the Quit icon, just drag it into the Flow window and position it in your program at the desired location. The Quit icon has no parameters. It does, however, have a requester that you can use to name the icon and make any necessary memos. Figure 11-5 shows the Quit requester in all its glory.



Icon Name: Quit the game!

Memo: Exit Dragon's Lair

Figure 11-5
The Quit requester is used to name a Quit icon or make necessary memos

What happens when you exit an AmigaVision program depends on how the application was started. If you began the program from the AmigaVision editing environment, you will return to that screen. If you started the program as an edit-protected run-time system from either the Workbench or the CLI, you will return there when you exit. If you have set one of the options in the AmigaVision icon to Edit, and if the program is not edit protected, you will return to the editor when you exit the program — even if you loaded it from its own icon and not from the AmigaVision editor. If you load the program from the Shell/CLI environment with the "a" command, you will return to the Shell or CLI when you quit the program.

AmigaVision:AV aMyProgram.AVf

The Subroutine Icon

The Subroutine icon, as its name suggests, is used to create subroutines within an AmigaVision program. A subroutine is usually a section of code that must be executed repeatedly or throughout the application. It is very convenient to be able to call a subroutine when you need it. In addition, it is also a very efficient use of resources to set up only one easily accessible routine instead of sprinkling multiple copies of the same code throughout your program.

The Subroutine icon can be a parent to any other icon except another Subroutine icon. Because it must always start in the leftmost column of your flow chart, it can be considered a sibling of the initial Module icon and any other icons in the far left column. It cannot be a companion to a Partner icon, and it cannot be a child of a Parent icon. (A duplicate of the Subroutine icon being referenced by a call icon will appear as the reference icon image in the call statement, as Figure 11-6 shows. Keep in mind that this image is not a Subroutine icon, but merely a representation of one.)

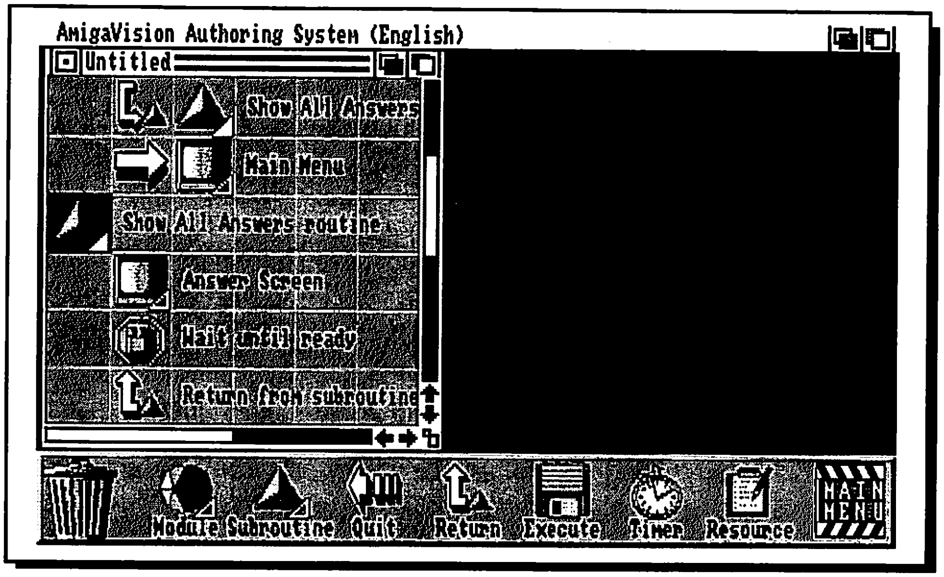


Figure 11-6
A referenced subroutine which represents a Subroutine icon

To add the Subroutine icon to your program, drag it from the System menu into the Flow window and position it in the leftmost column of the window. Generally, it is a good idea to group all your subroutines below the main body of the program, separating them from the main application with either a Quit icon or an absolute Goto icon that branches program flow back to the body of the program.

The Subroutine icon, like the Variable and Module icons, can be used to create variables for other icons to act upon. As with the Module icon, the variables that the Subroutine icon creates are local and have meaning only to its own children. To define the Subroutine icon or to create local variables with it, double-click on it. The Subroutine requester will appear (see Figure 11-7).

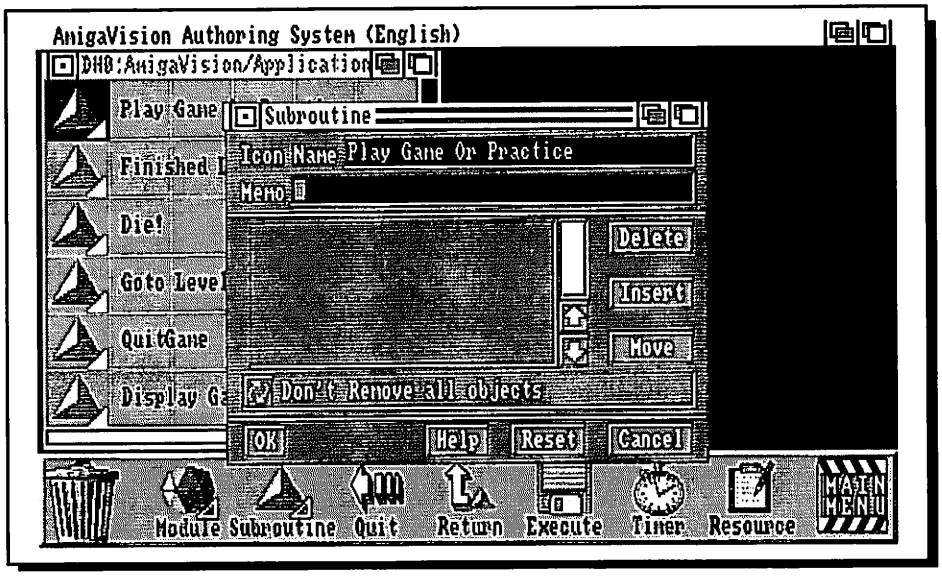


Figure 11-7
The Subroutine requester can be used to create new variables

As you can see from Figure 11-7, the Subroutine requester is very similar to the Module and Variable requesters. You can use it to create new variables: simply click on the Insert button to bring up the Expression editor, wherein you can define the variables. You can delete variables previously created from within the Subroutine icon by highlighting them with the mouse and clicking on the Delete button. You can also reposition variables in the list window using the Move button. To relocate a variable, click on it once to highlight it, click on the Move button, point to the spot in the list window where you intend the selected variable to go, and then click with the mouse. The highlighted variable will jump from its old position to the new location.

The Subroutine requester has an additional gadget — a multistate gadget — at the bottom of the variable list window. This gadget allows you to give AmigaVision instructions about how to treat any graphic objects it finds on the screen when the Subroutine is called. For example, if you had used the Graphic icon to place a brush, polygon, circle, rectangle, text, or variable on screen, you might want those things to be removed while the subroutine executes. On the other hand, you might want these things to remain on the screen while the subroutine runs.

In either case, the multistate gadget comes in handy. It has two states, or modes of action. The first is "Don't Remove all objects;" the other is "Temporarily Remove all objects." If you enable "Don't Remove all objects," everything that is on the screen when the subroutine is called remains there during the course of the subroutine's execution. If you select "Temporarily Remove all objects," AmigaVision removes those objects from the display while the subroutine is executing, but restores them when the subroutine finishes. Any objects added to the display during the course of the subroutine's execution are removed when that subroutine finishes. However, if you change the screen itself — by using the Screen or Animation icons to display a new screen image while in the subroutine — that picture remains when the subroutine finishes. If you want the old screen image to be restored, you must restore it yourself.

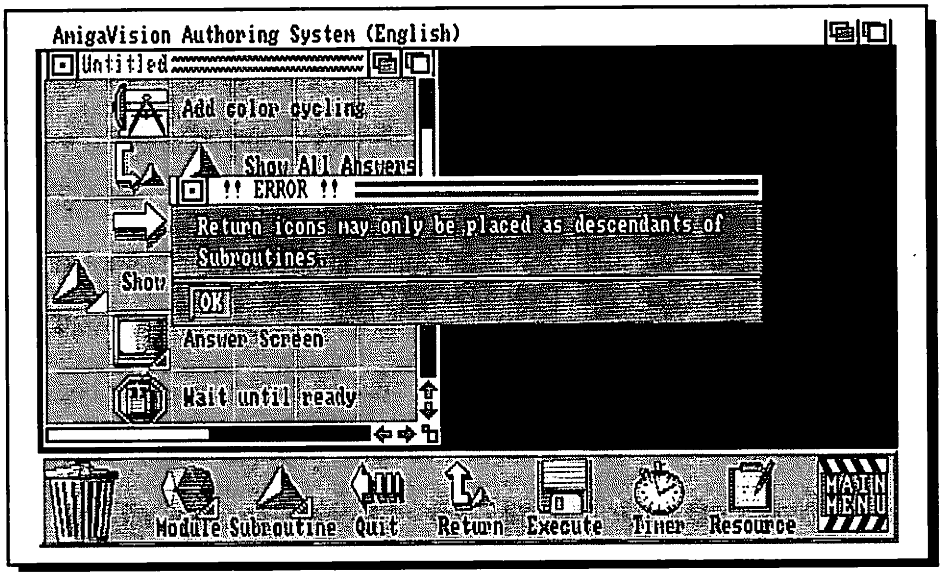
The Return Icon

The Return icon is used to exit a subroutine and return program execution to the icon immediately following the call icon from which the subroutine was accessed. The Return icon explicitly stops execution of a subroutine.

The Return icon must be placed as a child of the Subroutine icon. It can be a sibling or the companion of a Partner icon, although it cannot be a parent.

The *AmigaVision User's Guide* includes some contradictions and errors in its description of the Return icon. It states the Return icon will be ignored if encountered in the main body of your programs, then later notes that if a Return icon is encountered without a call, it has the same effect as an Exit icon.

The fact is that you cannot place a Return icon in the main body of an AmigaVision program using version 1.53G. (In the upgraded version 1.7e, you can place a Return icon in the main body of the flow.) If you attempt to do so, an error requester will appear to inform you that "Return icons may only be placed as descendants of Subroutines" (see Figure 11-8). So it is not true that the Return icon is ignored in the main program. In fact, Return icons cannot be used in the main program.



!! ERROR !!

Return icons may only be placed as descendants of Subroutines.

OK

Figure 11-8
The Return icon error requester appears if you attempt to place a Return icon in the main body of a program

Still, it is possible to execute a Return icon without explicitly calling a subroutine. This can happen if your program executes the subroutine without the help of a Call icon. For example, suppose you have defined a conditional or absolute Goto icon to jump to a subroutine (although AmigaVision allows you to do so, I recommend you avoid it). If this subroutine contains a Return icon (as would likely be the case) the program will not send you back to the main program when it reaches that icon because the subroutine was never called in the first place. Instead, the Return icon will act as an Exit icon and stop the program cold.

The other possible scenario is if the last icon in the program is not an Exit icon or a branch to another location in the main program. In this case, if a subroutine follows the main body of the program, when the program flow reaches the icon just above a subroutine (see Figure 11-9 for an example of this type of error) it will move on and execute the Subroutine icon, just as if it were a Module icon. Then, unless AmigaVision encounters a Return icon, program execution stops.

The Return icon is not the only way to induce a return from a subroutine — it is just the explicit method. If a called subroutine simply finishes and the program executes the last icon of the subroutine, AmigaVision treats this as an implicit return from the subroutine. Program flow then returns to the main body of the

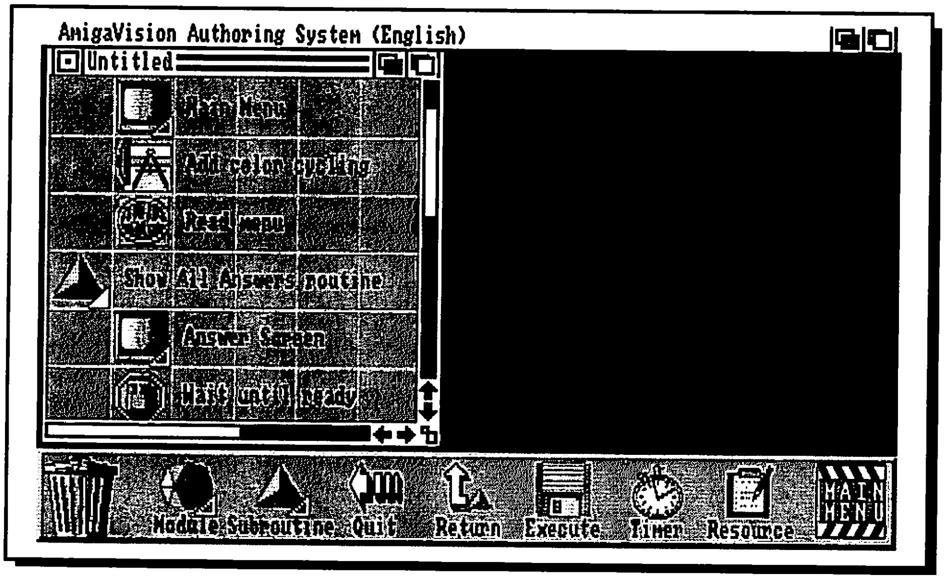


Figure 11-9
Error! Running past the end of the program and into a subroutine

program — to the icon immediately following the icon that referenced the subroutine — just as it would upon encountering a Return icon.

The Timer Icon

There are times when you need to measure the amount of time that has elapsed between events. The best way to do this in AmigaVision is via the Timer icon.

The Timer icon can be a child of a Parent icon, a sibling, or the companion of a Partner icon. It cannot be a Parent icon.

The Timer icon measures time in seconds and is accurate to two decimal places, which is to say that it reports its results to 1/100 of a second. AmigaVision allows up to nine timers to be active at once. To use the Timer icon, drag it from the System menu and position it in your program. Then double-click on it to bring up the Timer requester (see Figure 11-10).

The Timer requester has two important features. One is a command button labelled Timer #, which, when clicked, brings up a Specify Value requester. Here you can enter a numeral from 1 to 9. Entering a number larger than 9

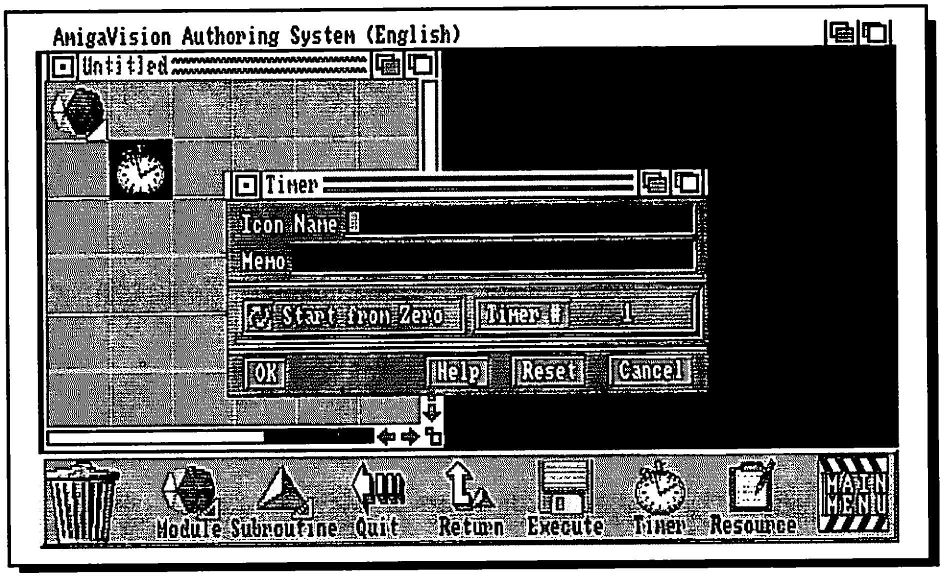


Figure 11-10
The Timer requester appears when the Timer icon is double-clicked

will generate an error. You can also use variables, which are available in the requester, to specify the Timer.

The other option is a multistate gadget that controls the mode of action of the Timer icon. It can exist in one of three states: Start From Zero, Stop, or Restart. To initiate a timer you must use the Start From Zero mode. After that, you can stop and restart it using the other two modes. In this sense, the Timer command is a toggle — it takes one Timer icon to start it working and another to make it stop.

To find out the value of a Timer, you must enter the Timer() function in the Expression editor. This function requires a parameter to indicate the Timer you want information about: Timer(1) returns information about the first timer, Timer(7) gives data on the seventh timer, and Timer(*i*) gives you the value of the "ith" timer, with *i* being a number from 1 to 9.

You can use the Timer() function in your programs to return a value to a variable, as in "Delay = Timer(*i*)," and you can use either the variable or the function in a more direct manner as a component of a conditional expression, as in the following:

```
Delay >= 60.0
Timer(i) >= 60.0
```

Because the Timer returns values using two decimal places, it is not a wise idea to use timer values in conditional statements that are absolute in nature. Here are a couple of examples:

```
Delay == 73.10
Timer() == 10.00
```

In these cases, your expressions will not evaluate as true unless Delay equals exactly 73.10 or unless Timer() is exactly 10.00, both of which could be unlikely. Try to use such operators as greater than (>), less than (<), greater than or equal to (>=), or less than or equal to (<=) when working with the results of the Timer icon.

The Resource Icon

The Resource icon is your key to seamless presentations and effective memory management. On Amiga systems that have ample memory (with or without limited disk-storage capacity), you can use the Resource icon to load large data files — including pictures, brushes, music, digitized sounds, and animations — into RAM. This allows you to avoid delays during the course of a presentation while information is loaded from disk.

The Resource icon can function as a sibling, as a companion to a Partner icon, or as a Child icon. It cannot be used as a parent.

The Resource icon is usually found at the beginning of a program, but you can add it anywhere else you find it to be expedient and useful. To use the Resource icon, drag it from the System menu and place it into position in the Flow window. Then double-click on it to open the Resource Control requester (see Figure 11-11).

On this requester is a multistate gadget that toggles the command mode between Load Files and Unload Files. You see, the Resource icon can be used not only to load data files into memory, but also to unload (remove) them from memory. The action (load or unload) is performed on the files contained in the list window. To select a file, you can either type its name (including the complete device and directory path) into the Filename field, or use the Directory gadget to select the file name from the standard AmigaVision file requester.

Once you have selected the file (either by typing it in or double-clicking on it in the file requester) you are returned to the Resource Control requester. The name of the file you chose now appears in the Filename field. To accept it, click on the Insert button on the right side of the requester. This places the file into the list window. You are not limited to choosing one file at a time; you can select multiple files and affect them all with the same Resource icon.

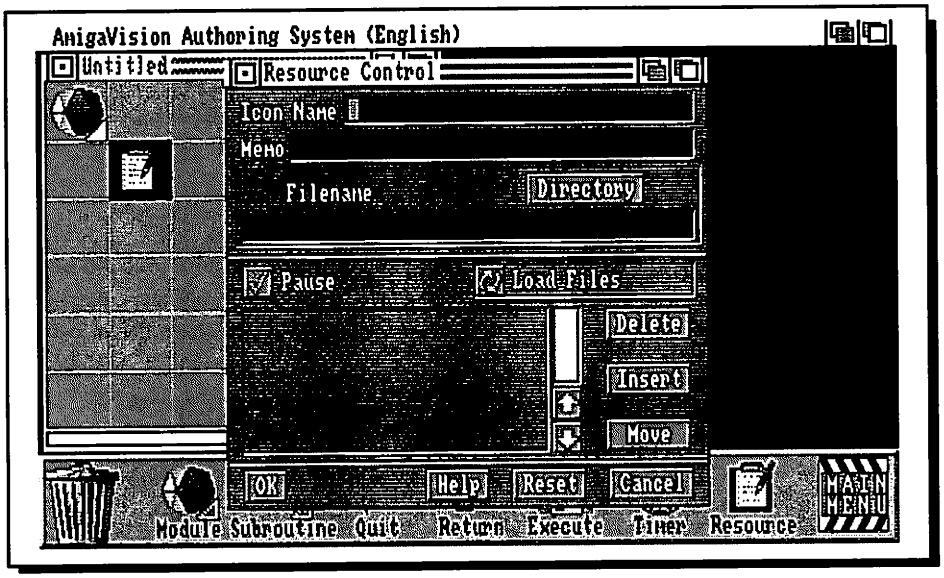


Figure 11-11
The Resource Control requester is used to load or unload data files into memory

Through the Resource Control requester, you can delete files from the list window by clicking on their names once to highlight them and then clicking on the Delete button (which is located above the Insert button). The order in which AmigaVision loads the files is the order in which they appear in the list window. If you want some files to load before others, use the Move option to position them at the top of the list. To shift a file, click once on its name to highlight it and then click on the Move button. Your pointer will change slightly to display a horizontal bidirectional arrow, which indicates that you are in Move mode. Just point to the location in the list where you would like to place the selected file and click the mouse button: The highlighted file will move to this position, and any other files below it will move down one slot to make room. Once you have done this, the pointer returns to normal and you are no longer in Move mode. You can also insert a file indicated in the Filename field by highlighting the position in the list you want to place it, and then clicking on the Insert button.

The Resource Control requester has one check-box button labelled Pause. As with other commands that provide this control, enabling the Pause option forces AmigaVision to stay at the Resource icon until its actions are finished. When not enabled (no check mark appears) AmigaVision initiates the actions requested, but moves on to other icons while the Resource icon is still executing. This feature allows you to load information you will need later while you present other parts of your application, and thus greatly improve your program's overall performance. Likewise, you can unload data contained in memory while you perform other actions.

Once you load a file into the Resource icon, that file is available for use by other icons. For example, if you load an IFF picture into RAM and then call it from a Screen command (using the exact name and path), the picture will appear on screen much more quickly than if you had summoned it from disk — especially from a floppy disk.

If you attempt to load a file into RAM that is larger than your memory bank can handle, AmigaVision ignores the load instruction. When your application finishes, however, AmigaVision automatically frees all memory allocated by the Resource icon.

Some types of data cannot be accessed through the Resource icon's RAM buffers. For example, AmigaVision does not call from memory the text files that the Text icon uses, but instead loads them from disk as it needs them. A database file is another example of a file that you cannot access via the Resource icon. Also, while you can load music scores into RAM, the instruments they use do not automatically load with them.

The Execute Icon

You can use the Execute icon to run external programs from within an AmigaVision application. You can start these programs from either the CLI or the Workbench. In addition, AmigaVision can execute ARexx script files and communicate with programs that are equipped with an ARexx communication port by way of the Execute icon.

The Execute icon can be a Sibling icon, a child of a Parent icon, or the companion to a Partner icon. It cannot be a parent itself.

To use the Execute icon, drag it from the System menu and place it into position in the Flow window. Then double-click on it to bring up the Execute File requester (see Figure 11-12).

The Execute requester's primary features consist of a Directory command, two multistate gadgets, and two command buttons. The Directory button brings up the standard AmigaVision file requester, wherein you can select the program or file to be executed. Once you choose a file, it appears, along with its complete path, in the Filename field. As usual, you can also click in the field and manually

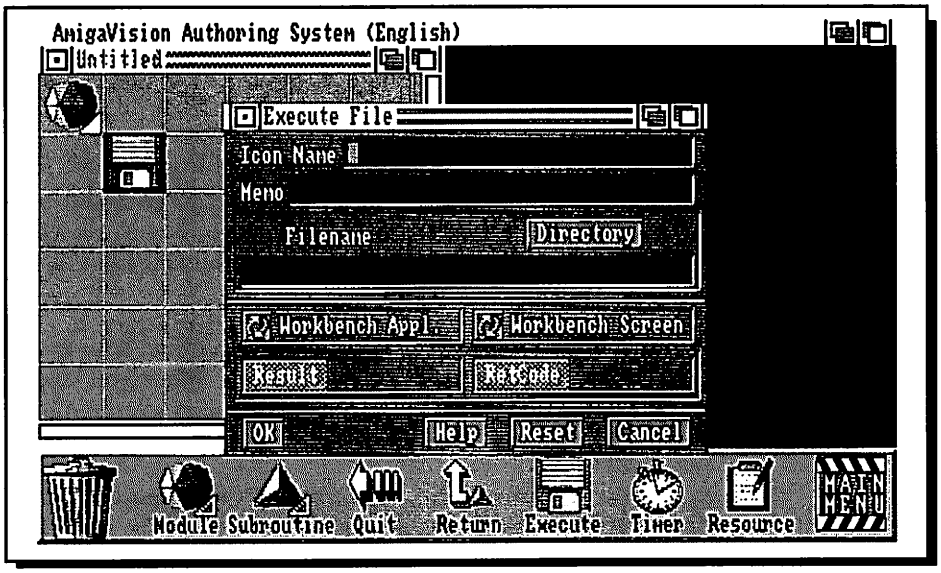


Figure 11-12
The Execute File requester appears when the Execute icon is double-clicked

enter the path and file name of the command you want to use. Keep in mind that it is very important to use the complete file name. Entering the command name alone is not sufficient — even for files for which you have assigned paths or for commands that are resident in the AmigaDOS shell.

For example, suppose you wanted to call up the directory of a disk and save it as a text file for later use in a text window. You would set the left multistate gadget on the Execute requester to read CLI Appl., place the directory command in the Filename field, and redirect the output to a text file. Here is an example of an incorrect entry:

```
DIR >ram:temp work:
```

Because the complete path is not included in this entry, the string would not execute properly and would generate an error requester reading, "Command specified in Execute icon cannot be performed." Following that message, another requester would appear to ask whether you wanted to continue or cancel the application.

If your entry in the Filename field of the Execute File requester includes the complete path name, however, the command will work fine. Here is the corrected version of the previous example:

```
SYS:C/DIR >RAM:TEMP WORK:
```

One multistate gadget allows you to choose how you wish to execute the programs. It toggles between ARexx, CLI, and Workbench Applications. When executing CLI commands, you can include any command-line options that the command normally allows. The trick to using CLI is to include the full path name.

A program that is executed using Workbench mode starts as if it had been double-clicked with the mouse. In most cases, if a program has an icon, you can start it in this mode. Generally, if a data file (a picture or animation, for example) has an icon that specifies a default tool, it too can be started with the Workbench command. A good example of this type of data file is a picture saved from within DeluxePaint (Electronic Arts). If you have the default tool set properly (i.e., if the icon's .info file contains the proper path to find the DeluxePaint program, which can display the picture), you can designate the picture as a Workbench application using the Execute command. Doing this would involve a lot of unnecessary work, however; using the Screen icon command to display your picture file is much simpler. The difference is that when the Execute icon executes the picture as a command, it loads DeluxePaint as well as the picture! With the paint program loaded, you can edit the picture and then resave it. When you subsequently exit DeluxePaint, you are immediately returned to the AmigaVision application that called it in the first place.

The ARexx option allows you to execute ARexx scripts, which themselves can load and control other applications. In addition, you can use the Filename field to send specific strings of ARexx commands instead of indicating a script to execute. (AmigaVision and ARexx consider the string to be a complete, small, ARexx script.) ARexx — the Amiga version of REXX — must be running in order for this option to work.

REXX, if you are not familiar with it, is a simple but powerful computer language used primarily for interprocess communication (IPC). With it, programs with IPC ports can communicate with, and in many cases be controlled by, other programs — including AmigaVision programs.

The Amiga version of REXX was implemented by William Hawes. It has become so popular with both users and developers that Commodore has included it as part of version 2.0 of the Amiga's operating system. (Users of earlier operating-system versions can purchase 2.0 from an Amiga dealer.)

Next to the Mode gadget is another multistate gadget that toggles between the Workbench Screen and Custom Screen options. This is used to tell AmigaVision what kind of display your external application should operate in. If the program normally opens on the Workbench, set the display to Workbench. If, on the other hand, it uses its own special screen, Custom is the proper setting. If the program does not use any screen of its own, set this gadget to Custom to keep the AmigaVision display visible.

Below the multistate gadgets are two command buttons that become available only when you are using ARexx mode. Labeled Result and RetCode (Return Code), these options allow your AmigaVision program to communicate with the ARexx applications you are using.

AmigaVision sets the RetCode upon completion of the ARexx script, and also indicates how successful the script was — whether it worked or not. Here are some examples of possible ARexx return codes:

0	Command executed successfully
5	Command is recognized but not performed
10	Command was not recognized
15	AmigaVision is not processing commands
20	AmigaVision is closing

The Result command allows AmigaVision to read any values being sent to it from an ARexx application or script.

Clicking on either button brings up a Specify Variable requester wherein you can link an existing variable to contain the information the field returns. You can use this value in your programs to make decisions based upon these results and upon information provided by ARexx applications.

Figure 11-13 is an example of an AmigaVision program that is used to launch a number of different programs based on user input.

Here, the user of your application is presented with a menu interface (see Figure 11-14) that displays a number of options. When he or she selects one, the appropriate application starts. When that application finishes, the user is returned to the "master control" program. Programs like this can be used to set up a shell that offers users limited access to the computer. An appropriate application for such a setup might be a school computer lab where you want to limit students to using only certain programs. If the computer is started with the AmigaVision shell application running, the students' choices are then limited to programs that the instructor deems important.

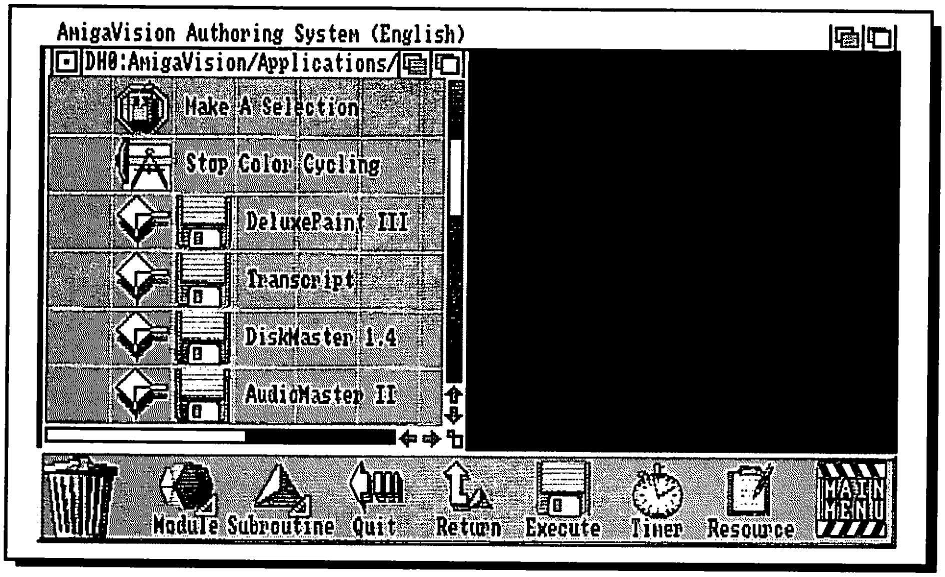


Figure 11-13
Using the Execute icon

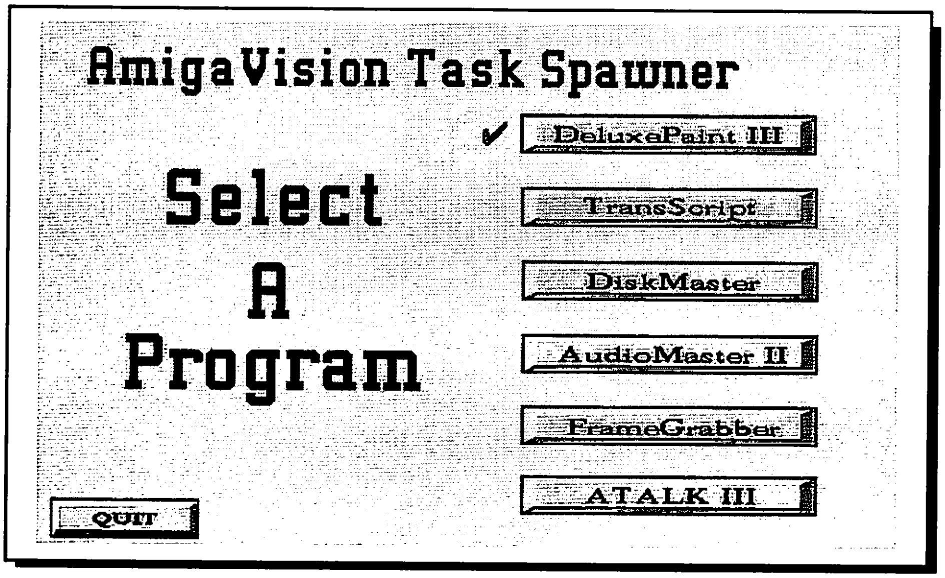


Figure 11-14
A sample AmigaVision "shell application" interface

Section Three Editor's Tools and Programming Techniques

– 12 – The Videodisc Controller

One of the strengths of AmigaVision, especially when compared to other contemporary Amiga authoring languages, lies in the ease with which the programmer (and, by extension, the application's user), can incorporate full-motion video in AmigaVision programs.

We have already discussed the features of the Video icon and to some degree, the Videodisc Controller (see Chapter 10). In this chapter, we will concentrate principally on the Controller, although the Video icon will be mentioned.

To make use of AmigaVision's video features, you must connect a serial-port-equipped laser-disc player to the Amiga's serial port via a special cable. (If you are using a Sony, you need a null-modem cable, if you are using Pioneer, you need a custom cable.) I also recommend having a genlock (video-overlay device) so that you can mix the computer's RGB graphic signals with the laser-disc player's composite video. If you do not have a genlock, you must use a second composite monitor or a television set to display video sequences that you select using AmigaVision on your computer monitor. The sequences shown on a second display will be video images only, because without the genlock, no overlay of computer signals is possible.

While there are many laser-disc players on the market, the ones that AmigaVision supports directly are just a small subset of the whole. At the time this book is being written, AmigaVision 1.53G supports 11 different industrial (serial-port equipped) laser-disc players. If you examine the SYS:devs/players directory of the AmigaVision boot disk, you will find the following list of supported systems:

Philips_405_1200	Sony_1200_9600
Philips_410_9600	Sony_1500_1200

Philips_835_1200	Sony_1500_9600
Philips_835_9600	Sony_1550_1200
Pioneer_2200_4800	Sony_1550_9600
Pioneer_4200_4800	Sony_2000_9600
Pioneer_6000_9600	Sony_Umatic9_9600
Pioneer_6010_9600	

You may notice that some of the laser-disc drivers are listed here twice — once for each of the baud rates they support. In general, where there are two choices, you should select the fastest baud rate your player supports.

The last device in the list above — the Sony_Umatic9_9600 — is actually not a laser-disc player at all. Instead, it is a fairly expensive frame-accurate industrial videotape deck that comes equipped with the serial interface that AmigaVision requires. I have a strong suspicion that as AmigaVision becomes more widespread in use, Commodore and third-party companies will create AmigaVision drivers for other tape systems as well.

AmigaVision's laser-disc support is geared to one specific disc format, called Standard, or CAV (Constant Angular Velocity). This is not the most common type of disc. Most commercial discs are created in CLV (Constant Linear Velocity), or Extended, format. CAV disks can be referenced by specific frames on the disc, while CLV discs are accessed by time, which is generally much less accurate. Unless your player has a number of digital features, you will find that CLV format is not well suited to multimedia programs. For example, a CLV disc cannot freeze a single frame of video as can a CAV disc. Nevertheless, you can play CLV discs on your AmigaVision- supported player and use them in applications.

You can access the software Videodisc Controller via the Video icon's Controller button, the Object editor's Project menu, or the Tools menu at the top of the AmigaVision editing screen (see Figure 12-1). When you call it from the Tools menu, you can use the Controller to search a video disc for sequences to use in your programs. You can also use it as a general laser-disc controller for viewing the contents of a disc. (I often watch laser-disc-based commercial movies on my Amiga, using the on-screen laser-disc controller as I would use a hardware remote-control unit.)

There are four distinct parts to the Controller requester. In the upper left is a series of buttons labeled Still, Play, Step, Scan, Slow, and Fast. Think of these as your remote-control buttons.

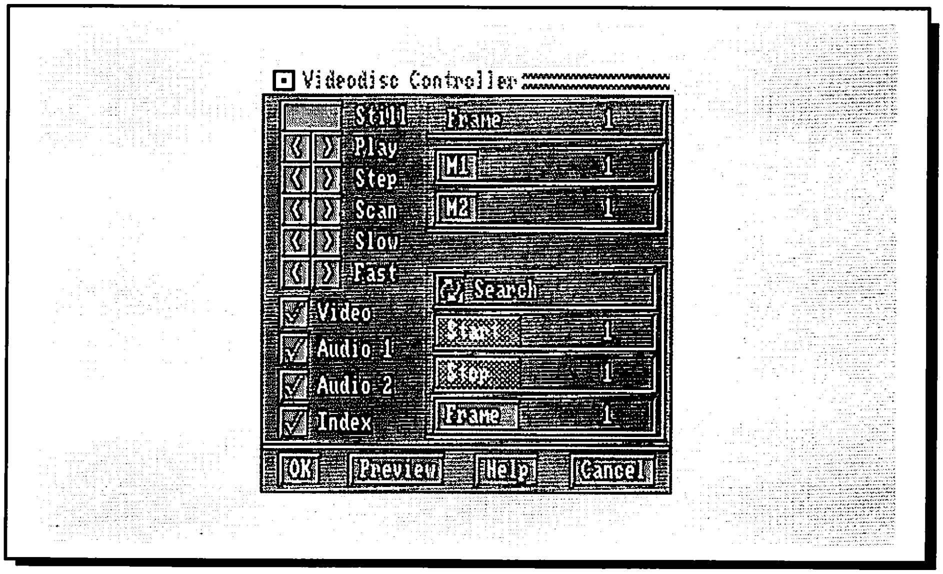


Figure 12-1
The Videodisc Controller requester is divided into four distinct parts

The Still button is used to pause, or freeze, the image on screen. When selected, it stops the laser disc at the current frame but does not wipe the video image from the screen. The result is a high-quality still picture. You can study this display for details that you may not see while the video is running. Once you stop the video motion with the Still button, you must use one of the other motion buttons to start it playing again or to otherwise change the frame.

Each of the remaining commands on the Videodisc Controller is associated with a pair of buttons. Both buttons in each pair are labeled with arrows pointing either left or right. In all cases, these arrows correspond to the direction of the action.

The right-arrow button of the Play command plays the disc forward at normal speed, while the left Play button plays the disc backward at normal speed. If you want to watch the video as you would normally watch a movie, select the Play forward gadget.

The Step buttons allow you to step through the video either forward or backward a single frame at a time. If a video disc is playing when you click on one of the Step buttons, motion freezes just as it would if you had clicked on the Still button.

The Scan gadgets can move you forward or backward through the video at a rapid pace. With Scan you can jump hundreds of frames in a couple of seconds.

The Slow command does just what its name implies: it plays the video disc slowly in either forward or reverse. Use Slow if you want to examine a sequence of frames closely but do not want to step through them manually.

Fast allows you to play the video very quickly, but more slowly than the Scan option.

All sound is disabled when you use any of these commands except for Play forward. The Play forward command generates the proper audio unless you disable one or both of the audio channels.

Below these controls is a second group of check-box buttons. These buttons — labeled Video, Audio 1, Audio 2, and Index — allow you to turn on and off the video signal, the dual audio tracks, and the frame-index function. When any of these gadgets are checked, the corresponding function is enabled; when no check mark appears, the function is disabled.

The Video button allows you to shut off the incoming video signal. You can enable it when you do not want video to be visible.

The two audio-track controls, Audio 1 and Audio 2, can turn the left and right stereo channels on or off. These gadgets are very useful for working with educational CAV discs. Many such discs have sound effects (music and natural background sounds) on one track and narration on the other. These buttons allow you to selectively use either track, depending on the needs of your application.

The Index option is supported by most laser-disc players. This command allows you to use an on-screen frame counter. With this feature enabled, you can see where you are at a glance, and even record the information for later use.

At the upper right of the Controller requester is the word Frame. During your work with the Controller (even while playing or scanning video), the space to the right of Frame shows the number of the frame currently on display. This read-out should not be confused with the Index button: Index displays a frame counter on the screen, while Frame shows the frame number on the requester itself.

Below Frame are two buttons labeled M1 and M2. These are variables of a sort, and allow you to store frame numbers for later use by the Start, Stop, and Frame buttons. To store a frame number in either of the two variable positions, just click on one of the buttons. The number corresponding to the frame currently on display is then stored in the appropriate location. The current values of these variables are always displayed to the right of the buttons.

In the lower right of the controller is a multistate gadget and three command buttons. All of these buttons are not always available; they may be ghosted, depending on the status of the multistate gadget. The gadget toggles between Play, Search, AutoStop, and Unknown modes. The buttons are labeled Start, Stop, and Frame.

Play mode allows you to define a sequence of frames and then to play just that sequence. To use it, click on the multistate gadget until it reads Play. In Play mode, the Start and Stop buttons become available for definition (the third button, Frame, is now ghosted in this mode). To use the Play option, you must define starting and ending frame numbers. To specify a starting frame, click on the Start button. A Frame Number requester, which has four options — Enter Frame, Current Frame, Memory 1, and Memory 2 — appears (see Figure 12-2).

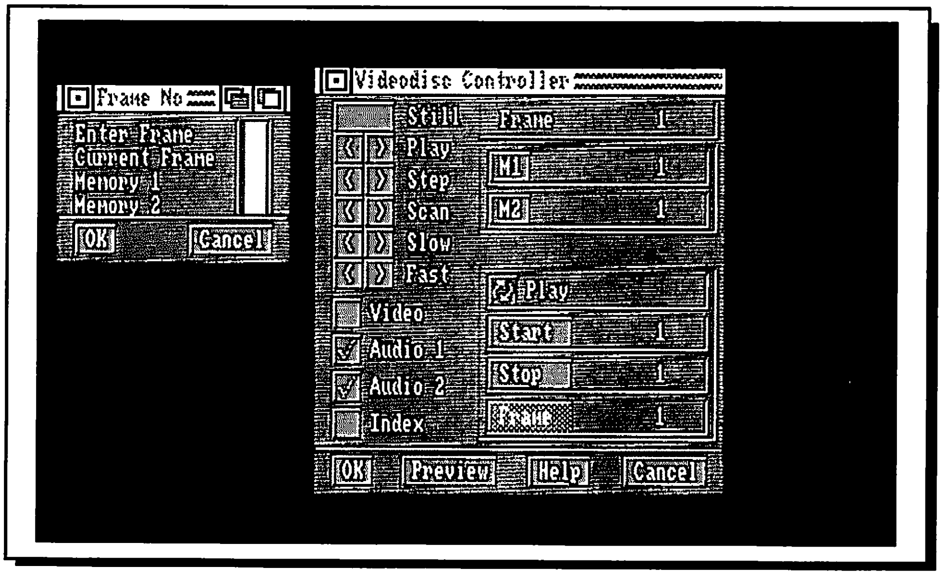


Figure 12-2
The Frame Number requester lets you define starting and ending frame numbers

Selecting Enter Frame brings up a Specify Value requester (see Figure 12-3), wherein you can enter a number or variable to indicate the desired frame. If the screen is already displaying the frame you want, just click on Current Frame, and the number of the current frame will be entered. Or, if you have entered any values into M1 or M2 of the Controller requester, you can enter those by just selecting the appropriate option. To define the ending frame for the sequence, just click on the Stop button and repeat the process.

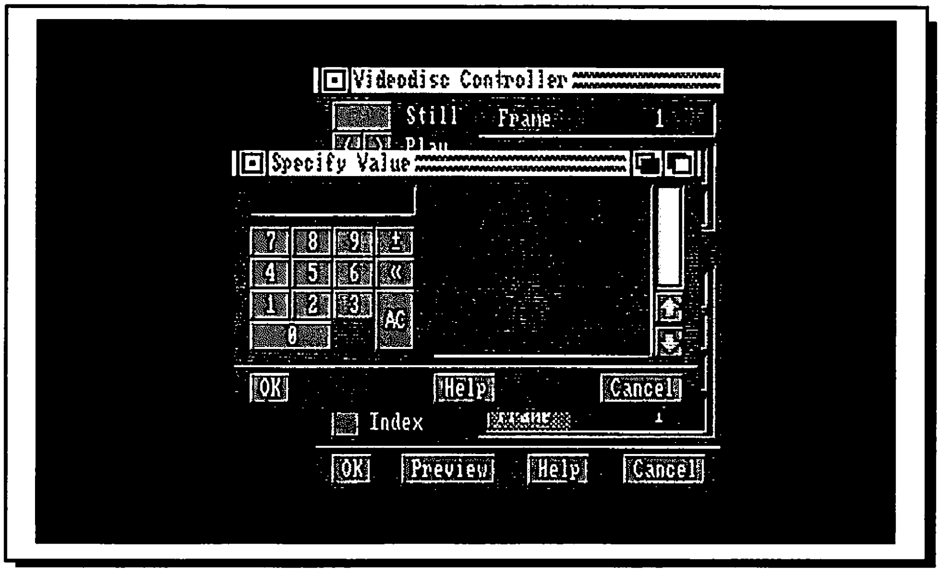


Figure 12-3
Using the Specify Value requester to select a video frame

To locate a specific frame, click on the Controller requester's multistate gadget until it changes to Search. Once in Search mode, you will notice that the Start and Stop buttons appear ghosted, or disabled, but that the button labeled Frame is now available. (Because the Search function is used to find just one frame, only one value is needed.) Click on Frame and the Frame Number requester appears. Here, you can choose from the current frame or one of the memory variables (M1 or M2), or you can enter a new frame via the Specify Value requester.

Another mode available from the multistate mode gadget is AutoStop. It lets you specify a frame for cessation of the video action. In other words, it allows you to specify where a Stop command is to be issued. This is useful when there is an upper limit on how far to play a disc. For example, suppose you want users of your application to have the ability to step through or play only the first 25,000 frames of a disc. If you set AutoStop to 25,000 the video will stop at that frame automatically. AutoStop is defined in exactly the same manner as Search: You specify a frame number by clicking on the button and setting the frame number. (Like the Search command, AutoStop needs only one parameter.)

The final option is Unknown. When you access the Videodisc Controller from within the Video icon via a command other than Play, Search, or AutoStop, the

Controller's Mode gadget is set to Unknown. This means that these commands do not need any parameters.

Once you have defined your commands, you can preview their actions by clicking on the Preview button at the bottom of the Controller. Of course, only commands such as Play and Search will have any visible effect.

If you entered the Videodisc Controller through the Controller button on the Video icon, the settings you specify within the Controller are used to define the Video icon once you exit the requester. For example, if Video and Audio 1 are enabled (but Audio 2 and Index are turned off), and if you have set the mode to Play, and have specified frames 25,000 - 26,500 upon exiting the requester, the Video icon will be set to Play mode with the gadgets and parameters defined to the selected frames, just as you had set them in the Controller.

A trick of the Controller is its ability to change size or disappear entirely. If you position the mouse pointer over the Controller requester and press the right mouse button, the requester will shrink, leaving only the remote-control buttons visible (see Figure 12-4). The smaller controller size makes it easier to watch the screen without missing any of the image. To return the Controller to its full size, place the pointer over the requester and again press the right mouse button.

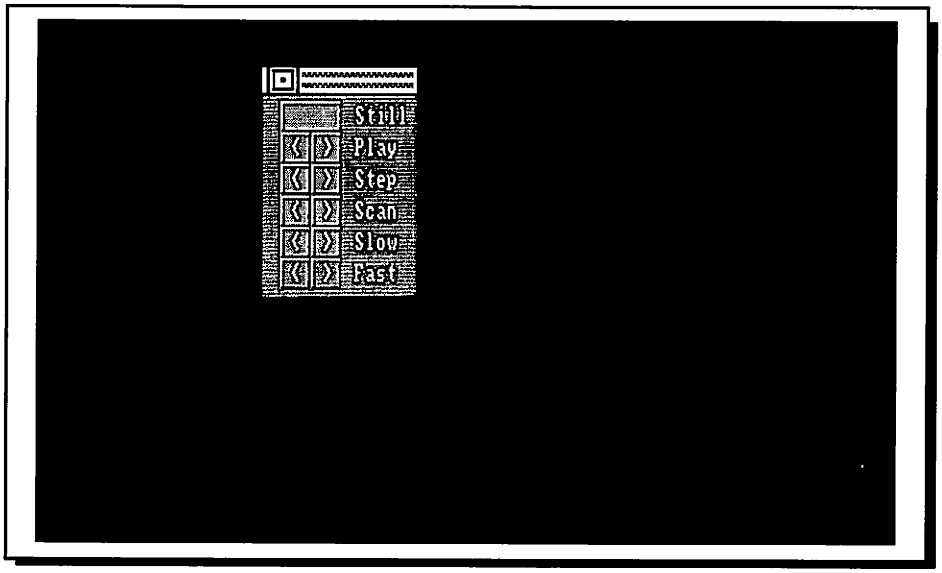


Figure 12-4
The shrunken Videodisc Controller lets you view the screen without missing any of the image

If you press the right mouse button while the pointer is not positioned over the Controller, the requester image will disappear completely. To bring it back into view, just click again with the right mouse button while the pointer is positioned outside the requester window. Using the mouse to hide or display the requester works regardless of whether the controller window is large or small at the time.

– 13 –
The Expression Editor

A computer language must be able to use and manipulate data in the form of variables and expressions. While writing the program, you can create conditions that allow the application to manipulate information, without even knowing the value of that information. In AmigaVision, we use the Expression editor to create, define, and evaluate variables and expressions.

AmigaVision supports four basic data types, all of which can be used as variables. The data types supported are String (characters and words), Integers (whole numbers), Floating Point numbers (numbers with decimal points), and Boolean values (logical values).

Strings are usually enclosed in quote marks, and can be from one to 255 characters long. Here are some examples of strings:

"Lisa"
"Press a key when ready."
"The answer is 1000."
"0123456789"
"$&*()#@=-_+\|?>,<.';:}[{]!`~"

Integer numbers are those without any decimal places — that is, whole numbers. They can be either positive or negative values. Some examples of integers are:

32768
256
1
- 1
100000000
- 35

Floating-point numbers do include decimal points. AmigaVision supports up to nine decimal places for this data type. Like integer numbers, floating-point values can be either positive or negative. Here are some examples:

100.00
0.001
3.14159
- 11.11
- 32768.0

Boolean values are logical states. They have only two possible conditions — true or false. You can also think of these conditions as being either yes or no, or 0 or 1. Two examples of Boolean values are:

Answer = True
Result = False

Variables created in AmigaVision take on the characteristics of the data they define. Some languages (such as BASIC) require you to use certain characters in the variable names to indicate the type of data within. For example, a language might require that you use the modular (%) symbol for integer variables, the dollar ($) symbol for string variables, and so on. This is not the case in AmigaVision, however. If you equate a variable name to a particular type of data, the variable takes on that form. For instance, if you create a variable with the assignment x = "Amiga" the variable *x* becomes a string variable. Likewise, defining it as x = 1.01 creates a floating-point variable, and x = 1 makes it an integer variable.

Once you define a variable, its name will appear in the Expression editor's Variable List window accompanied on the right by a single character that identifies the data type of that variable's definition. The data-type identifiers are I (integer), F (float), S (string), and B (Boolean).

You can access the Expression editor (see Figure 13-1) from within a number of icons in AmigaVision, including the Module, Subroutine, Variable, Conditional, and Loop icons. In addition, the Object editor's Input Field requester allows Expression editor access.

As you can see from Figure 13-1, the Expression editor is a large and impressive requester. It has an input field into which you enter expressions via keyboard or mouse. It also offers two specialized keypads containing numbers and operators, two different list windows, and six buttons for adjusting the content of the input field.

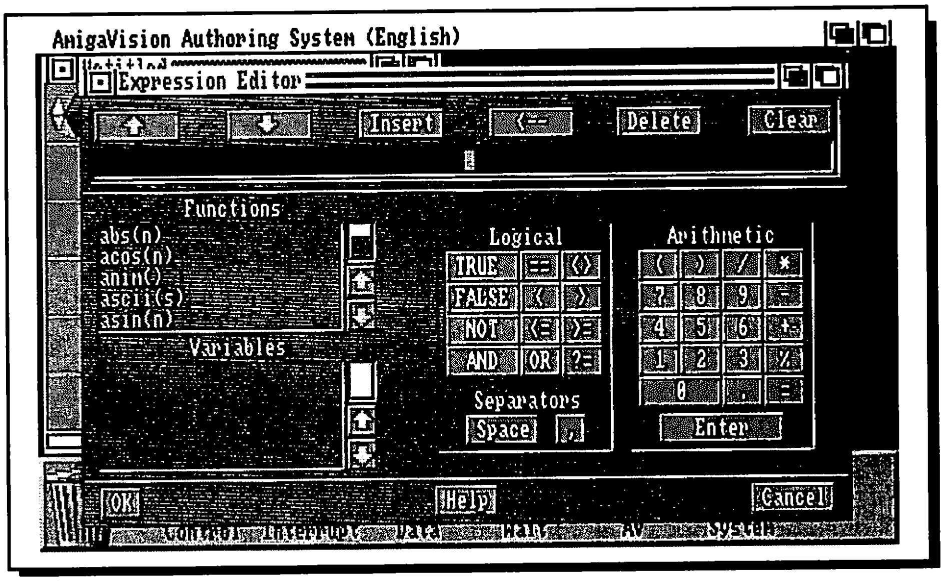


Figure 13-1
The Expression editor can be accessed from a number of icons

One of the keypads is a normal numeric keypad with the usual five mathematical operators. The other is dedicated to Boolean-style logical and conditional commands. One list window displays all the variables currently defined at the point in the program where you entered the Expression editor. The other window contains all the various functions you can use in AmigaVision (there is a total of 40 functions in AmigaVision 1.53G). At the top of the Expression editor is the input field, into which you enter your expressions. This field has six associated command gadgets.

There are two basic types of expressions that you can use in a program. The first is the Assignment expression. You can use it to either create a new variable or alter the value of an existing one. You can define a variable (assign it a value) using a constant (a number), one or more other variables, the supplied functions, or any combination of these. For example, the following are all assignment expressions:

X = 27
I = 100
V = 99.95
J = V + 1

```
Lastname = "Wallace"
Z = (X * I) / V
Y = X / Z
Remainder = Y % 2
K = SIN(X)
Date = Date()
Truth = True
```

All these expressions share a common ingredient, that is, the equal sign (=). In all these cases, the equal sign serves as an assignment operator, wherein some variable on the left side of the equation (to the left of the equal sign) is defined (assigned a value) in terms of what is on the right side of the equation. The value on the right can be either a constant (such as 27, 100, or 99.95) or another expression (X / Z, for example).

The other form of expression is the Conditional expression. This form is used to evaluate a variable or expression. Based on the results of that expression, AmigaVision can then take appropriate action in the program. Conditional expressions are usually created for use in such commands as the If-Then, If-Then-Else, Conditional Goto, Wait Condition, Loop icons, or the Input Field requester. Although you do not actually type If, Then, or an action command, in all cases you are asking AmigaVision to take some action if the defined condition evaluates as true. The action command is implied by the command itself. Here are some examples of conditional expressions:

```
X == 27
Y == 99.95
Z >= 100.0
Lastname == "Wallace"
Married == FALSE
X < Y
X AND Y
Firstname <> "James"
Sum > (X + Y + Z)
SIN(X) <= 3.14159
```

Notice the presence of the double equal sign (==) in many of these sample expressions. This differs from the assignment expression where the variable on the left is assigned the value on the right by using a single equal sign (=). In Conditional expressions, no assignment of value is being performed. Instead, Conditional expressions compare the values of each side of the expression. Depending on which logical operator you use, the result of the comparison is either true or false. It is this result that the If-Then, If-Then-Else, Conditional Goto, and other conditional icons act upon. One of the most common errors you will make when programming is to inadvertently use a single equal sign (=) when you mean to use the double equal sign (==), and vice versa.

There are several different logical operators available in the AmigaVision Expression editor. Each evaluates an expression in a different way. Which you use will depend greatly on the circumstances of your program. Below is the list of logical operators and their definitions. Following each is an example and a breakdown of how AmigaVision interprets that expression.

== equal to

A == B: Is the value of A equal to the value of B?

<> not equal to

A <> B: Is the value of A not equal to the value of B?

<= less than or equal to

A <= B: Is the value of A less than or equal to the value of B?

>= greater than or equal to

A >= B: Is the value of A greater or equal to the value of B?

< less than

A < B: Is the value of A less than the value of B?

> greater than

A > B: Is the value of A greater than the value of B?

In addition, AmigaVision allows you to use five Boolean operators in your decision-making expressions:

NOT	- logical negation
AND	- logical conjunctions
OR	- logical disjunction
TRUE	- Boolean reserved value
FALSE	- Boolean reserved value

Functions

Besides the usual mathematical operators plus (+), minus (-), multiply (*), divide (/), and modulus (%), AmigaVision has a large number of additional functions. These are found in the Functions window of the Expression editor. You can use them in your program by typing them in from the keyboard or by clicking on the function name in the list window with your mouse.

There are three classes of functions available: Mathematical, String, (character oriented), and a miscellaneous class called Special. The three types are mixed together for presentation in the function list window in alphabetical order. They are not separated into groups based on the type of action they perform. In describing them here, however, I have grouped the functions by type. Following, then, are the functions and their definitions:

Mathematical Functions

Abs(x)	This function returns the absolute value of the constant or variable enclosed in the parenthesis. The absolute value is always positive, so even if *x* is negative, the resulting value is positive *x*. For example, abs(-1.12) equals +1.12, and abs(+1.12) also equals +1.12.
Acos(x)	The arccosine trigonometric function is the inverse of the cosine function. For example, if x=30 and y=cos(x), then acos(y)=x.
Asin(x)	This arcsine trigonometric function is the inverse of the sine function.
Atan(x)	The arctangent trigonometric function is the inverse of the tangent function.
Cos(x)	This trigonometric function generates the cosine of an angle.
Exp(x)	Returns the value of e^x, where you supply the value of *x* as a constant or variable.
Inc(x)	The increment function increases the value of the supplied variable by 1.
Integer(x)	This function reports the integer of the supplied value. For example, Integer(12.12) equals 12 (with no decimal places). If your program is generating or working with numbers in the

	form of string variables, this function will still work: Integer ("12.12") also equals 12. If the string is a name rather than a number, the returned value is 0. Integer("Louis") equals 0.
Float(x)	Related to the Integer(x) function, Float(x) returns the floating-point value of either numbers or strings. Like Integer(x), if the supplied string is not a number, the returned value is 0.
Log(x)	This mathematical function delivers the natural log of the supplied number or variable.
Log10(x)	Returns the base 10 log of the supplied number or variable.
Max(x1,x2)	The result of this function is the larger of the two supplied numbers or variables. For example, if x1=2 and x2=10, then Max(x1,x2) equals 10.
Min(x1,x2)	This is the inverse of the Max function. It reports the smaller of the two supplied numbers.
Pow(x,y)	The power function raises *x* to the power of *y*, that is, x^y. For example, if x=2 and y=4, then Pow(x,y) is equal to 2^4, or 16.
Sin(x)	This trigonometric function returns the sine of an angle.
Sqrt(x)	Delivers the square root of the supplied number or variable.
Tan(x)	This trigonometric function gives you the tangent of an angle.

String Functions

Findstr(s1,s2,n)	This function returns a number that indicates where in string2 it found string1. The search begins at the *n*th character in string2. If the string is not found, a value of 0 is returned.
Strcat(s1,s2)	This function concatenates, or affixes, string2 to the end of string1. For example, if s1="Multi" and s2="media", then the result of Strcat(s1,s2) is "Multimedia."
Strcmp(s1,s2)	A function found only in the latest version (1.53G) of Amiga-Vision, this compares two strings and returns T (true) if they are the same and F (false) if they are not. This function is case sensitive, so "Sharon" is not the same as "sharon."

String(x)	Converts the number *x* (integer or floating point) into a string.
Strlen(s)	This reports the length of a string.
Substr(s,x1,x2)	Returns a string consisting of x2 characters beginning at character x1 in the string *s*. For example, if s="AmigaVision", x1=6, and x2=3, then Substr(s,x1,x2) will return "Vis."
Tolower(s)	This function changes all the characters in the string *s* to lowercase. Keep in mind that the change alters the source string and does not merely return a new string. You can use this function without an equal (=) sign, i.e., tolower(s).
Toupper(s)	This converts all characters in the string *s* to uppercase. Like tolower(s), toupper(s) alters the source string and does not merely return a new string.

Special Functions

Anim()	The Anim() function reports the number of the frame of animation currently on display. It is very useful for syncing sound or other action to specific frames of an animation.
Ascii(s)	Returns an integer value that is the ASCII value of the character string *s* (*s* should be a single character). For example, if s=A, ascii(s) will return the integer value 65.
Boolean(v)	This function generates a T (true) if the value of the variable *v* is T, True, Yes, or any number other than 0.
Char(x)	The result of this function is a string consisting of a single character that corresponds to the ASCII character represented by the integer *x*. For example, if x=66, char(x) will return B.
Clock()	This function returns a string containing the current time in the format HH:MM:SS. The hours are in 24-hour format, i.e., 18:30:00.
Date()	Delivers a string containing the current date in the format MM/DD/YYYY, i.e., 12/01/1990.
Mouse()	Mouse() is a function found only in the latest version (1.53G) of AmigaVision. Its resulting response string indicates which mouse button was last clicked. LMB indicates the left mouse

	button, and RMB the right mouse button. Note that clicking the right mouse button in an editable AmigaVision application quits the application and returns you to the editor, so this function is useful only in an application that was saved with the Edit Protect option during the creation of distribution disks.
Random(x1,x2)	This is a random number generator. It returns an integer value in the range defined by x1 and x2, where x1 is the lowest and x2 is the highest acceptable values.
Response()	One of the most important functions in AmigaVision, the Response() function returns a value defined in a hot spot created with the Object editor. The actual response is a string that was defined in the object's response field.
Screenx()	Screenx() gives you the X (horizontal) position of the pointer. The value is an integer that represents the pixel position on the screen. By convention, 0 is the leftmost screen location. The upper range is 319 for low-resolution displays, and 639 for high-resolution displays. These figures are slightly higher for overscanned displays. Note that this function reports the position of the pointer when the *left* mouse button was clicked last.
Screeny()	This function returns the Y (vertical) position of the pointer. By convention, 0 is the top of the screen. The maximum values are 199 for noninterlaced displays and 399 for interlaced displays. If you are using overscan they can be somewhat bigger. Note that this function reports the position of the pointer when the *left* mouse button was clicked last.
Status()	Generates a T (true) or an F (false), depending on the results of the last system request.
Timer(x)	This returns the value of one of the nine timers available in AmigaVision. The results are in seconds, reported as a floating point number.
Video()	Like the Anim() function, Video() returns the number of the frame of video currently playing on a video-disc player. By using this function to read the frame of video currently in use, you can have the program make specific actions at predefined frames.

Version()	This function is found only in version 1.53G of AmigaVision. It tells you the number of the AmigaVision version being used to run the application.

Using the supplied functions along with common mathematical operators, you can create more complex mathematical functions than those supplied with Amiga-Vision. For example, suppose you need a function to find the hyperbolic secant — sech(x). Because that function is rather obscure, it is not built in to AmigaVision. You can, however, generate it using the following expression

```
sech(x) = 2/(exp(x)+exp(-x))
```

Once you define this function as an expression in a Variable icon, you can put it in a subroutine and call it up whenever you need to use a hyperbolic secant. You can also build many other mathematical functions this way.

You can build new functions for manipulating strings in the same manner. As an example, perhaps your program needs a function that changes source strings to exactly ten characters in length: If a string is longer than ten characters, the function must chop off the rest, and if it is shorter than ten characters, the function must add trailing spaces until it reaches the length of ten characters. While there is no supplied function capable of doing that, you can easily build one. Here is a simple set of expressions that would make a supplied string variable *s* exactly ten characters long:

```
spc = "          "
s = s + spc
s = substr(s,1,10)
```

You can add this function to a single Variable icon by just entering the expressions in the correct order. (The sequence is critical, as it represents the order in which the expressions are executed.) This Variable icon then becomes a new function, one that quite effectively creates strings ten characters long from the supplied source variable *s*. With a little more work, you could build a function that was even more flexible, able to create strings of any desired length.

These two examples of creating new functions demonstrates something more important than merely adding functions to AmigaVision. It demonstrates the power of programming. From simple elements you can create something very powerful. Make no mistake about this — AmigaVision is a programming language. To effectively use it, you must learn at least the basic elements of writing a program, and the best way to learn is by doing. Start with something simple, perhaps the creation of new functions such as the ones above. Just keep working on it, and before long you will realize you have become a programmer.

– 14 – The Database Editor

One of the most important components of AmigaVision is its database support. Because it comes complete with a database editor and a full complement of specialized commands for creating custom database applications, you can use AmigaVision for a great many record-keeping needs.

A database is a highly structured collection of data on disk. Each database file consists of individual records, and each record is broken down into fields of information. If you were to create a database to take the place of your address book, you would set up fields for last and first names, street address, city, state, zip code, and phone number. All the information related to one person makes up a record. If you have 100 names in your address book, you will have 100 records in your database. Each of the individual bits of information — the first and last names, addresses, phone numbers, and so on — constitute fields within each record.

AmigaVision allows you to define one or more fields as key fields. A key field is used as a basis for indexing, or sorting, the records. For example, in a name-and-address file, you might use the last-name field as a key so that you can easily keep all the records in alphabetical order by surname. If you wanted to index the records by zip codes, you could designate the zip-code field to be a key field. AmigaVision does not limit you to a single key field, however, you can select multiple fields as keys. Thus, if you chose the zip-code field as the first key and the last-name field as the second, you could sort the records first by zip code and then further sort records with a common zip code by name. If you reversed the order of the key fields, the file would sort first by name and then by zip code. This key-field index feature is both useful and powerful. It can make database access fast and efficient.

There are two components to AmigaVision's database-management system. The first part is the set of commands that allows you to write programs that can use and manipulate a database (these we discussed in Chapter 8). The second part is the Database editor itself. It is here that you can create database files for your programs to access. Keep in mind that the Database icon commands do not allow you to create new database files, but only to read from or write to existing files. The actual file creation is done either in the AmigaVision Database editor or through an external database-management program that creates files compatible with AmigaVision. (AmigaVision's database file format is that of dBASE III, a program by Ashton-Tate designed for IBM PCs and clones. Several Amiga database programs, including Superbase Professional by Progressive Peripherals & Software and dbMAN by VersaSoft can generate compatible database files.)

The single most important consideration in the creation of any database file is planning. Although Superbase and dbMAN allow you to modify the structure of an existing database, once you have created a file within AmigaVision, you cannot alter its structure. Thus, it is essential that you know well in advance of creating the file exactly what you will need to store in each record. For example, suppose you finished creating your address-book database, only to find that you had forgotten to add a field for the phone number. At that point, you could not go back and add a new field to accommodate telephone numbers. You would instead have to create an entirely new database file to include the required field, and then either rekey any information you had entered into the file or write a program to transfer information from the old database into the proper fields in the new one. As you can see, adequate forethought is both desirable and necessary for the creation of any database. It's best to think your project through and anticipate any fields you might need *before* you begin creating the file.

AmigaVision supports four types of data in its database fields. They are: string, numeric, data, and Boolean. Any information you want to store and access must be in one of these data forms. Each data type has certain field limitations that you cannot exceed. String fields can contain up to 254 characters. Numeric data fields are limited to a total of 15 digits; for floating-point numbers this 15-digit limit includes the decimal point as well as the decimal places (you can place up to nine digits to the right of the decimal point). Boolean fields can hold only one character, and date fields can contain ten characters.

You can have as many as 128 different fields in a record, and each record can have up to 4000 characters. The only practical limit to the number of records your database can contain is the amount of space available on your floppy or hard disks. Key fields have a very specific limit: the total number of characters in all the key fields in your database is limited to 100. Again, the secret to success is careful planning.

To create a database file in AmigaVision, you access the Database editor from the Tools menu at the top of your AmigaVision editor screen (see Figure 14-1). This will open the Database window, a requester that allows you to design a new database file.

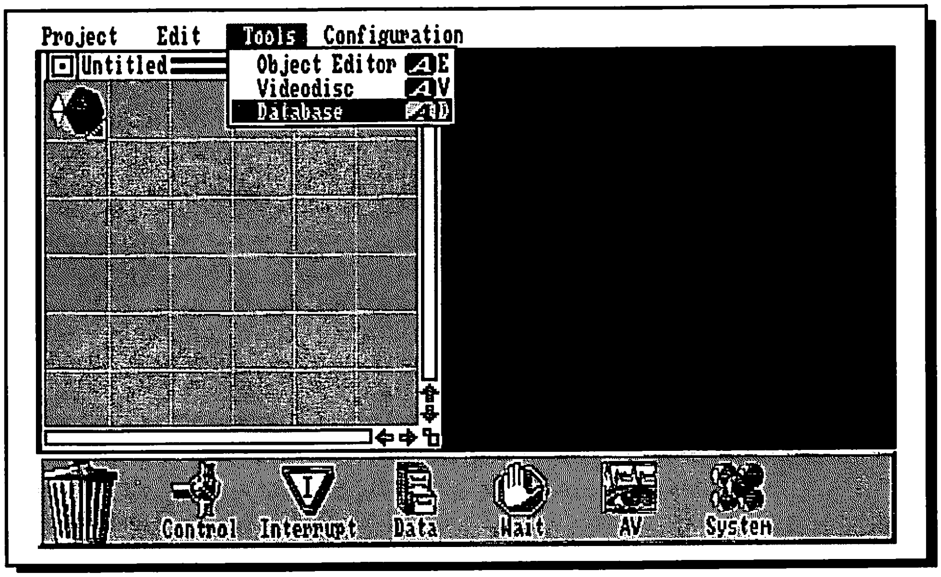


Figure 14-1
Selecting the Database editor from the Tools menu

The Database window (Figure 14-2) consists of 13 command buttons, one multistate gadget, and two input fields. At the top of the requester is an input field and a Directory command button. Using the Directory gadget, you can select a disk device from which to retrieve an existing database file or upon which to save a file you have just created. Below this is a set of four command buttons that act upon the entire database file. These operational-mode buttons are used to create a new database, delete an existing database, modify the contents of the records in an existing database, or change key-field designation.

The major portion of the requester is taken up by a field list window and buttons that allow you to create and manipulate the individual fields in the database. The list window shows all the various fields in each record, their sizes, and the type of data they contain. The four buttons associated with the list window (Delete, Insert, Move, and Clear) allow you to add, remove, and rearrange the fields in the field list window.

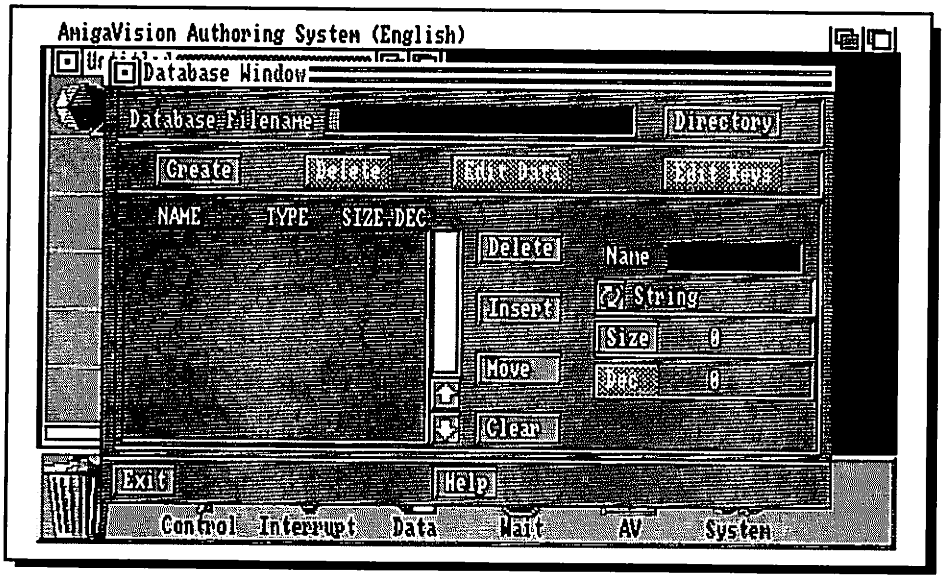


Figure 14-2
The Database Creation requester appears when the Database editor is accessed

To the right of the list window are two command buttons (labeled Size and Dec), one multistate gadget (which toggles between data types), and a single input field (Name), which you can use to name a field when you create it. Finally, at the bottom of the requester are the standard Exit and Help buttons.

When you first enter the Database editor, the Delete, Edit Data, and Edit Keys functions are not available for use. These functions are useful only when you are working with a defined database and are not helpful for building a file initially.

To create a new database, you must do several things. First, click on the Directory button to bring up the Directory requester. Select a disk device and a directory wherein you want the new database file to be stored. Then, in the input field labeled File, type the name you wish to give the new database. (Figure 14-3 shows a new database called Movies being assigned to the RAM disk.) Once you have selected the location and given the file a name, click on the OK button to return to the main Database requester.

Now, the Database requester's Filename input field will contain the full name and path where the file is to be created. The next step is to define the fields' names, sizes, and data types. At this point, you should have already given a great deal of thought to what kinds of fields are needed in your database.

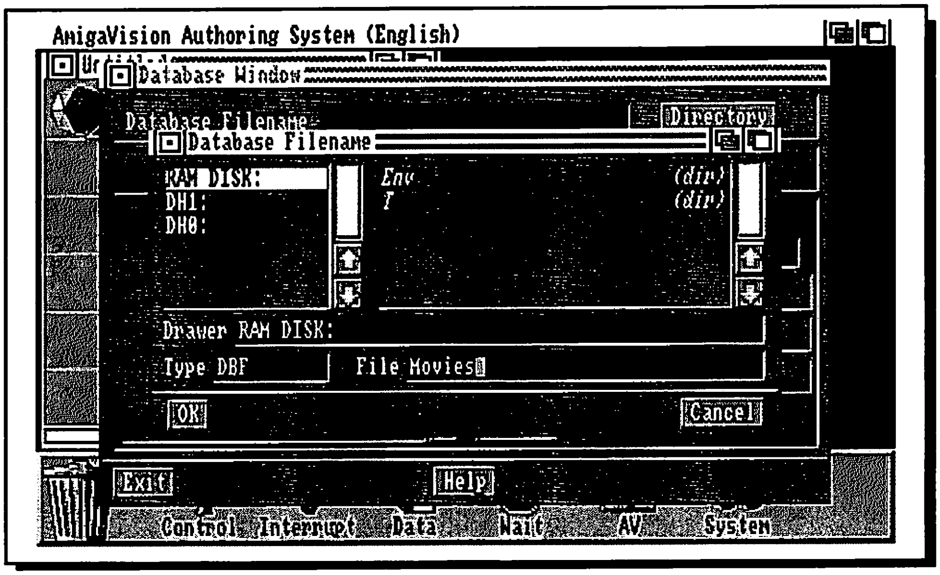


Figure 14-3
Naming a new database file

As an example, suppose you wanted to make a simple database for your collection of movies. You would want to list the title of each movie, its length, what format(s) you have it in, its genre (category), and perhaps include a comment field for a personal observation.

You would first type the name of the field you want to create into the Name input field. If your first field is to hold the movie titles, you will probably type the word Title into the Name field. (The AmigaVision Database editor limits field names to a maximum of ten characters.) Next, make sure the multistate gadget is set to String, so that you will be able to enter alphanumeric characters into the Title field. Then specify a size for the field, equal to the maximum number of characters that can be entered into it. In this case, a value of 72 is used. Finally, click on the Insert button to place the newly defined field into the list window (see Figure 14-4).

You can define the Genre and Comment fields as you did the Title field: as alphanumeric strings of 72 characters. (After you define each field, place it into the list window using the Insert button.) For the Length field, use a numeric field four characters long: one space to indicate the film's length in hours, one for the decimal place, and two for numbers indicating the minutes. Finally, you will need a string field of one character to indicate whether the movie is on videotape (T),

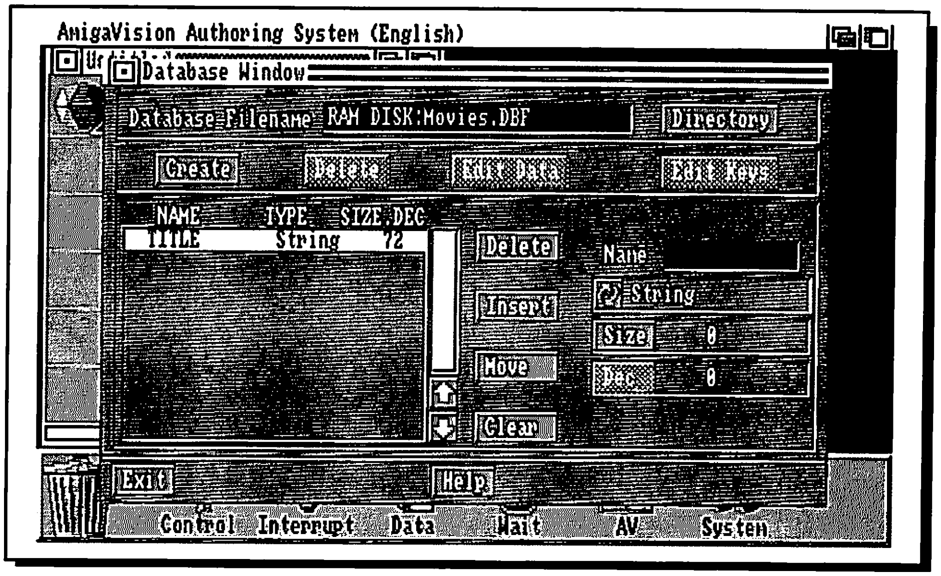


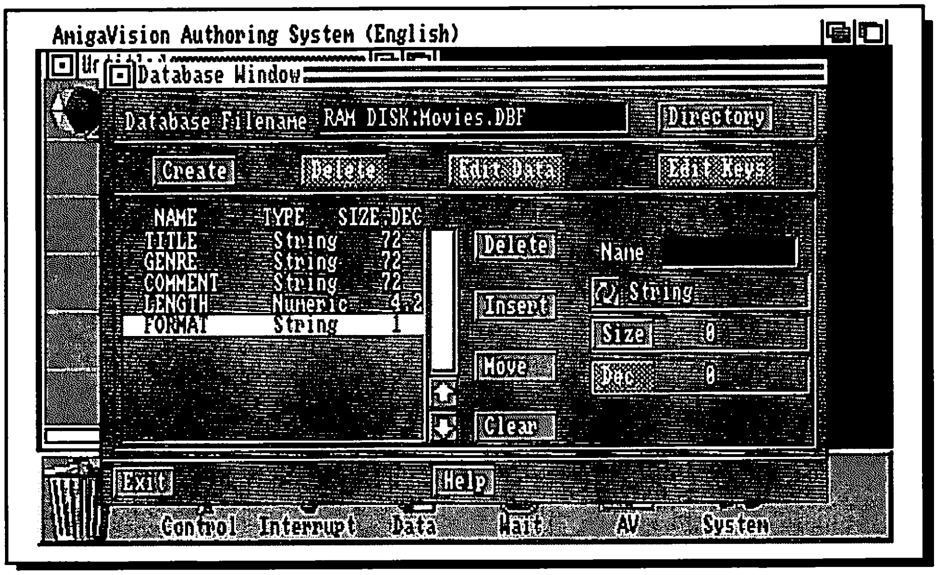
Figure 14-4
Inserting the field into the list window

laser disc (L), or both (B). At this point, all the fields you have defined appear in the list window (Figure 14-5). If you want to change their sequence, just click once in the list window on the field you want to relocate, and then once on the Move button to enter Move mode. Finally, click on the desired destination of your selected field in the list, and the field will be inserted at that position.

Once you have defined and ordered the fields in your database, you must then instruct AmigaVision to actually create the file. To do so, click on the command button labeled Create. This generates a database file containing the information you have specified: the number of fields, their types, and their sizes. (So far, there are no actual records in the database, and we have not defined any key fields with which to index the file.)

After you click on the Create button, there is a brief period of disk activity while the file is created, then the Database requester undergoes a change. The Create button, as well as the buttons associated with defining fields and arranging them in the list window, become indistinct, or ghosted, indicating that they are no longer available. However, the Delete, Edit Data, and Edit Keys buttons are now available.

The next stage in creating your database is defining which fields are to be used as key fields for sorting the file. For our example database, the Title field is probably the only key you will need.



Database Filename RAM DISK:Movies.DBF

NAME	TYPE	SIZE	DEC
TITLE	String	72	
GENRE	String	72	
COMMENT	String	72	
LENGTH	Numeric	4	2
FORMAT	String	1	

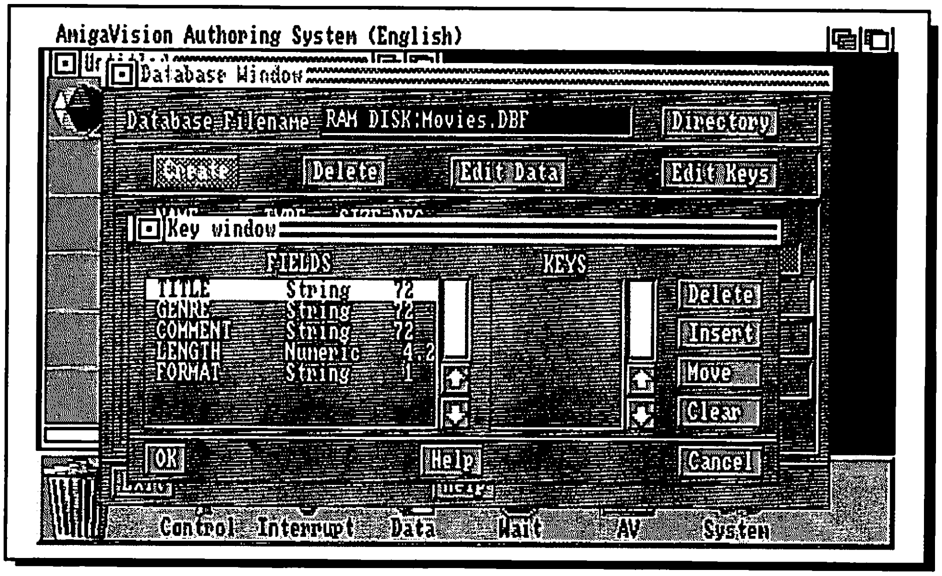
Size 0
Dec 0

Figure 14-5
The list window contains all the defined fields

When you click on the Edit Keys button, a new window, the Key window (Figure 14-6), appears. To designate a field as a key, click on its name in the Fields window, then on the Insert button to the right of the Key window. This places the name in the Key window, indicating that the corresponding field is to be used as a key. For this example, select the Title field and click on Insert; the word Title will appear in the Key window (see Figure 14-7). From this point on, all entries added to the database will be sorted (indexed) by the contents of the Title field.

If you need to designate two or more fields as keys, you can just continue to select them from the Fields window and insert them into the Key window. Keep in mind that when using multiple key fields, the way you order them is very important. The first field in the Key window becomes the main index, and subsequent keys become secondary sort criteria. In this example, if we had selected both Title and Genre as key fields but placed Genre first, AmigaVision would sort the records based on the contents of the Genre field before sorting each genre category by title. Remember to place the key fields in the proper order, using the Move button to rearrange them if necessary.

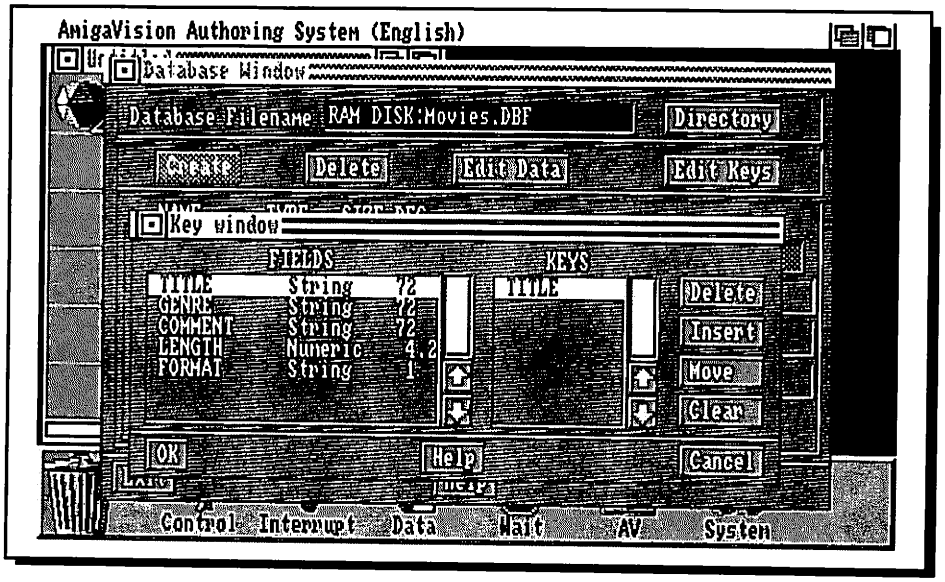
Although it is not possible to change the structure of a database once you have created it, you *can* redefine keys without damaging the contents of the database file. If at some point you change your mind as to what fields you wish to use as



Database Filename RAM DISK:Movies.DBF

```
TITLE      String    72
GENRE      String    72
COMMENT    String    72
LENGTH     Numeric  4.2
FORMAT     String     1
```

Figure 14-6
The Key window appears when the Edit Keys button is clicked



Database Filename RAM DISK:Movies.DBF

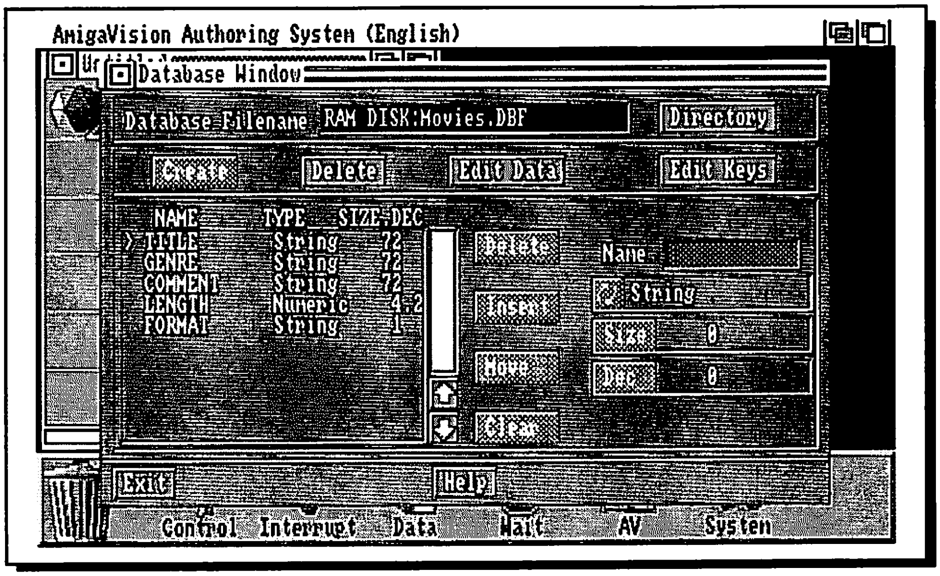
```
TITLE      String    72
GENRE      String    72
COMMENT    String    72
LENGTH     Numeric  4.2
FORMAT     String     1
```

KEYS: TITLE

Figure 14-7
Inserting field names in the Key window

keys and what order they should be in, you can come back and rekey the database file.

Once you have selected your key fields, click on the OK button to return to the main database requester. In the list window, you will see that each field you have selected as a key is marked with a greater than (>) character to the left of its name. In (Figure 14-8), you will see that the Title field is so marked, showing it is the designated key.



Database Filename: RAM DISK:Movies.DBF

NAME	TYPE	SIZE	DEC
> TITLE	String	72	
GENRE	String	72	
COMMENT	String	72	
LENGTH	Numeric	4	2
FORMAT	String	1	

Name:
String
Size 0
Dec 0

Figure 14-8
The key fields are labeled with the greater than (>) symbol

At this point, the database is ready to accept information in the form of records. There are two methods you can use for placing data into your file. The first (and best) is to write a program using the AmigaVision database commands as discussed in Chapter 8. It is entirely possible, however, to use AmigaVision as a database and never write a single program. By selecting the Edit Data button on the database requester you can enter data directly into the database you created.

Clicking on the Edit Data button opens the Edit Database window (Figure 14-9). This screen will contain all the fields you have defined in your database (unless they do not all fit on a single screen, in which case you can use the Next Page and Previous Page buttons at the bottom of the window to move through the entire database form).

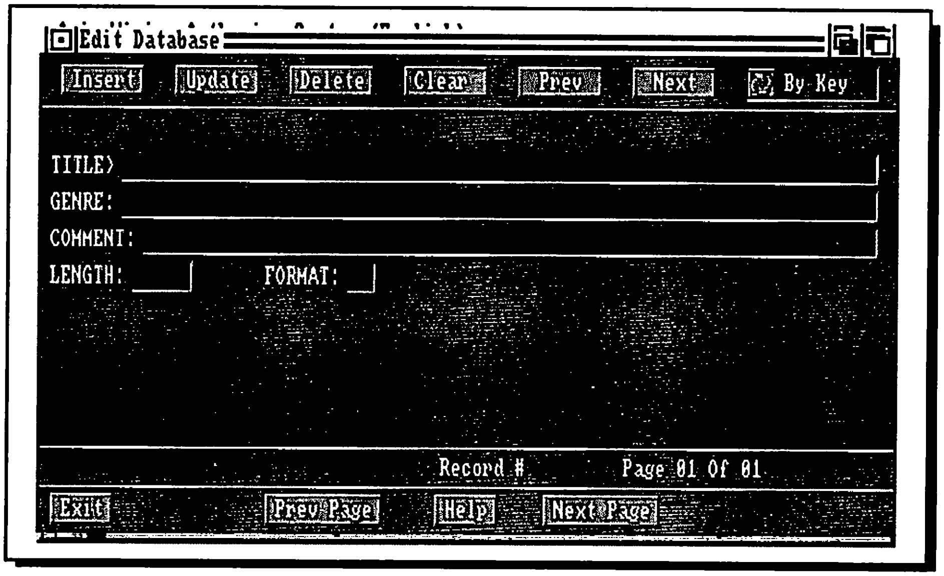


Figure 14-9
The Edit Database window appears when the Edit Data button is clicked

The Edit Database window has ten command buttons and one multistate gadget. The command buttons represent standard database-editing functions. For example, Insert places the contents of the fields into a new record. Update acts as a replace-record function, overwriting the contents of the record on disk with the contents of the fields on screen. Clear wipes all information from the fields, and Delete removes the record entirely. Prev and Next move you back and forth through the records in the database.

To add information via the Edit Database window, just enter it into each field. Then click on the Insert button at the top of the window, and the information will be added to the disk file. If you want to change the contents of an existing record, use the Prev or Next buttons to move through the database to the desired record. Once you have made the appropriate changes, click on Update to store the changes permanently in the database.

When working with the Edit Database window, you can use the multistate gadget to determine how the records are made available to you. The gadget has two modes, Record # and By Key. When working in By Key mode, clicking on the Next or Prev buttons moves you through the database in a sequential manner based on the key fields. In the case of our Movie database with its Title key, that sequence is in alphabetical order based on movie title. If you toggled the multi-state gadget to Record #, however, the Next and Prev buttons would move you through the records based on the order in which they were created — in other words, by the number that AmigaVision assigns to the records when you create them.

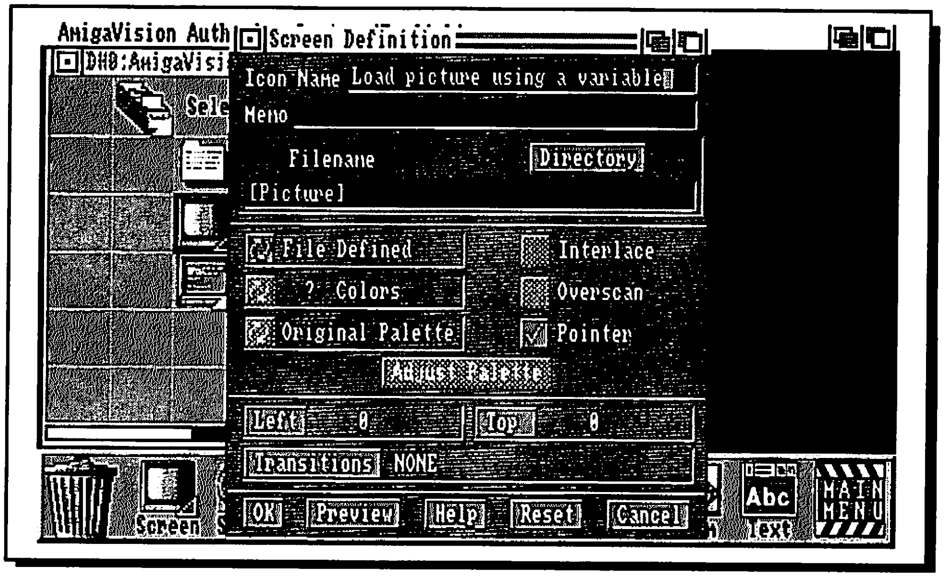
While you certainly can use the Edit Database window in AmigaVision as your primary means of entering data, this window offers a poor method of recalling information. It does not offer the ability to generate a report as the Output Icon does. Thus, you will want to write custom applications for most of your applications to allow yourself flexibility and ease of use.

While we have used just two examples — the mailing list and the movie catalog — in our discussions of databases here and in Chapter 8, there are almost unlimited applications for databases. For example, you can create database programs to maintain your check book, store your home or business inventories, maintain employee records, keep information on students and their grades, keep a customer list, handle recipes, maintain your stamp, coin, record, or CD collections, or even to organize your library of computer art and animations. All it takes to create these programs is a little imagination and a little time. Compared to traditional methods, AmigaVision makes program-creation an easy task!

A Path to Other Data Types

When I first began using AmigaVision, I was a bit disappointed to discover that for all its multimedia power, the Database editor offers only string, numeric, Boolean, and date data types — there are no provisions for pictures, animations, or sounds! However, I quickly discovered a simple trick for including such elements in a database. While you cannot keep pictures, sounds, or animations in fields, you *can* store their names, and the paths to locate them, in fields! With a file's name and path stored in a field, you can use the name as a variable to recall information. For example, suppose you have a picture called Betty Boop stored in your AmigaVision:ILBM directory. Suppose too that you have a field in your database called Picture. If, when creating a new record, you typed AmigaVision:ILBM/Betty Boop into the Picture field, you could read the contents of the Picture field into a variable using the Read/Write Record icon. This would allow you to display that picture during program execution by placing the variable's name in the Filename field of a Screen icon.

To use a variable as a file name, just enclose it in square brackets ([]) in the Filename field (Figure 14-10). This instructs AmigaVision to use the contents of the variable as the file name when it needs to execute the icon. The same technique can be used for sounds, animations, text files, or other information.



Icon Name Load picture using a variable
Memo
Filename Directory
[Picture]
✓ File Defined
Interlace
? Colors
Overscan
✓ Original Palette
✓ Pointer
Adjust Palette
Left 0
Top 0
Transitions NONE
OK Preview Help Reset Cancel

Figure 14-10
Using variables as file names

Keep in mind that this technique has two distinct limitations. First, you must define every file name and path explicitly in the database. (If you do not, AmigaVision will not be able to find the file.) Also, because variables are defined only while your application is running, when you create a disk for distribution, AmigaVision does not know what the variables will hold and therefore cannot move the called files onto the distribution disk. Keep these limitations in mind when writing your programs.

– 15 – The Object Editor

When you begin the task of creating an interactive application with AmigaVision, you must think beyond merely writing mouse- and keyboard-handling routines. The key to making useful AmigaVision programs is presenting information on the screen. It is here the user will be looking, it is here she or he will be clicking with the mouse, and it is here you must create your user interface. Central to the design and implementation of that interface is the Object editor.

The Object editor is the tool you will use to create your program's interface and its associated display features — graphic objects, text information, and input fields — all of which can be passive display elements or interactive hot spots. The Object editor is also the tool you will use to define the string responses your program will receive when the user clicks on hot spots on the interface. Finally, you will use the Object editor to define the audio and visual feedback that the user will receive when he or she clicks on those hot spots. Nearly every type of AmigaVision application will make use of the Object editor in some manner.

Many icons depend on feedback from the Object editor, and AmigaVision offers access to this essential editor in two ways. Some icons have a designated command button which, when clicked, whisks you into the Object editor. You can also enter the Object editor from the main edit work screen via the Tools menu (Figure 15-1).

When you enter the Object editor from the Tools menu, you are presented with a plain black screen. When you enter it from a command icon that is preceded in the program flow by a defined Screen icon, however, you have the opportunity to automatically load and display that screen while in the Object editor. This latter approach is quite useful if you plan on adding hot spots or other information fields to an existing screen image. (If you decide to display the previous screen image in the Object

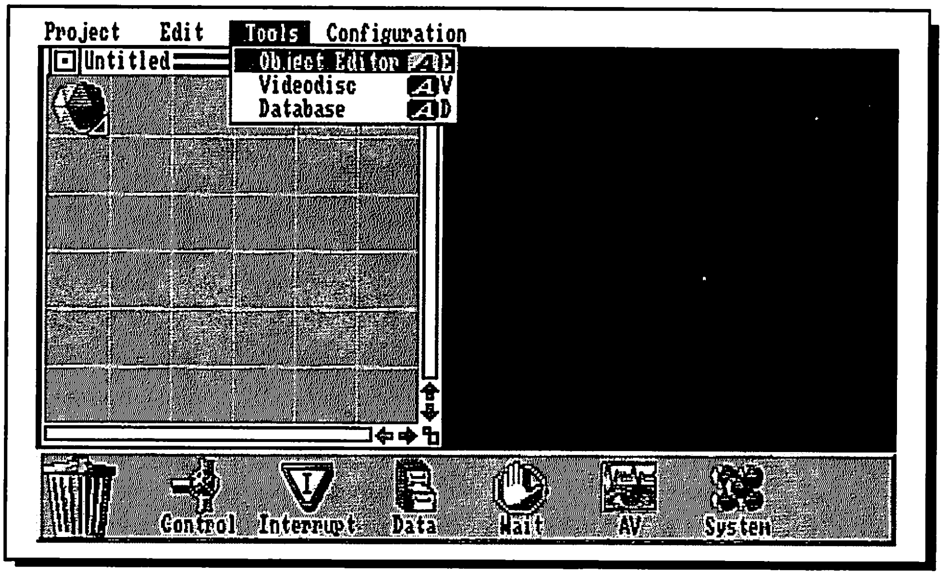


Figure 15-1
Object editor access from the Tools menu

editor, the screen will configure itself to match the characteristics, resolution and number of colors, of the loaded picture.)

The two entry methods are different in other ways as well. If you access the Object editor from an icon, what you do in the Object editor is associated with that icon. When you save the program, the information generated by the Object editor is saved and linked to the icon. If you enter the Object editor via the Tools menu of the main AmigaVision editing window, however, any display objects you create are not saved with a program, or associated or linked to any specific icon. Instead, you can save your creation as a file with a .dob suffix. This suffix indicates that it is a "display object" file, which you can load into the Object editor via multiple icons within a program. If your program has the need to reuse an interface display or if you want to use the design in multiple programs, creating the display in the Object editor accessed from the Tools menu is a great time-saving option. In fact, you can generate libraries of reusable interfaces this way.

When you enter the Object editor without displaying a predefined screen (or when you enter it from the Tools menu), the initial display is black and featureless. However, if you hold down the right mouse button, you will see that there are two menu headings at the top of the screen (Figure 15-2) labeled Project and Objects.

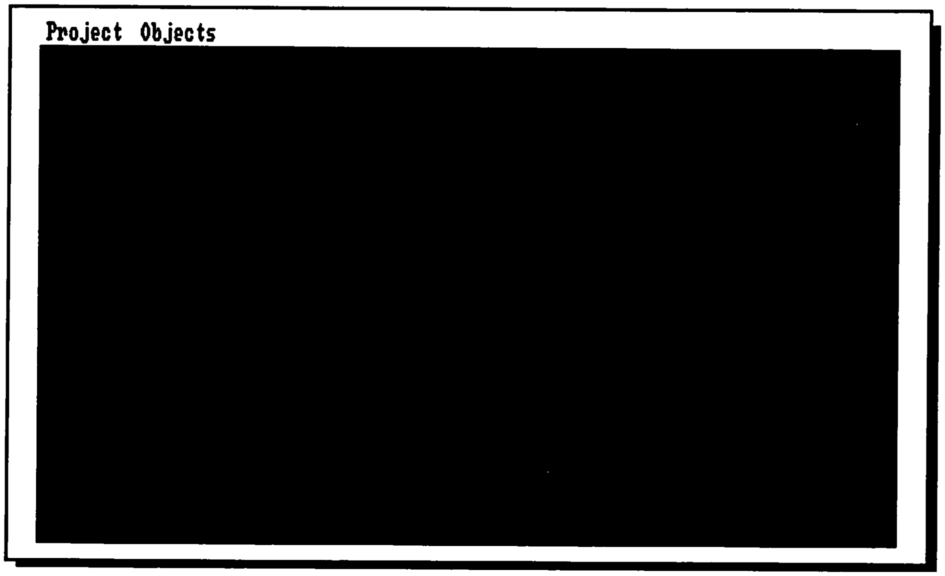


Figure 15-2
The Object editor menus appear when the right mouse button is held down

The Project Menu

The Project menu (Figure 15-3) allows you to load or save a defined set of objects, to configure the screen, to access a frame of video from a laser-disc player, or to test the interface you are designing. There are nine options in the Project menu.

The first Project menu option is Screen. When you move the mouse pointer over this selection, a submenu appears offering four items: Definition, Palette, Video-disc, and Clear (Figure 15-4).

If you select the Definition item, the Screen Definition requester appears (Figure 15-5). With this requester you can determine the type of screen, its resolution, and number of colors. Alternately, you can use Load to load an image and let it determine the characteristics of the display.

By clicking on the multistate gadget at the top left of the requester, you can toggle between high resolution (640 bits wide), low resolution (320 bits wide), HAM, or Hold-And-Modify mode (320 bits wide, 4096 colors), and Extra_Halfbrite mode (320 bits wide, 64 colors). The multistate gadget immediately below this — labeled Color Select — controls the number of colors the screen will use. The

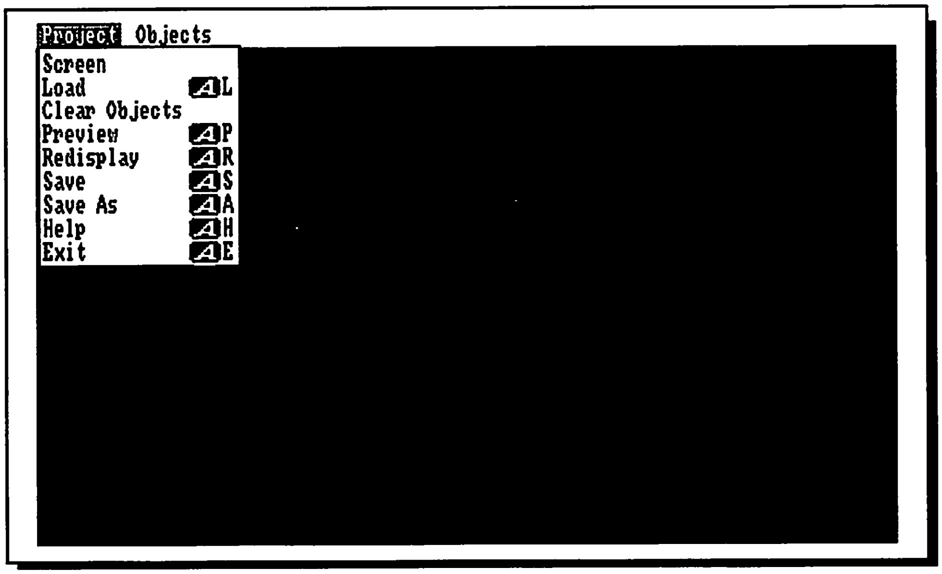


Figure 15-3
What you get when you select the Object editor Project menu

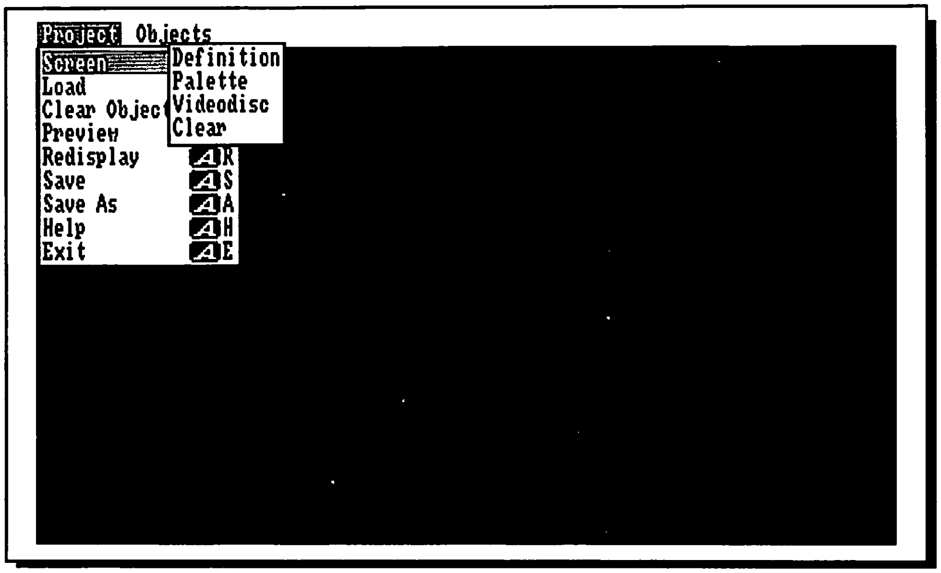


Figure 15-4
What you get when you select the Object editor Screen submenu

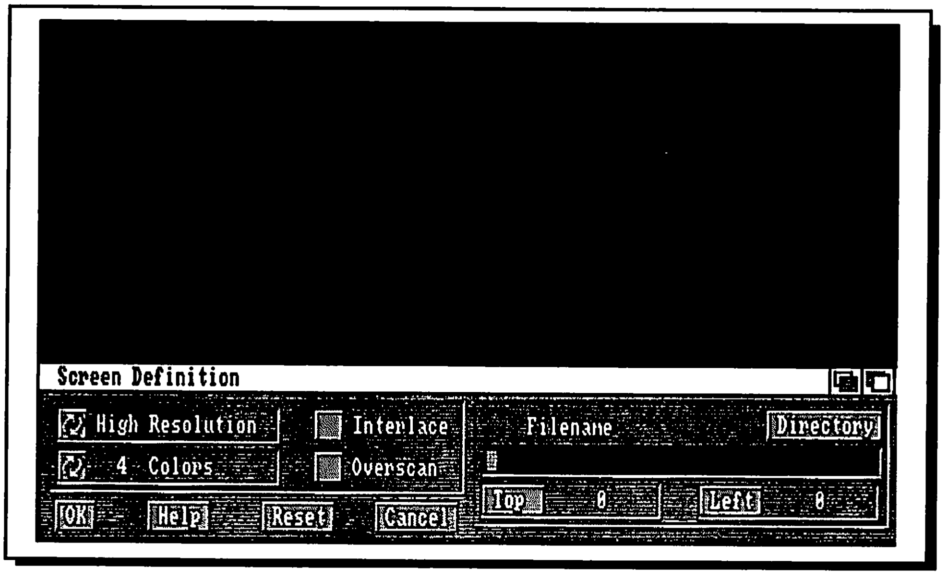


Figure 15-5
The Object editor Screen Definition requester appears when the Definition item is selected

options it allows depends upon which display mode you choose. If you set the screen mode to high resolution, this gadget allows you four choices: 2, 4, 8, or 16 colors. If you set the display mode to low resolution, the color options will be 2, 4, 8, 16, or 32. However, if you choose either HAM or Extra_Halfbrite, the Color Select gadget becomes ghosted and unavailable. This happens because each of these modes has only one possible number of colors: 4096 for HAM and 64 for Halfbrite.

To the right of these multistate gadgets are two check-box gadgets. The top one is labeled Interlace, and when selected, it changes the screen from a 200-line to a 400-line (interlaced) display, regardless of the mode set by the top multistate gadget. (Interlaced mode, as you might be aware, is notorious for its flickering on standard RGB monitors. This effect can be eliminated, however, by using MicroWay's flickerFixer or Commodore's A2320 Display Enhancer on the A2000, or by using ICD's Flicker Free Video on the A500 or A1000 — or with another similar device. No such board is needed for the A3000, as it has a built-in solution to automatically remove flicker. In any case, you will also need a multiscan monitor to get the desired effect.)

The bottom gadget is labeled Overscan. When enabled, it allows the Amiga to use overscan modes, which eliminate borders on the screens and increase both the horizontal and vertical resolutions of the display. Overscan is often used when the Amiga's output is videotaped because it makes the display appear less like it was generated by a computer and more like a television display. The exact amount of overscanning used is dependent on what mode you selected for overscan in the main AmigaVision Project/Defaults menu (see Chapter 4). Below is a table that shows the possible display resolutions when the Overscan gadget is enabled.

Overscan Resolutions

Mode	*No Interlace*	*Interlace*	*Overscan Setting*
Low	352-by-240	352-by-480	Standard Overscan
High	704-by-240	704-by-480	Standard Overscan
Low	368-by-240	368-by-480	Maximum Overscan
High	736-by-240	736-by-480	Maximum Overscan

On the right side of the requester are a standard Filename field and Directory command button. If you want to load an existing image, you can either type its name and path into the Filename field or click on the Directory button and select it from the file requester. In either case, when the image is loaded, the Object editor screen will take on the picture's display characteristics.

Below the Directory and Filename fields is a set of two fields used to define offsets from the top or left of the screen. These work exactly as do their counterparts in the Screen icon (see Chapter 10). When set to any number other than zero, the displayed picture will be moved up, down, left, or right, depending on whether the values you enter are positive or negative. To define an offset, click on either the Top or Left button and enter a value into the Specify Value requester that appears.

Keep in mind that defining the Object editor's display with this Screen Definition requester is a temporary action. It is valid only for the duration of the editing, and when you return to the Flow or Edit windows, the Object editor "forgets" the screen definition you specified. AmigaVision allows you to define the screen so that you can create the graphics, text, and hot spots necessary for the user interface in the same display modes as your application's Screen icon will generate.

The second option in the Screen submenu, Palette, enables you to alter the palette of the screen. When you choose it, a Palette requester (Figure 15-6) appears. This requester will offer from two to 32 colors for modification, depending on the screen mode you have selected. Two exceptions are Halfbrite and HAM modes. In Halfbrite mode, in which you can display 64 colors at one time on the screen, the palette allows the modification of only the first 32. (This is because the second set of 32 colors in Halfbrite mode are half-intensity shades of the first 32. Thus, changing one of the initial 32 colors automatically changes a color on the other set.) In HAM mode, where it is possible to have 4096 colors on screen at once, the Palette requester will have only 16 colors available for adjustment. (As with Halfbrite, this limitation is a result of how the Amiga creates the 4096-color display. In HAM mode, there are only 16 unique colors; the other 4080 colors are generated using the Hold-And-Modify technique from which HAM gets its name. In this mode, each pixel's color is only slightly different from its neighboring pixels. A complete change from any one color to another can be accomplished in three pixels or less.)

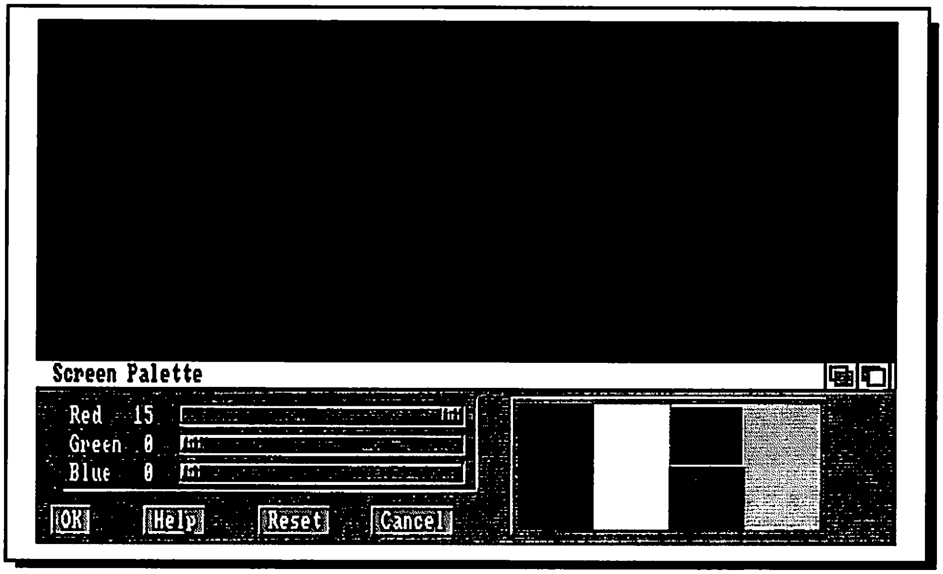


Figure 15-6

The Object editor's Screen Palette requester appears when the Palette item is selected

The third Screen submenu item is Videodisc. Selecting this option causes AmigaVision to load the Videodisc Controller discussed in Chapter 12. From here, you can search for and display frames of video from a laser-disc player. With the video image displayed and the screen set to a suitable mode, you can overlay the video image with objects from the Object editor. Remember that if you have access to a laser-disc player and a genlock, you are not limited to using bitmapped images generated by the Amiga: you can use any video image as an interface!

The final option in the Screen submenu is Clear. When you select Clear, Amiga-Vision removes any picture you might have loaded into the Object editor, but does not remove the display objects you added. For example, suppose you have loaded an IFF picture of an insect and added hot spots and labels to the display. If you then select Clear, the insect picture will disappear leaving the hot spots and labels intact.

The second option on the Object editor's Project menu is Load (the keyboard shortcut is ⒶL). You can use this option to call up any previously defined display object files. Selecting Load summons a Load Object file requester from which you can choose the object file you want to use. Keep in mind, however, that loading a display object file erases anything you have already done to the screen.

Third on the Object editor's Project menu is Clear Objects. (This was called Clear All in versions of AmigaVision prior to V1.53G, and is referred to as Clear All in the AmigaVision manual.) This command does just what its name implies: it clears any display objects you have added to the screen. It does not affect background pictures, however. This command is the exact opposite of the Screen submenu's Clear command, which erases background pictures but not objects.

You can use the fourth Object editor Project menu option, Preview (ⒶP), to test the interface you have designed. This preview performs only those actions defined in the Object editor, and not those in the main program. With it, you can look at the colors, see how the text lines up, and listen to the digitized feedback sounds, in order to determine if the interface elements are as you want.

Fifth on the Object editor Project menu is Redisplay (ⒶR), which simply redraws the display objects. This is helpful when you have done a lot of editing and, as a result, some of the objects appear to be missing segments. Generally, this missing-segment phenomenon means that some objects have been overwritten by others. Redisplay refreshes everything on the screen and redraws the missing pieces.

The next two options are Save (ⒶS) and Save As (ⒶA). These are used to store the current set of display objects as a file (with the .dob suffix) that can be loaded and reused at a later time. Save As gives you the chance to name the file before saving it, while Save stores the file using its current name, automatically replacing any existing file of the same name. (If you select Save when you want to store a new, unnamed file, AmigaVision will respond as if you had chosen Save As, giving you the opportunity to name the file.) Remember, only the display objects are saved.

The last two options on the Object editor Project menu are Help (ⒶH) and Exit (ⒶE). Choosing Help opens a standard Help window that gives instructions on using the Object editor. Exit lets you quit the Object editor and return to the AmigaVision editing environment.

The Objects Menu

The second menu of the Object editor is the Objects menu (see Figure 15-7). From this menu you can add any of the display objects supported by AmigaVision, and manipulate objects that have been added to the screen.

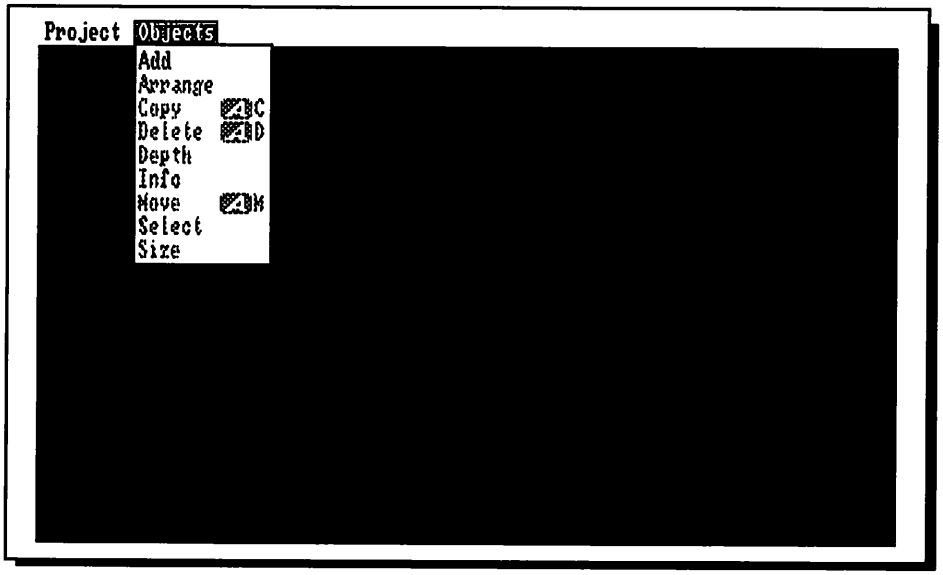


Figure 15-7
What you get when you select the Object editor Objects menu

All but one of the options in the main Objects menu are concerned with modifying or manipulating display objects already on the screen. In Figure 15-7 they appear ghosted because no display objects are yet present. Those commands — Arrange, Copy, Delete, Depth, Info, Move, Select, and Size — will be covered later in this chapter once we discuss how to get display objects on the screen.

The only option that is actually available in the Objects menu when you access it from a blank screen is the first option — Add. If you move your mouse over this item, a submenu will appear (Figure 15-8) listing all of the available display elements you can add to your programs. These objects are Rectangle, Polygon, Line, Circle, Ellipse, Text, Brush, Input Field, and Text Window. (You can also select these options using the keyboard equivalents listed on the right side of the submenu. Note that the keyboard equivalents for the display objects are single-character, lowercase letters.) Each of these objects has a corresponding requester that can be used to customize its appearance and purpose.

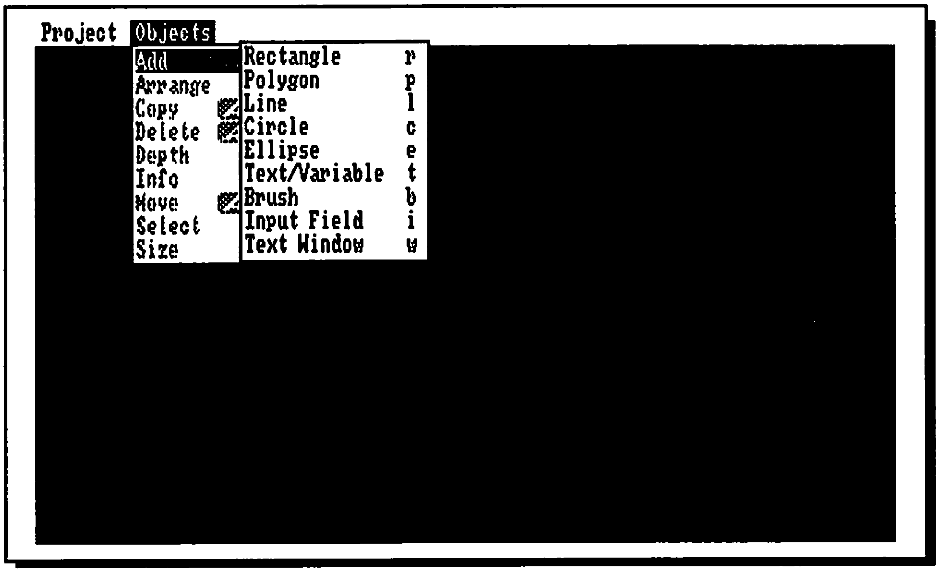


Figure 15-8
The display objects found under the Add submenu

Rectangle

The Rectangle command (*r* is the keyboard equivalent for selecting this object) is used to draw a rectangle — either as an outline or a solid object. To use it, select Rectangle from the menu or press *r* and your mouse cursor will change to a cross hair. Move the cross hair pointer to the point at which you want a corner of the rectangle to appear, and press the left mouse button. Then, while holding the mouse button down, move the cursor to open up a rectangle. As you move the mouse, the rectangle will change, getting larger as you move further from the point of origin, or smaller as you move closer to the original corner. When the rectangle is the size and shape you want, release the mouse button. Don't worry if the rectangle is not perfectly positioned — you can adjust its placement later using the Move function on the Objects menu.

Once you have drawn the rectangle, you will notice that its border appears to flash. This indicates that it is the active display object. Objects menu options such as Move, Copy, Delete, and so on are available only to the active display object, and only one object can be active at a time. To define an object, simply double-click on it or click on it once and then select Info from the Objects menu. Either action will summon an Info requester — the same requester used for all the geometric display objects (rectangle, line, polygon, circle, and ellipse). We will examine this requester in detail in relation to the Rectangle display object, but keep in mind that the information is relevant to the Line, Polygon, Circle, and Ellipse commands as well.

From the Rectangle Info requester (Figure 15-9), you can select the colors, feedback sounds, and response strings that the object will use. You can also choose the visual feedback for the object and/or link it to a Boolean variable.

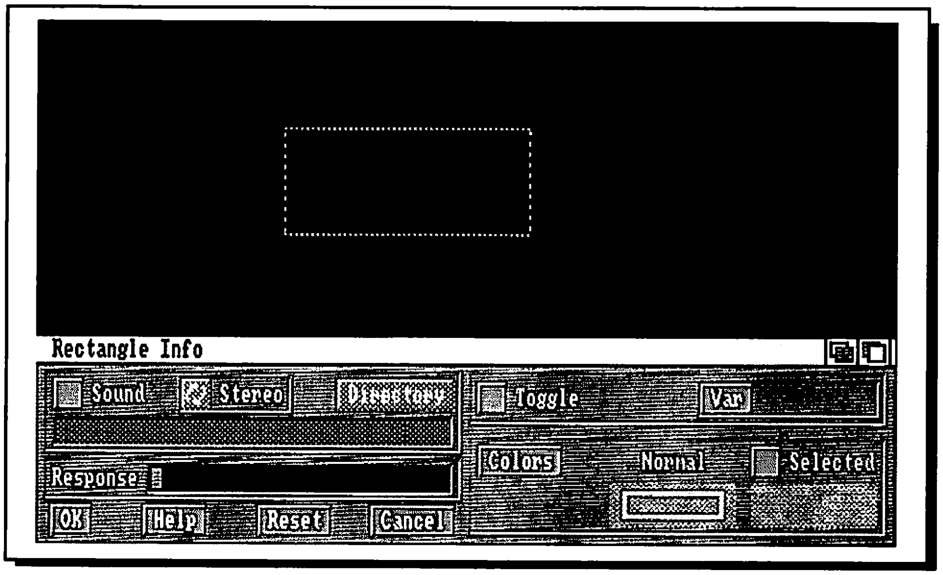


Figure 15-9

The Rectangle Info requester lets you select the colors, feedback sounds, and response strings the object will use

Suppose you want the object to give off an audible sound when the user clicks on it with the mouse. To set this up, you can simply click on the Sound button in the upper-left corner of the requester. (When enabled, the Sound gadget displays a check mark to indicate that a digitized sound is to be played when the user clicks on the object.) If you do nothing else, the program will use whatever has been designated as the system default sound. If you want to use a custom sound, however, click on the Directory gadget to the right of the Sound button. This brings up the standard file requester, through which you can select a digitized sound from your sound drawer. The field below the Sound and Directory gadgets will then contain the path and name of the selected sound. You can further customize the sound aspects of the object by clicking on the multistate gadget between the

Sound button and the Directory gadget. This gadget switches between Stereo, Left, and Right, allowing you to direct the output of the sound to either or both of the sound channels.

Below the sound options is the Response field. Here you can enter the text string you want AmigaVision to use when the object is clicked. For example, if you are building a database interface, you will likely need an Insert button. If you enter INSERT into the Response field for that object, every time the user clicks on the button, the main AmigaVision program would receive that word when the Response() function is used. The program will then know that the user selected the Insert gadget.

Although you can enter very long phrases into the Response field, I suggest that you use phrases that are clear and just long enough to be easily recognized. If you use a phrase that is too long or one you might have trouble remembering, using it in your program will be a bit more difficult.

Entering text into the Response field turns the display object into a hit box (hot spot). If this field is left undefined, the object is merely a graphic element that, when clicked, has no affect on the rest of the program.

On the lower right of the requester you will see a Color command button, a Selected button, and two fields showing the currently selected colors. Clicking on the Color button brings up a requester that allows you to set the border and fill colors for the object's normal (unselected) state and its selected state, that is, the way it appears once it has been clicked. (See Figure 15-10.)

There are two multistate gadgets on the Select Color requester. The first toggles between Normal Border, Normal Filled, Selected Border, and Selected Filled. Each of the geometric objects (rectangle, polygon, circle, and ellipse) can have different colors for its border and its internal space. In addition, you can toggle the border and internal colors to be visible or transparent using the second multistate gadget.

To assign a color, merely click on the top gadget until it displays the option you want, and then toggle the bottom gadget so that it reads "Visible." Then, using the mouse, choose a color for that option by clicking on one of the boxes in the palette; when you do, the color box you select will appear to be outlined.

Once you have determined the colors you want, click on the OK button on the Select Color requester. You will be returned to the Rectangle Info requester, where the two color fields will now reflect the choices you made for the normal and selected modes in the Select Color requester. The object will not automatically toggle between the two color modes, however, unless you enable the Selected button by clicking on it. (When enabled, the Selected gadget will display a check mark.)

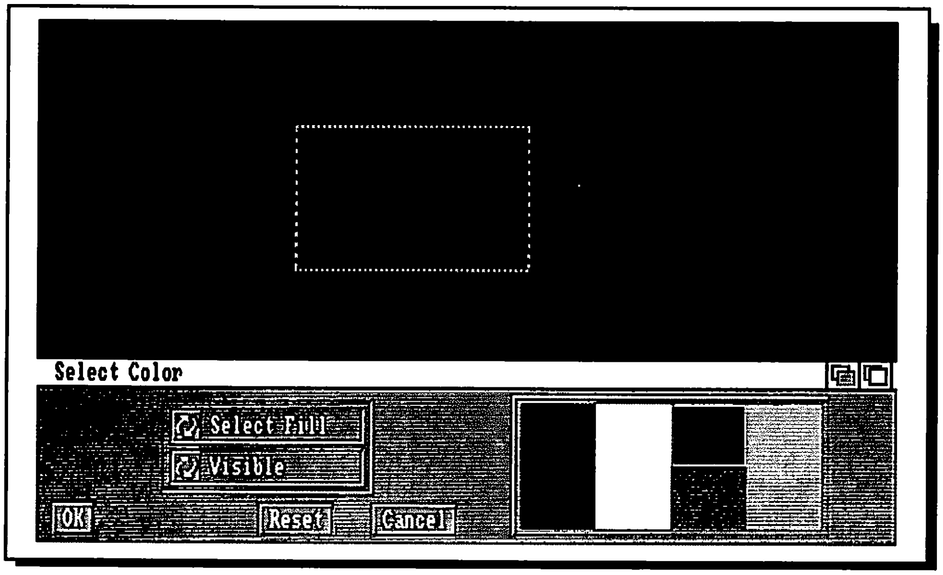


Figure 15-10
The Select Color requester appears when the Color button is clicked

On the top right of the Info requester are a check-box button labeled Toggle and a command button labeled Var. Each of these can greatly influence the way the display object reacts.

When enabled, Toggle tells AmigaVision to simply change the state of the display object — from unselected to selected (if the object was not selected when you clicked Toggle) or from selected to unselected (if it was already selected). In other words, when you enable Toggle mode, the corresponding object will switch from whatever mode it was to the other mode every time the user clicks on it. Once in the new mode, the object will remain there until the user switches it back by clicking on it again.

When Toggle is not enabled, the result when the user clicks on the gadget is quite different. In this case, the mode changes, but only temporarily; it then rapidly switches back to its original state. How you set the Toggle button will depend on the program you are writing, but both modes are very useful.

The Var button allows you to link the object to a specified Boolean variable. (Remember that Boolean variables have only two possible values: true or false.) When you click on the object, the state or value of the linked variable changes, based on whether the object is selected.

Var has an unexpected — and undocumented — feature, that is, the linkage from a display object to a Boolean variable is two-way. This means that not only does the variable change when you click the object, but altering the variable affects the linked object! Let's think about the uses of this feature. Suppose you design an interface that has four buttons, each representing a state of activity for the program. Further, suppose that the program can support only one of the activities at a time. If you set the four buttons to be toggles, it would be possible to have more than one selected at the same time. However, if you add a routine to your program to change the value of the other variables to false every time the user selects one of the options, the variables that change to false will cause the display object linked to it to also become false, or unselected. Using this technique, only one menu item will be selected at any given time (see Figure 15-11).

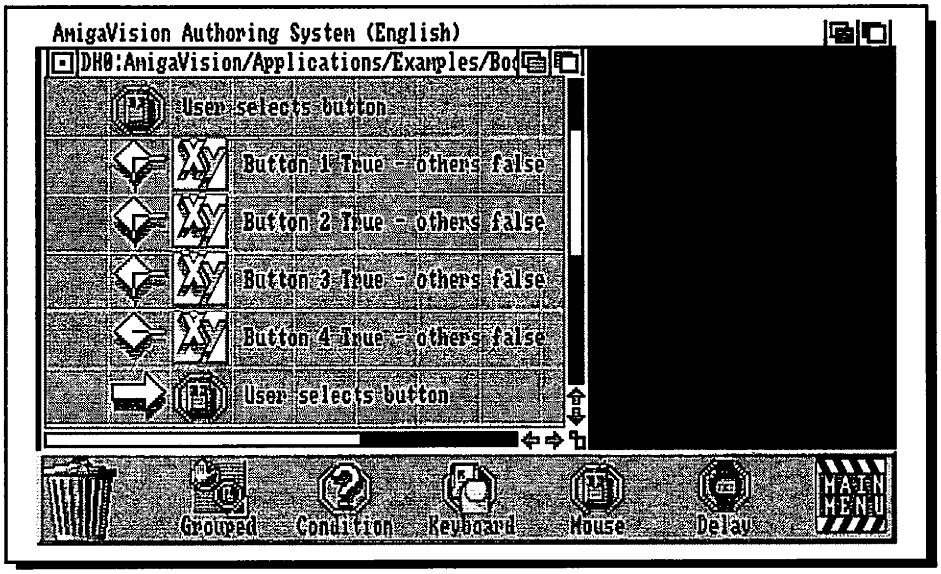

AmigaVision Authoring System (English)
DH0:AmigaVision/Applications/Examples/Bo
User selects button
Button 1 True - others false
Button 2 True - others false
Button 3 True - others false
Button 4 True - others false
User selects button

Figure 15-11
Altering objects linked to Boolean variables

Polygon

The Polygon (p) command is used to outline irregular objects that do not fit neatly inside such geometric shapes as rectangles, circles, or ellipses. To create a polygon, move the mouse pointer to the position at which you want to begin drawing and click the left mouse button once. As you move the mouse, you will see a straight line stretching from the point of origin to the mouse pointer's current position. Move the pointer to the screen position where you want to add a section of the polygon and click the left mouse button again. A line segment will be drawn, and as you continue to move the mouse, another line will follow your mouse cursor from the second point. Continue this process, moving the mouse and clicking the left mouse button at the points you want the corners of the polygon to be. Finish by connecting a line back to the original point. When you have finished drawing the shape, click the right mouse button once. This will complete the polygon, drawing a line from the first point to the last. With a little practice you will find yourself quickly becoming proficient at drawing polygons.

A good way to practice drawing polygons is to load a detailed, bitmapped picture such as a digitized image, and outline the components of the picture, using the polygon tool. (See Figure 15-12 for an example.)

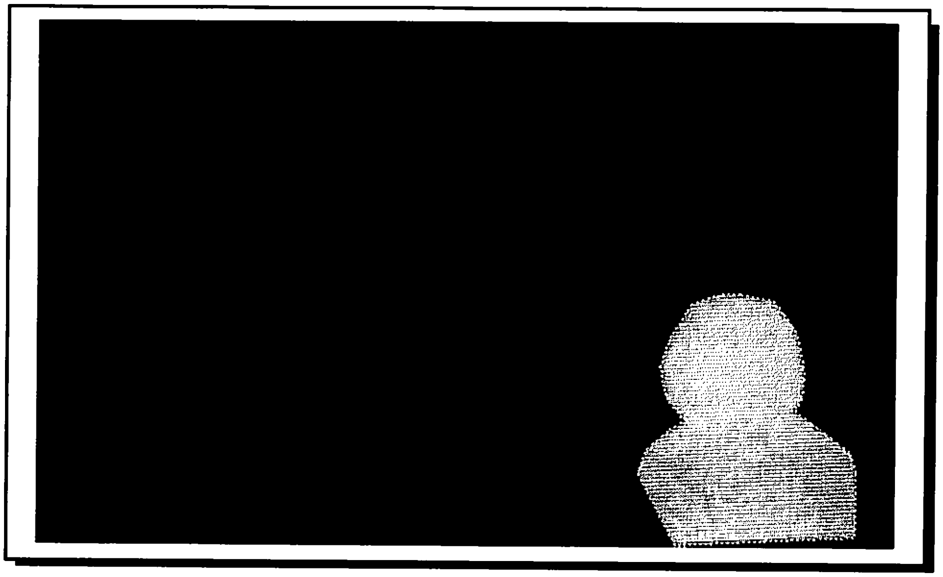


Figure 15-12
A polygon outlining of a section of a picture

To define the characteristics of the polygon, just double-click on it. This summons an Info requester that is functionally the same as the one we examined in relation to the rectangle object. If you want the polygon to be a hit box (hot spot), add some text to the requester's Response field.

Line

The Line (l) display object is very simple. It is not really suitable for use as a hit box, but is useful for enhancing the graphic display. To add a line to your screen, just select Line from the menu, place the mouse pointer where you would like the line to begin, and click the left mouse button. Then move the mouse cursor to the spot where you would like to place the other end point and click again.

Although the Line display object is not particularly useful as a hit box, it nevertheless has an Info requester that you can access by double-clicking on a line you have drawn. This requester is identical to that associated with the rectangle, except that its corresponding Select Color requester does not have an option for an outline color for the line object (the line is limited to one pixel in thickness).

Circle/Ellipse

The AmigaVision Object editor has two circular geometric objects, Circle (c) and Ellipse (e). Both are created by the same technique: you position the mouse pointer at the point on the screen that you want the center of the object to appear, press and hold down the left mouse button, and move the mouse to expand or reduce the circle or ellipse until it is the right size, then release the left mouse button. Further defining a circle or ellipse follows the conventions described for the rectangle.

Each of the next four display objects has its own Info requester. Of these four, two (Text/Variable and Brush) are always available to any of the icons that have Object editor access. The other two (Input Field and Text Window) are available only from the Data Form and Text icons, respectively. There is just one exception to the limited access of these objects: if you have accessed the Object editor from the Tools menu, all display objects are always available.

Text/Variable

The Text/Variable (t) display object allows you to place textual and numeric information on the screen in any font, color, and soft style. (The term soft style refers to the display characteristics of the character. AmigaVision's soft style modes are bold, italic, and underline.) You can use the Text/Variable object to display the contents of variables.

Placing a Text object on the display is easy. After selecting Text from the Object editor's Add menu, point with the mouse to the position at which you would like to begin your text box. Click the left mouse button and drag open a rectangle the approximate length of the string you will be displaying (the size can be adjusted later). Then double-click on the rectangle you have created to bring up the Text Info requester (see Figure 15-13).

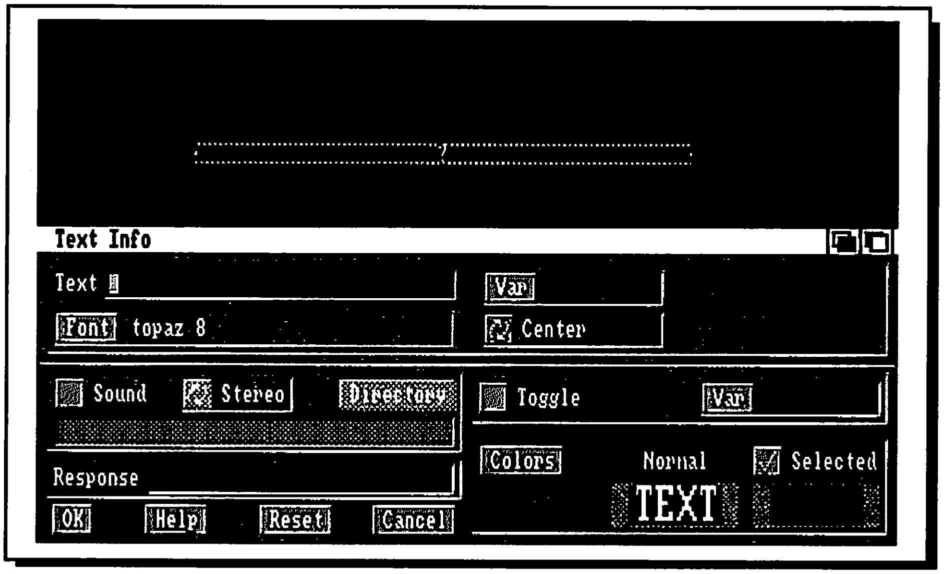


Figure 15-13
The Text Info requester appears when the rectangle is double-clicked

As you can see from Figure 15-13, the Text Info requester shares many of the features of other Info requesters. You can turn your text box into a hot spot by adding a response string to this Info requester. Also through this requester, you can link digitized sounds with your text for audible feedback when the user clicks on it. In addition, you have the option of specifying normal and selected colors for the text. The Select Color requester for the Text/Variable object is a bit different than other ones, as it does not have an outline option; instead, it offers a background color option. You can choose these background colors from any of those available in the palette and, as with other display objects, make them transparent to allow other display objects or background images to show through.

Other standard features are the Toggle and Var buttons, which operate here just as they do in the other Info requesters. Toggle switches the mode of the display object from one state to another (selected to unselected or vice versa) and leaves the object in that mode until the user clicks on it again. Var allows you to link the object to a Boolean variable.

At the top of the requester you will find several new options. On the top left is a Text Field and below that is a Font command button. You can enter any text string you want printed — up to 255 characters — into the Text Field. Keep in mind, however, that the Text/Variable display object will print only what will fit on a single line. No line wrap occurs if the string is longer than one line; anything extra is just truncated. Of course, how much will fit on a line depends on the typeface and point size you are using.

To change the typeface or point size of your text, click on the Font command button. This will open a new requester, the Available Fonts requester (see Figure 15-14). From here you can select any font in your Amiga's Fonts: directory. The available fonts are listed in the left-most window, and you can select one by just clicking on its name. To the right is a smaller list window showing all the available point sizes of the selected typeface. Selecting a point size is as easy as selecting a font: just point to your selection with the mouse cursor and click the left mouse button. (Both of these list windows are equipped with scroll bars and gadgets, so if all the font names or sizes will not fit into a single window, you can easily move through the list.)

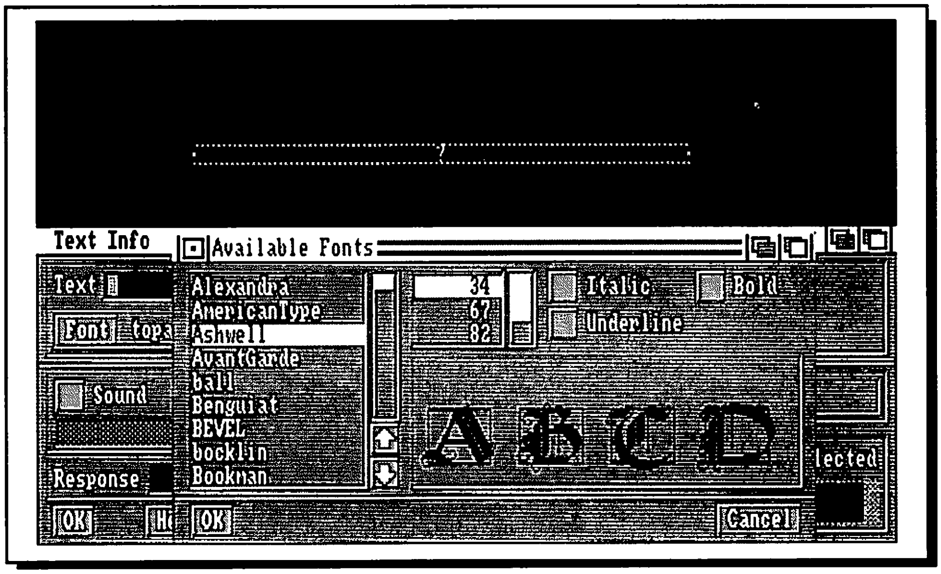


Figure 15-14
The Available Fonts requester appears when the Font button is clicked

To the right of the point-size window is a set of three check-box buttons. Each controls a soft style setting (Italic, Bold, or Underline) that can be used when the string is printed. Below these buttons is a display window that shows your current font in the size and soft style you have selected, allowing you to determine at a glance if these are indeed the settings you want to use. Using the commands and options on the Available Fonts requester makes it very easy to examine a large number of fonts and sizes while designing your user interface.

Another option unique to the Text Info requester is a second Var button. It appears at the top right of the requester and allows you to print the value of a variable onto the screen. Clicking on it brings up a Specify Variable requester (see Figure 15-15), from which you can select any of the defined variables in your program for output.

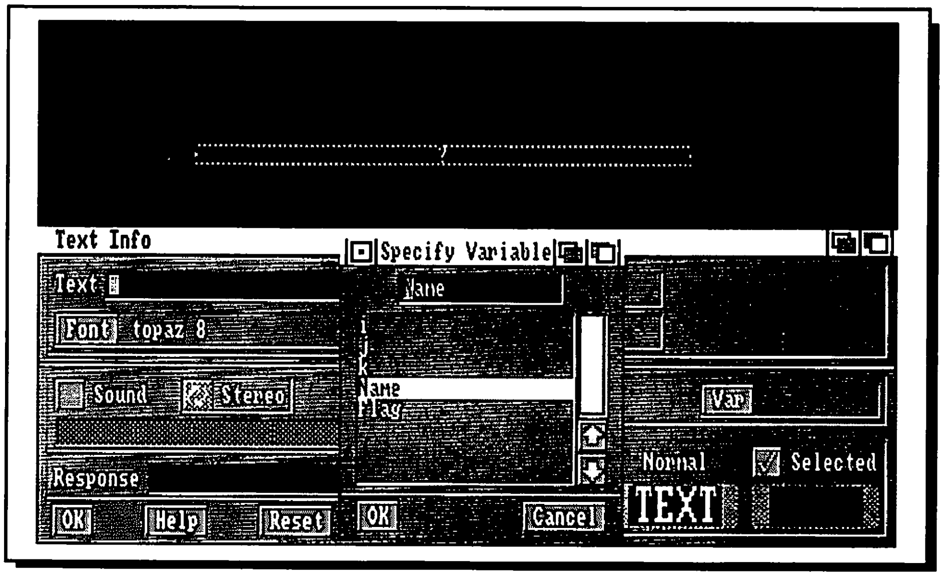


Figure 15-15
Selecting a variable for the Text/Variable display object

If you leave the Text Field empty and select a variable, you can print just the value of that variable. If you add a string to the Text Field *and* select a variable for output, however, the text string will be printed first and the variable value will be printed immediately afterward. This can be very useful when you want to print both a variable and a string describing the variable data. For example, suppose you have a variable *x* with a value of 10. If you enter the text string "The value of *x* is" into the Text Field and then select *x* from the variable list, the output would appear on screen like this:

```
The value of x is 10
```

Another feature of the Text Info requester is a multistate gadget that determines how the output is positioned, or justified, within the text rectangle you create. Clicking on it switches the mode between Center, Left, and Right. Select whatever mode is best for your application.

Once you have finished determining the color, position, typeface, and point size of the text to be output, click on the Text Info requester's OK button. When the requester disappears, you will notice that the text rectangle is modified to meet the requirements you selected. Besides having its color changed, it will also be larger or smaller, depending on the font and the point size you selected. You can move or resize the rectangle as necessary using the Move and Size commands from the Objects menu.

Brush

A very powerful feature of the Object editor is the ability to use brushes (small pieces of bitmapped images) as display objects. These Brush (b) display objects can be used either as hit boxes or as non-responsive parts of the display.

Selecting Brush from the Object/Add menu (or pressing b) gives you a cross hair cursor very similar to that used to draw a rectangle — in fact, you use the same technique to draw a rectangle display object as you do to draw a rectangle to house the brush. Once you have drawn a brush rectangle of the approximate size you need, double-click on it to bring up the Brush Info requester (Figure 15-16).

As you can see in Figure 15-16, the Brush Info requester has several features that we have already covered in detail (i.e., Sound, Response, Toggle, and Var buttons) in our discussions of other requesters. Thus, we will limit this discussion to the many features that are unique to this requester.

You will notice that at the top of the requester there are two Directory gadgets and two file-name fields. These are necessary because you can use either one or two brush images in a single display object. The left set of controls is for defining the unselected brush image — that is, the brush as it normally appears on screen — while the right set is for defining the selected brush image —that is, the way the brush appears when the user clicks on it. When you employ two brushes, the display changes to the alternate (selected) image when someone clicks on it. To give you an example of how this is useful, suppose you have drawn a button to be used as a display object, and that you want the button to look as though it has been pressed down when the user clicks on it. By drawing the second image to look like a depressed version of the first button, you can give definite visual feedback to the user of your program.

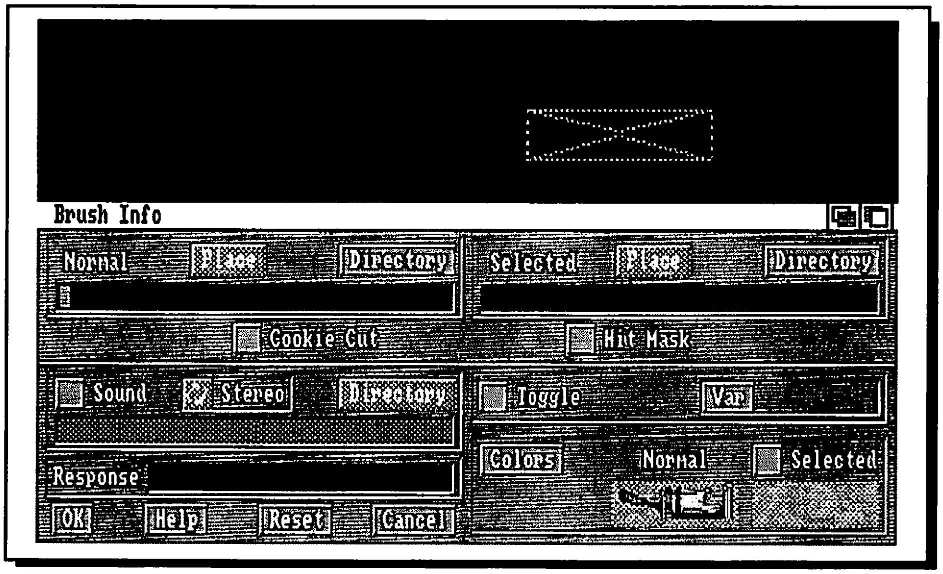


Figure 15-16
The Brush Info requester appears when the brush rectangle is double-clicked

Each of the two sets of controls has a button labeled Place. Selecting it allows you to use a brush that is larger than the rectangle you have drawn, moving the brush within the rectangle so that only the section you are interested in shows. Place allows you to easily position the brush. The result is that only part of the brush becomes a display object.

Below the directory/file-name fields are two check-box gadgets: Cookie Cut and Hit Mask. These buttons are used to determine how much of the brush is shown and how much is an active hot spot.

Every brush is a rectangular area of bitmap, and without Cookie Cut enabled, every bit of that rectangle shows on your display screen. In many cases, however, much of the rectangle won't actually be part of the image you want to include on your display. Consider a brush that contains a round button. While the brush actually consists of a rectangle surrounding the button, with Cookie Cut enabled, all the background area of the brush (background is usually color 0) is treated as transparent, so that only the circle is displayed.

Just as Cookie Cut determines what is visible in the brush, Hit Mask determines what is active — in other words, how much of it is a hot spot. If Hit Mask is not enabled, clicking anywhere in the rectangle (even if part of it has been rendered transparent by enabling Cookie Cut) is considered a click on the hot spot. With Hit Mask enabled, however, only the visible part of the brush (the nontransparent part) will act as a hot spot.

If you click on the Colors button, the Select Color requester appears (Figure 15-17). This feature works differently in relation to the brush display object than it does in other circumstances. In this case, instead of selecting a color for the brush, you choose how the brush is to be displayed. Associated with each brush (selected as well as normal) is a multistate gadget that cycles between three display modes: Transparent, Multi-colored, and Stencil.

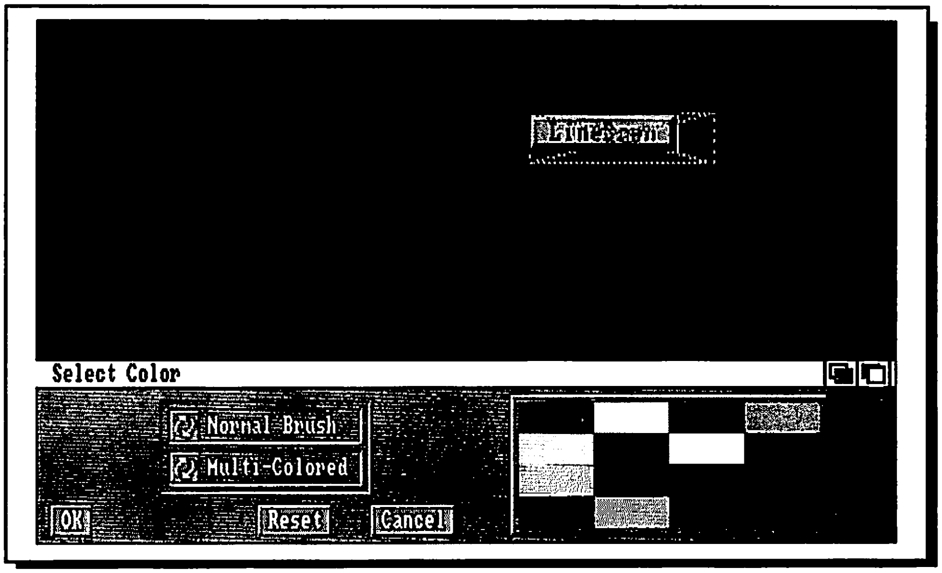
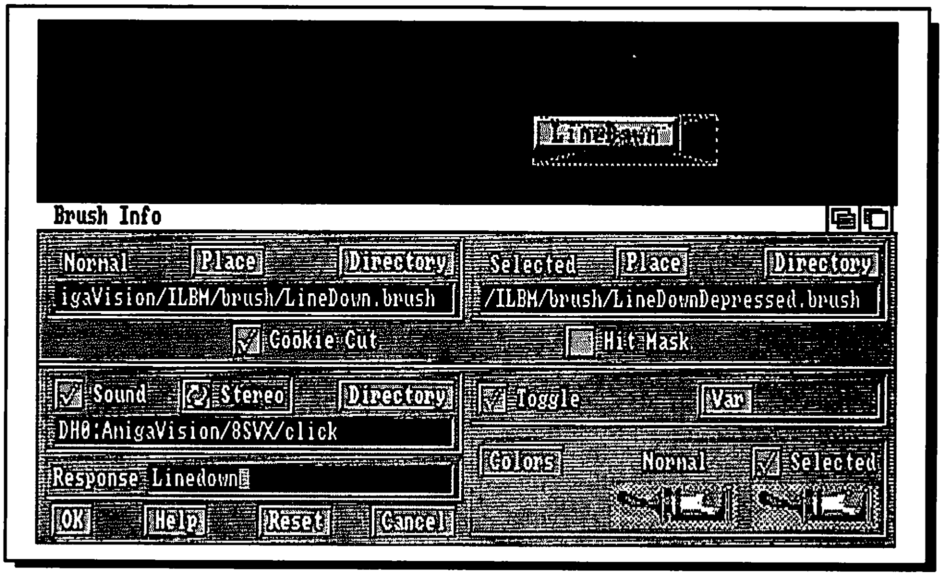


Figure 15-17
The Select Color requester appears when the Color button is clicked

Transparent makes the brush invisible, so that only the background image or display objects under the brush will show. Selecting Multi-colored instructs AmigaVision to make the brush visible. Stencil mode shows display in one-color brush. If you select Stencil, you can also choose a color for the stenciled image from the palette to the right.

Keep in mind that AmigaVision does not display the brush using the brush's own palette; instead it uses whatever palette the screen happens to be in. If you are not careful, then, a beautiful brush can end up looking extremely odd. The solution to this potential problem is to initialize the screen with a picture that uses the same palette as the brush. (To get the palette I need without any unwanted images, I often load a blank screen that was saved using the brush's palette.)

Although in some cases you will use the Brush display object with only a single image, the results are much better if you take advantage of the double-image feature. Figure 15-18 shows an example of a fully defined brush object complete with two images. In this case, the image is a button that appears to be depressed when clicked.



Normal: igaVision/ILBM/brush/LineDown.brush

Selected: /ILBM/brush/LineDownDepressed.brush

✓ Cookie Cut

Hit Mask

✓ Sound

Stereo

DH0:AmigaVision/8SVX/click

✓ Toggle

Response Linedown

Normal

✓ Selected

Figure 15-18
A fully defined brush object that includes two images

Input Field

The Input Field (i) display object allows a program to accept text or numeric input from the user of your program. This option is available only when the Object editor is accessed from the Data Form icon or from the Tools menu. (From all other icons, the option appears ghosted in the menu.)

To create an Input Field, select Input Field from the menu (or press the i key). A cross hair will appear to allow you to create a rectangular field in much the same way as you would create a text box. Once you have the Input Field on the screen, double-click on it to open the Field Info requester (see Figure 15-19).

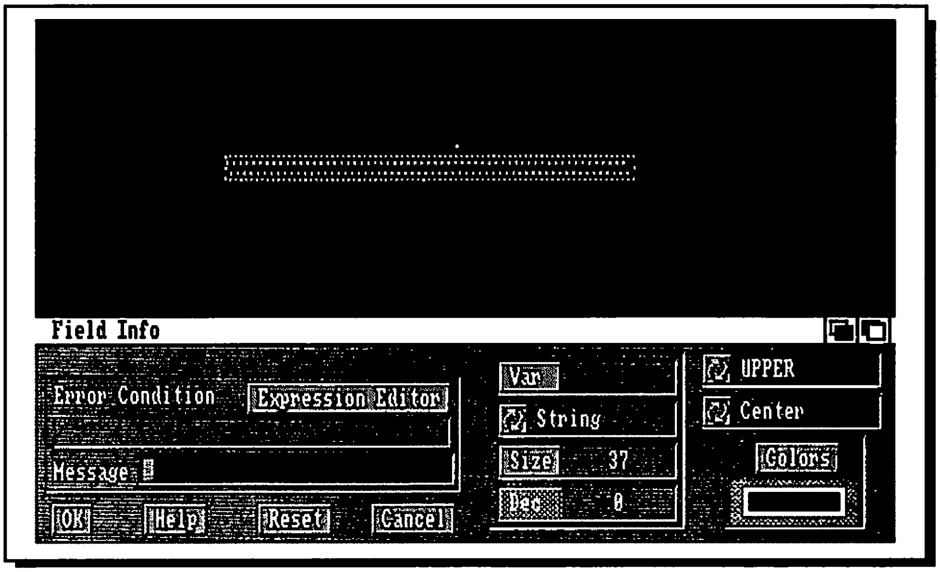


Figure 15-19
The Input Field Info requester appears when the Input field is double-clicked

Basically, the Input Field allows you to link the field to a specified string, numeric, Boolean, or date variable. While the date and Boolean data types are of a fixed length and format, you can designate strings of any length up to 255 characters and numbers up to 15 digits. To choose a variable with which to link your field, click on the Var button. Doing this brings up a Specify Variable requester from which you can select any defined variable.

You can specify an exact size (in digits) for the field by clicking on the Field Info requester's Size button. This brings up a Specify Value requester which you can use to enter your value. If the input field is to contain decimal values, the Dec field will allow you to enter the number of decimal places to be allowed.

Further input restrictions can be set by clicking on the multistate gadget to the right of the Var button. This gadget cycles between UPPER, lower, or MiXeD, and determines the allowable case of the text strings you can enter into the field. Although you can enter text in whatever case you want in UPPER or lower mode, when you press the Return key all input is converted to the specified case.

You can also position (justify) the text string in the field, just as you can in the Text Info requester. To do so, use the other multistate gadget (above the Colors gadget) to toggle between Center, Left, and Right justification.

Although the Colors gadget allows you to change the border color of the input field, there is no provision for changing the color of the text or the background used. The color of the background is color 0 on your pallet, the color used for the input text is color 1 on your palet. For that matter, you cannot change the typeface or point size used in the field, either; these things are hard-coded in AmigaVision.

Another gadget lets you enter an error condition for the input field. To create such a condition, click on the Expression Editor button. This brings up the Expression editor, wherein you can enter the specific condition test to be performed on anything the user types into the Input Field. If the condition fails, an error message is displayed. You can specify your own custom error messages by entering them into the Error Field. Then, if an error condition occurs — for example, if the input string is greater than a defined length (Len(string) > 20) or a value is out of range (x > 100) — the user will receive that message.

Because the Input Field is so closely associated with the database functions, and especially the Data Form icon, you should refer to Chapter 8 for additional information on this display object.

Text Window

The Text Window (w) can be accessed only from the Text icon in the Audio Visual menu or from the Object editor's Tools menu. It allows you to create informational windows for displaying a disk-based ASCII text file. You can customize the display to use any font with any color or soft style. In addition, you can add formatting commands that can change the soft style of the text or determine how each line is broken.

The Text Window display object is created in much the same manner as a rectangle: using the mouse, you drag open a rectangle and then double-click on it to bring up the Text Window Info requester (see Figure 15-20).

This is a fairly simple requester; it offers only Font and Color gadgets. Clicking on Font brings up the Available Font requester, which allows you to pick the typeface and point size you want to use. It does not allow you to set the soft style; changing soft styles is done by embedding special commands into the source text file.

Clicking on Color opens a Select Color requester. From here you can choose colors for the text, the text window's background, the text window's border, and for highlighted text. (Highlighted text is designated within the ASCII file by enclosing the string with special characters.)

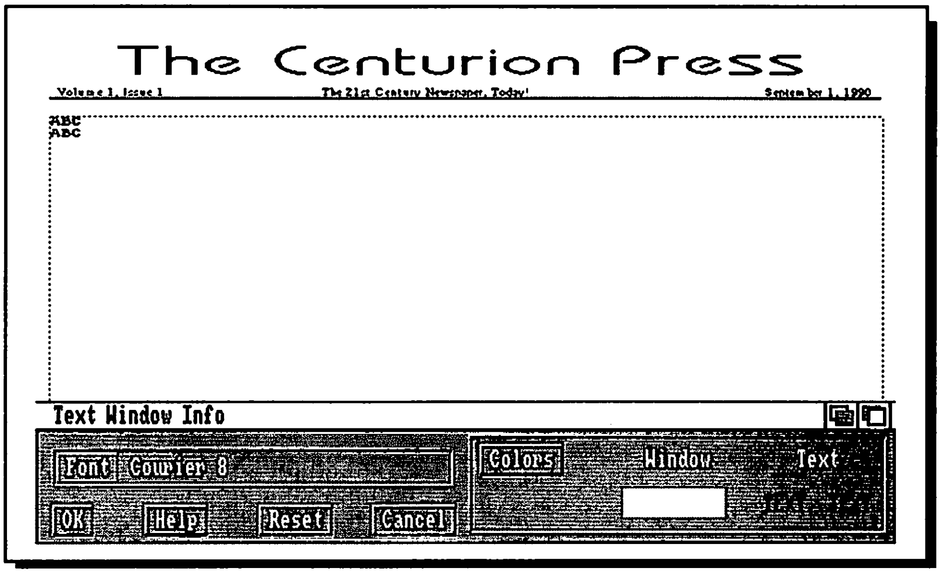


Figure 15-20
The Text Window Info requester only has Font and Color gadgets

Notice the Text Window Info requester offers no provision to select the text file that will be displayed in the window. That task is handled outside of the Object editor, through the Text icon itself. There is no option to search through the file for a specified string, either, although AmigaVision supports such a function — again, through the Text icon.

With a Text Window display object, you can easily include text files in your applications. To allow the user of your program to move through the file, you can add hot spots to the screen display. The hot spots can be brushes, rectangles, circles, text, or any other object that can return a response string. AmigaVision uses these response strings to manipulate the contents of the Text Window. The responses that AmigaVision supports are:

Pagedown
Pageup
Lineup
Linedown
Quit

These must be entered as single words with no spaces. Although your AmigaVision manual incorrectly states that the strings are two-word commands — Line Up, Line Down, Page Up, and Page Down — attempts to control the text window using these strings will fail. Case is not considered, however, so "PAGEUP" is treated the same as "pageup."

If you link one of these commands with a Text Window display object, it will respond accordingly when the user clicks, moving the text up or down a line or page at a time. And once a text window has been opened in your application, program execution halts at the Text icon until the Quit command is given.

The commands to alter the soft style or word wrap are shown below. You must add these instructions using a text editor, not from within AmigaVision. Each command consists of one or two characters preceded by a vertical bar (|).

|B - Bold toggle on/off
|I - Italic toggle on/off
|U - Underline toggle on/off
|H - Highlight toggle on/off
|FD - Normal word wrap using one carriage return
|FW - Normal word wrap using two carriage returns
|FN - No word wrap

The Text Window is discussed in detail in Chapter 10 as part of the information presented on the Text icon. Refer to that chapter for additional information on creating and using Text Windows.

Object Manipulation Features

Once you have defined some display objects, the eight options on the Objects menu that were previously ghosted become available (see Figure 15-21). These options — Arrange, Copy, Delete, Depth, Info, Move, Select, and Size — allow you to modify the objects.

The display objects on your screen are arranged in a definite hierarchy. Higher-order objects are accessed before lower-order ones. The Arrange command allows you to order a group of display objects into a sequence that fits your needs, which is not necessarily the order in which they were created. To use the Arrange command, select it from the menu and then click on each object in the order that you want them to appear in the hierarchy. When you finish ordering the objects, click on the right mouse button.

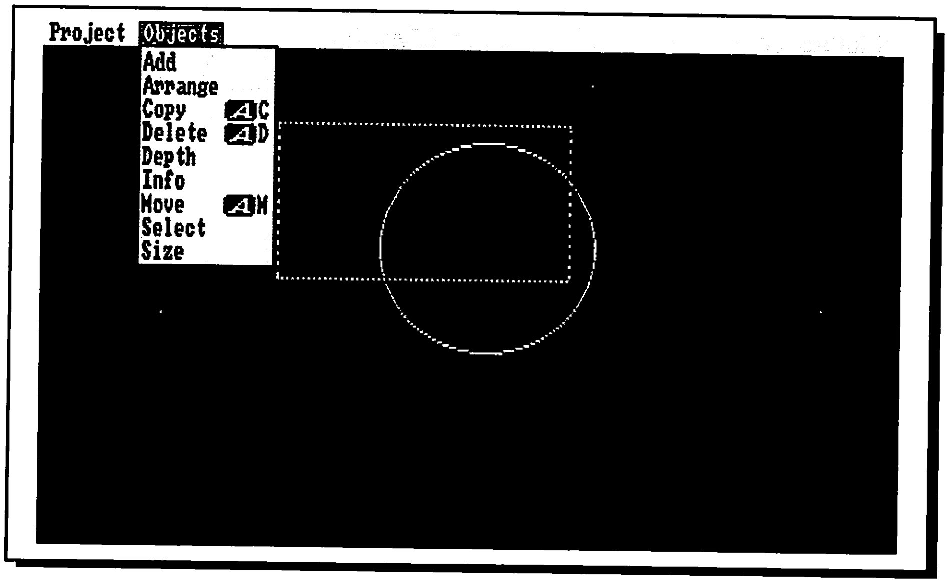


Figure 15-21
The object-manipulation commands become unghosted once some display objects are defined

The arrangement of your objects can be of critical importance when you are designing database input fields. Let's say, for example, that you want the user to enter the data in a certain order. The Arrange command makes ordering the input fields very easy, so you can set up the cursor to jump to the next input field in the sequence as the user finishes entering information in each field.

Often when you design an interface, you will find that you require multiple display objects that share some attributes. Although you can create each object individually, in most cases it is more productive to design just one, and then use the Copy (AC) option to make duplicates of it. You can then simply modify the clone object and thus save yourself a great deal of development time.

To use the Copy function, select it from the Objects menu or use the AC keyboard equivalent, point to the object you want to duplicate, and press the left mouse button. While holding the mouse button down, you can drag a copy of the indicated object to a new location.

With the Delete (AD) option, you can remove from the screen display an object that is no longer needed. Just select Delete from the Objects menu and then click on the object you wish to remove, being careful to select the object you want deleted and *not* another, overlapping object!

Depth, which is related to the Arrange command, allows you to specify which objects take precedence over others in the display. Positions of objects in the stack of display elements can affect the execution of a program because some objects will be more accessible and others will be covered up by higher level objects. Depending on your object's color and on whether it is transparent, this means that unless you order them carefully, large objects can cover smaller ones. A hidden object, of course, is not available as a hit box.

When you place your mouse over the word Depth in the Objects menu, a submenu appears offering four options: Front, Back, Lower, and Raise (see Figure 15-22).

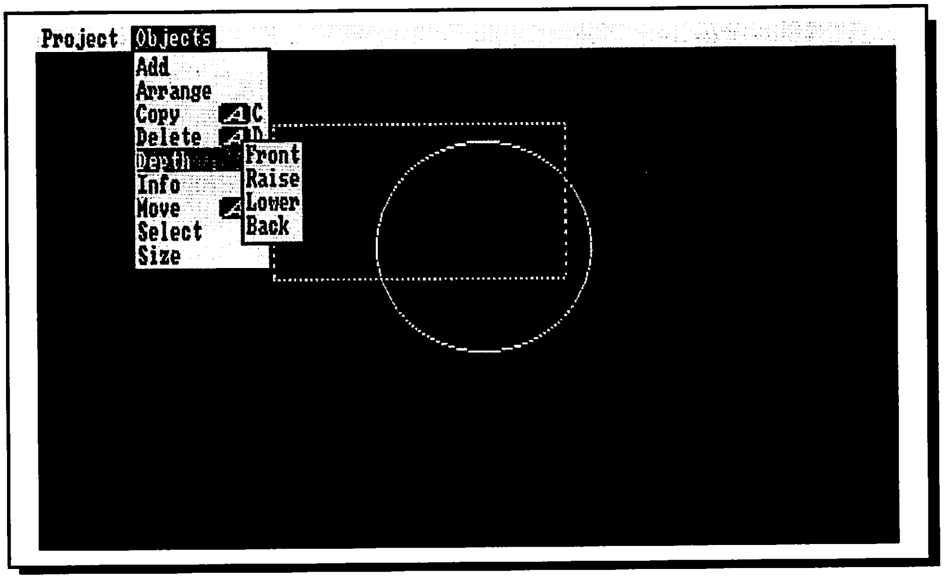


Figure 15-22
What you get when you select the Depth submenu

These submenu options are the tools you use to move objects above and below others, rearranging their positions in the object stack. To select an object for reordering, click on it once (doing so will make the object's outline appear to move). Then select one of the options from the Depth submenu. Raise moves the selected object up one level higher in the object stack, Lower moves it down one level. Front brings the object all the way to the top of the heap, while Back pushes it to the bottom. An object at the top is always accessible, while those under other objects cannot be selected by the mouse in those areas where other objects overlap them.

The Info command is used to open the Definition requester for a display object, and is functionally equivalent to double-clicking the object. To use it, select the object by clicking on it once, then select Info from the Objects menu. Personally, I consider this option to be almost useless. It is much easier just to double-click on the object to open its Definition requester than it is to click on it once and then access the Objects menu.

Move (AM) is used to reposition a display object on the screen. To use it, select an object by clicking on it once, and then select Move from the menu. The AmigaVision manual instructs you to place the mouse pointer on the object and drag it to its new position, but you do not really have to drag it from its point of origin. Once you select the object and choose Move, you can place the mouse *anywhere* on the screen and drag it to another location by holding down the left mouse button. While it is often preferable to actually position the pointer over the object before you begin moving it, this is not a requirement.

The Select command can make an object active. While this is usually done by just pointing to the object and clicking the left mouse button once, Select comes in handy when the object you need is completely covered by other objects. When you position your mouse pointer over the Select option in the Objects menu, a submenu will appear (Figure 15-23) with three options: First, Next, and Previous. Using these commands, you can select an object by moving through the stack in the order determined by the Arrange command.

The Size command can alter the size of any object except a polygon. Just click on the object you wish to reduce or enlarge, choose Size from the Objects menu, position your pointer over the selected object, and then press the left mouse button, holding it down while you move the mouse. The object will be resized according to how you move the mouse. Some objects, such as the Text/Variable and Input Field objects, can be altered only horizontally. (The Text/Variable object's vertical size is fixed by the point size of the font, while the Input Field is a fixed height.)

One of the undocumented features of the Objct editor is the space bar, pressing the space bar will repeate your last command. For example, suppose you had just sized an object, you can select another object, press the spacebar, and be in Size mode again. Just pressing ths spacebar will save you the time of having to choose a command from the Objects pull-down menu.

As you can see, the Object editor is a powerful addition to AmigaVision's authoring tools. With it you can design graphic displays and user interfaces, display text files, add input fields, and even play digitized sounds.

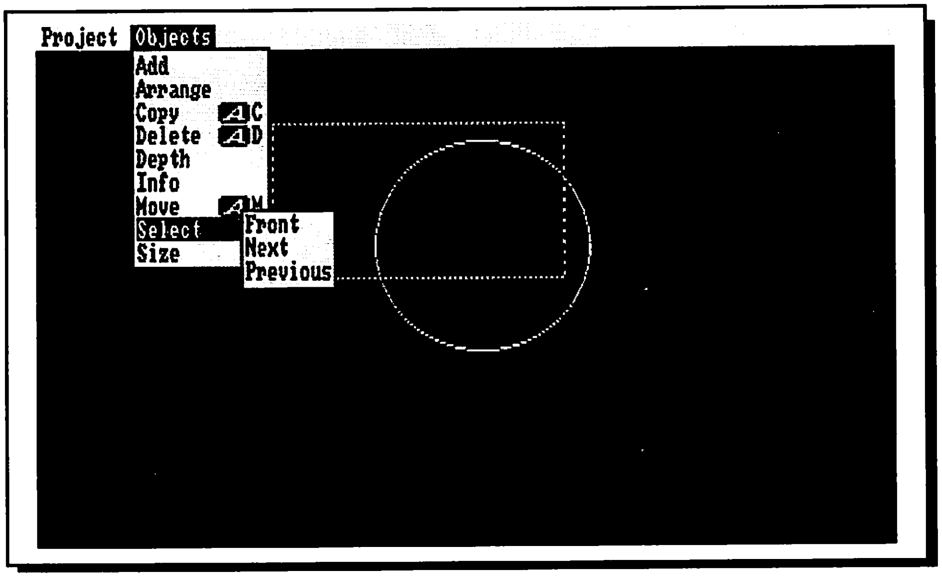


Figure 15-23
What you get when you select the Select submenu

Because so much of what you do with AmigaVision is connected to the Object editor, it is essential that you become proficient in its use. I suggest you experiment with all its features and options, writing small programs that use one or more of them. As you gain experience with the Object editor by writing practice programs, development on your applications projects will proceed much faster and you will encounter fewer problems.

– 16 –

Programming Basics

From the start of this book, I have referred to AmigaVision as a programming language. Of course, it is also an authoring package, but in reality, what is an authoring program but a tool for generating other programs? Because AmigaVision is so exceptionally powerful and easy to use, I can say without reservation that anyone — regardless of whether they have ever written a program before — can create courses and programs with AmigaVision. Many tasks that AmigaVision is suited for can be done almost by intuition — with a little experimentation.

But for all its ease of use, to effectively develop serious applications with AmigaVision, you need to at least be familiar with some of the basic concepts that all programmers must know — regardless of the language used. Armed with this information, you can create more complex and powerful programs that execute faster and are smaller in size.

In this chapter we will take a look at decision, branch, loop, and subroutine commands and program structures. We have already covered the icons themselves and, in many cases, have spent a fair amount of time discussing how they work. Here, we will continue this coverage by taking a closer look at how you can use them more effectively. We will finish the chapter with an examination of variables and a couple of programming tricks that can help you get even more power out of variables than they were designed to allow.

Making Decisions

During the execution of most programs, decisions are made based on the answers to questions. Because of this, you will need to create a variety of decision structures to handle the possible outcomes of these events.

There are two basic types of outcome that can occur in your programs. As a result of a decision, either one or more alternative statements within the decision structure can be executed, or program execution can branch to another part of the program. These events can be based on almost any activity that requires an expression to be evaluated. Examples of decision events that can occur in your programs are:

Did the user click on the Stop button?

Has the user made a menu selection?

Is the current frame of video 12345?

Is the current ANIM frame 47?

Is x equal to 60?

Is A less than B?

Is Name equal to "Terra?"

Is Response ( ) equal to "Sound?"

Is i greater than 10?

During the execution of your program, decisions are evaluated by the If-Then, If-Then-Else, and Conditional Goto commands. Other conditional events can occur in AmigaVision programs with the Wait Condition command and the Error Condition of the Object Editor's Input Field. We will limit this discussion, however, to the If-Then, If-Then-Else, and Conditional Goto commands.

The If-Then command is used to evaluate an expression and then to execute an additional command (its partner icon) if the expression is determined to be true. For example, suppose you wanted to test a variable to find out whether it is equal to or greater than ten, and then set the variable back to zero if it is. Here is the logic presented in a conventional programming manner:

```
IF x >= 10 THEN x=0
```

If you were to translate the above statement into English, it would read as follows:

If *x* is greater than or equal to ten, then the value of x is set to zero

First, the expression is evaluated (is *x* greater than or equal to ten?). Then, if the expression proves to be true (if *x is* equal to or greater than ten), the second part is executed — *x* is set to the new value of zero.

In AmigaVision this decision is made using the If-Then command. Figure 16-1 shows a Flow window with the If-Then command followed by a Variable command. You can see, simply by looking at the flow chart, that a decision structure exists, but except for the comment to the right, you do not know what expression is being evaluated or what will occur if it proves true. The only certainty is that the resulting action will involve variables.

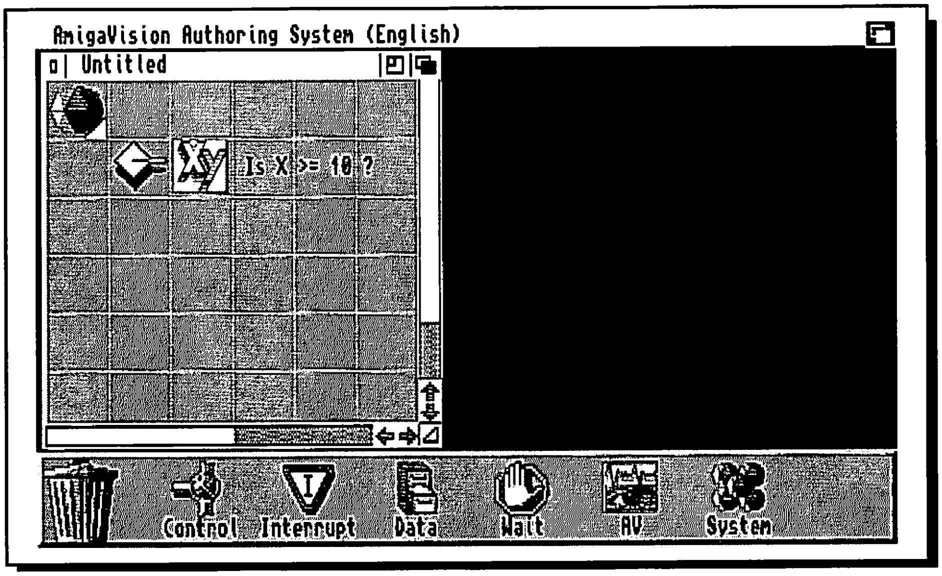
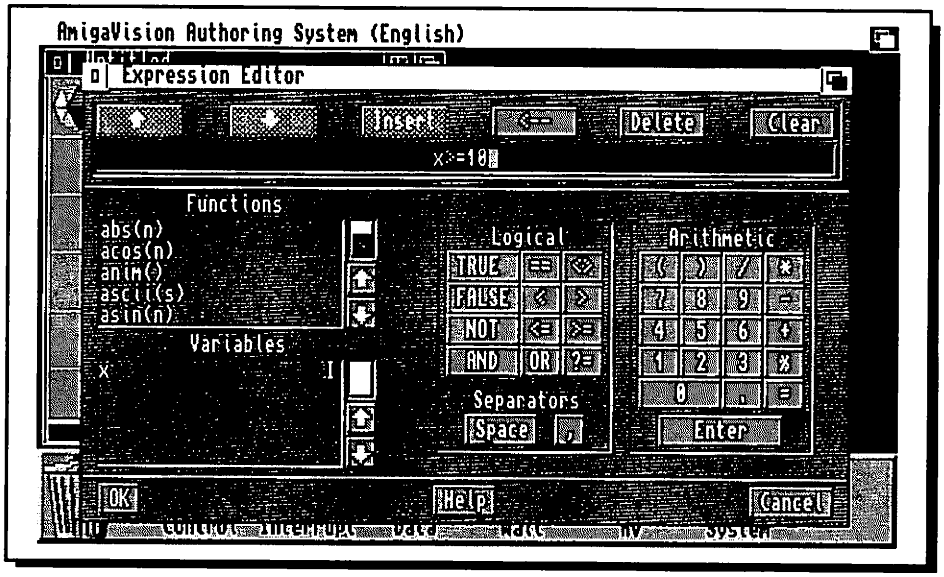


Figure 16-1
An If-Then decision structure where the If-Then command is followed by a Variable command

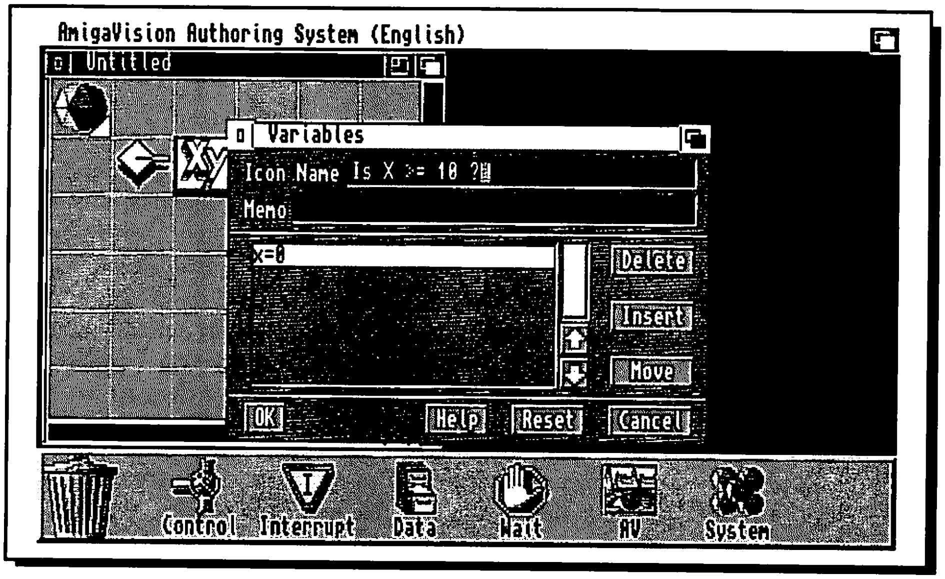
By double-clicking on the If-Then icon, you can open the Expression editor and see exactly what expression is to be evaluated. In Figure 16-2, you can see the expression x>=10 has been entered into the input field. When AmigaVision encounters this If-Then command, it will evaluate this expression. The program will look up the value of the variable *x*, and then check to see if it equals or exceeds ten. If it does not — in other words, if *x* is less than ten and the expression evaluates as false — AmigaVision will continue to the next icon below it. If the expression proves true — that is, if *x* is equal to or greater than ten — AmigaVision will execute the partner of the If-Then icon which, in this case, is a Variable icon.

To see what happens when the expression evaluates as true, double-click on the Variable icon to bring up the Variable requester (see Figure 16-3). Here in its list window, you can see that *x* is assigned the value of zero. After this instruction is carried out, program execution continues to the icon following the If-Then command.



x>=10

Figure 16-2
Examining the expression in the If-Then command

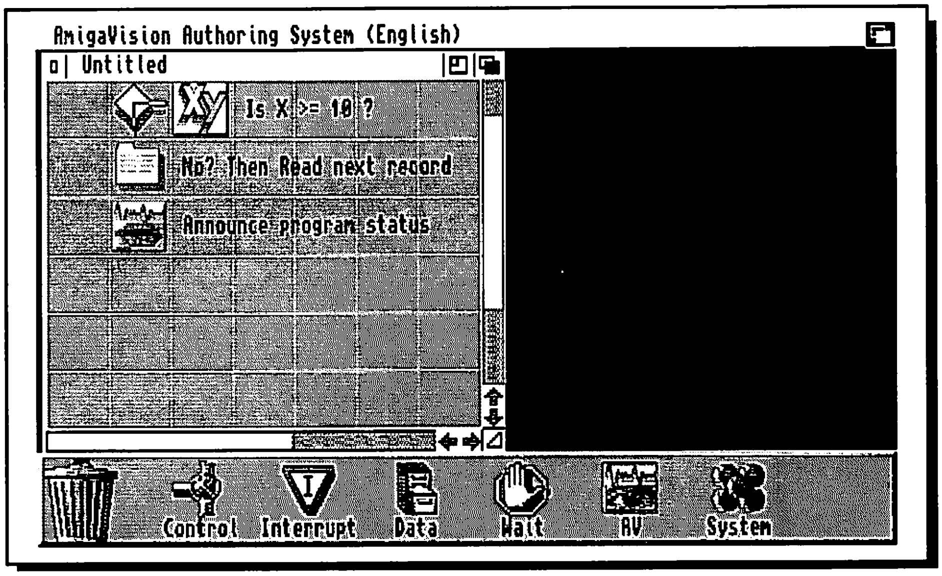


Icon Name Is X >= 10 ?
Memo
x=0

Figure 16-3
Examining the Variable icon acting as a partner to the If-Then icon

As you have seen, the program first evaluates the expression in the If-Then icon. If it proves true the partner is executed, and if it is false the partner is ignored. Either way, program execution continues on to the next icon in the flow.

The If-Then-Else command works almost exactly like the If-Then command. Figure 16-4 shows a simple If-Then-Else decision structure. As with the previous example, this sequence evaluates the expression x>=10. If the expression is determined to be true, the Variable partner icon is executed; if it is false, the partner is skipped.



Is X >= 10 ?
No? Then Read next record
Announce program status

Figure 16-4
An If-Then-Else decision structure

So far, the If-Then-Else example has followed the same path as the If-Then command. However, once the expression is evaluated and the partner is either executed or skipped, the similarity ends. Here, if the expression is true, the partner is executed and the Else part of the If-Then-Else block (the Read/Write Record command) is skipped. Program flow continues to the next icon after the end of the If-Then-Else structure which, in this case, is the Speech command. If the evaluated expression is false, the partner is skipped, the Read/Write Record icon is executed, and program execution moves on to the Speech command.

The If-Then-Else command allows you to execute a single command from two possibilities based on the result of an expression evaluation. This is a very powerful programming concept. If you need to evaluate more than two possibilities in a single decision, one solution is to include multiple If-Then statements in a sequence. For example, suppose you need to test variable *x* for several possibilities and then, based on the results, call different subroutines. Using If-Then commands, this sequence would look like this in conventional code:

```
If x==0 Then Call Subroutine1
If x==1 Then Call Subroutine2
If x==2 Then Call Subroutine3
If x==3 Then Call Subroutine4
If x==4 Then Call Subroutine5
```

In AmigaVision, the same code sequence would consist of several If-Then icons. In Figure 16-5, I used a Module icon as a partner for each If-Then command, and then placed a different subroutine call within each module.

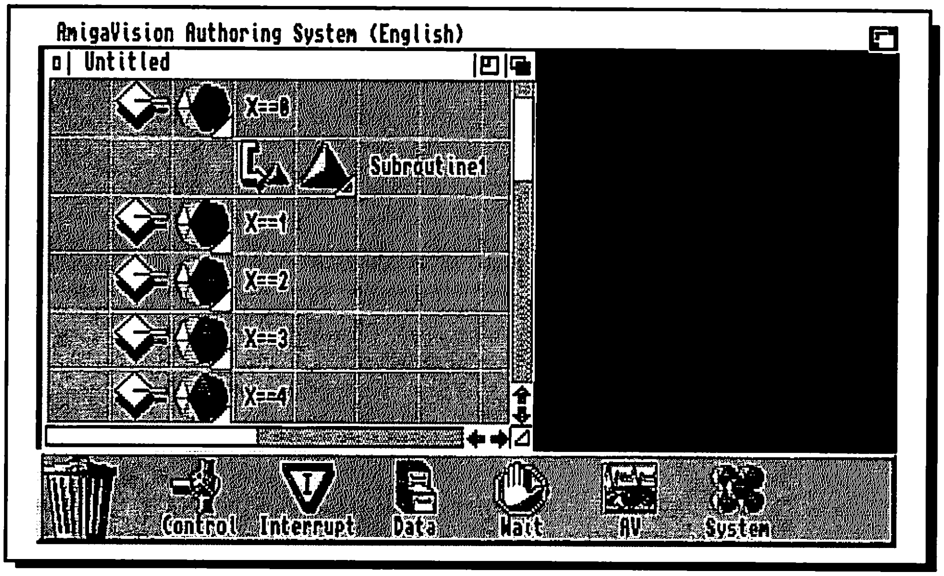


Figure 16-5
A complex If-Then decision structure in AmigaVision

While this sequence works, it is weak in its design. When AmigaVision executes the first If-Then in the sequence, it checks to see if *x* equals zero. If it indeed does, the partner is executed, program execution goes to Subroutine1, and upon completion of the subroutine, AmigaVision jumps to the icon following the original call to the subroutine — in this example, the next If-Then icon in the list. Although the rest of the expressions will evaluate as false, because of the way the If-Then command works, AmigaVision will test all five If-Then expressions before continuing on.

A more efficient approach involves using four If-Then-Else commands and one If-Then. Generally, the result is a faster-executing structure. In conventional programming language, such a sequence would look like this:

```
If x==0 Then Call Subroutine1 Else
If x==1 Then Call Subroutine2 Else
If x==2 Then Call Subroutine3 Else
If x==3 Then Call Subroutine4 Else
If x==4 Then Call Subroutine5
```

Figure 16-6 shows the same decision structure using If-Then-Else icons, a structure that is much more effective than the sequence of If-Then commands. If the first command in the sequence is true, its partner is executed and the rest of the commands in the decision structure are skipped. If this first expression is false, the second is checked. If this one is true, its partner is executed and all following If-Then-Else commands are skipped. If false, the next one is checked, and so on until one is found to be true or the program runs out of possibilities. The efficiency comes from the fact that once a member of the structure is evaluated as true, all the others are skipped, saving your program valuable execution time.

Figure 16-6 uses an If-Then icon as the final member of the decision structure. If I had used all If-Then-Else icons, everything would work as before, except that the icon following the last If-Then-Else would be skipped unless all the preceding decisions had been evaluated as false. Unless I had desired that result and planned for it, the result would be a logic error — one that might be difficult to determine during the test-and-debug phase of program development.

In both of the above examples, I wanted the Call icon to be the companion of the If-Then command. Because a partner-requiring icon cannot have another partner-requiring icon (such as the Call icon) as its companion, however, I used a Module icon as the partner. Then I placed the Call icon and its partner as children of the Module, thus achieving the same effect.

The power of this technique should not be overlooked. For example, suppose you have a menu that offers many possibilities, all of which cause a very different sequence of actions and events to occur when selected. By using the Module icon as a parent, you can add as many actions as you need, all of which will be executed only when the If-Then (or If-Then-Else) evaluates as true.

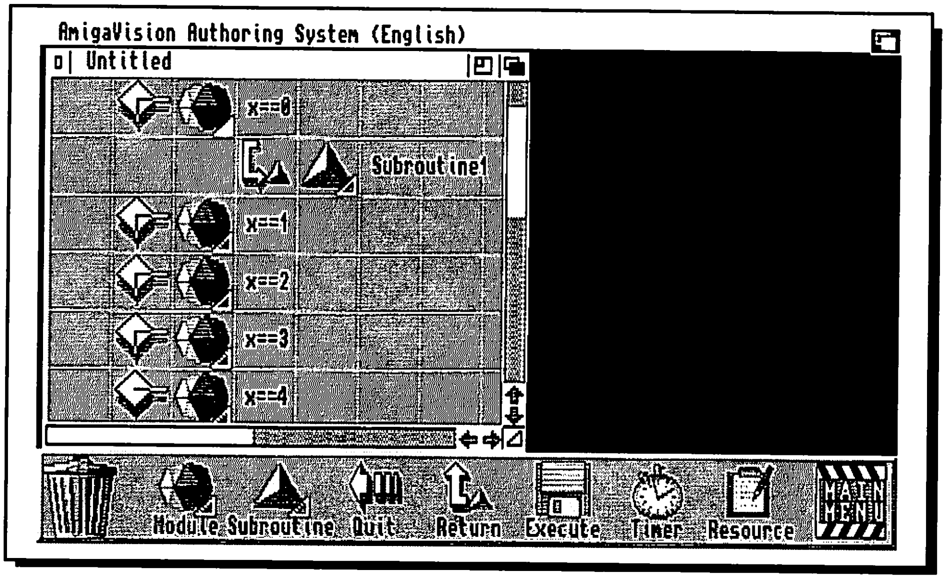


Figure 16-6
Using If-Then-Else and If-Then to make a complex decision structure

Using the Module icon, you can have If-Then structures as children of other If-Then commands. Figure 16-7 shows a series of If-Then icons acting as children of the first If-Then command. In evaluating this complex structure, AmigaVision looks at the parent expression. If *x* does not equal one, then the expression is false and the entire decision structure is skipped. If *x* does equal one, however, then the next icon — another If-Then — is executed. This child command checks to see if *y* is equal to two. If it is not, the decision structure finishes and AmigaVision continues on. However, if the expression proves true (if *y* does equal two) then the third If-Then expression is evaluated. If it is false, the structure ends, but if *z* is equal to three, then the last icon, which is a Variable icon that sets the variable *a* equal to x+y+z (a=x+y+z) is executed. Only if all three If-Then expressions are true will this final command be executed.

You can make the above structure more complex and extensive if you need to. For example, you could perform additional calculations and actions before the If-Then icon using sibling icons. In fact, there is no limit to the level of complexity you can create.

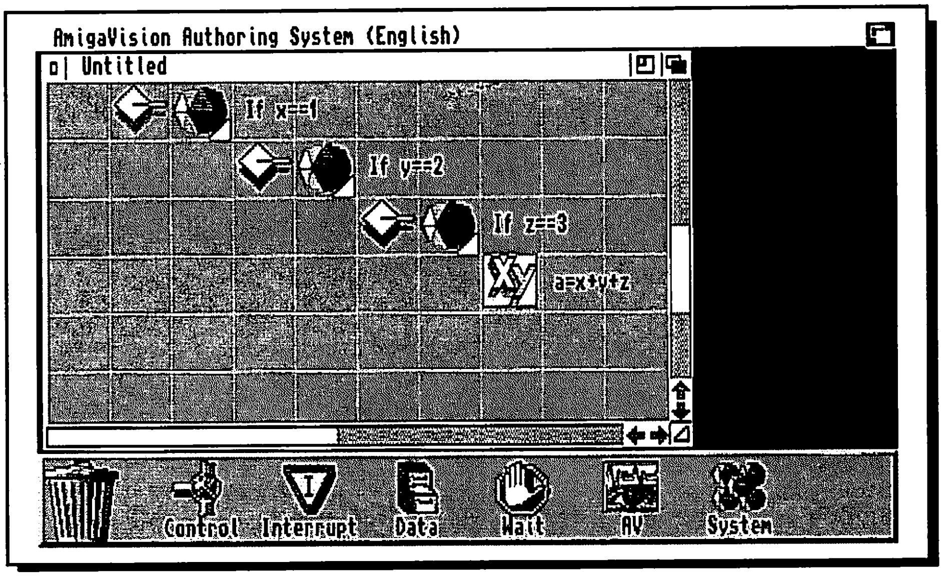


Figure 16-7
If-Then commands as children of other If-Then commands

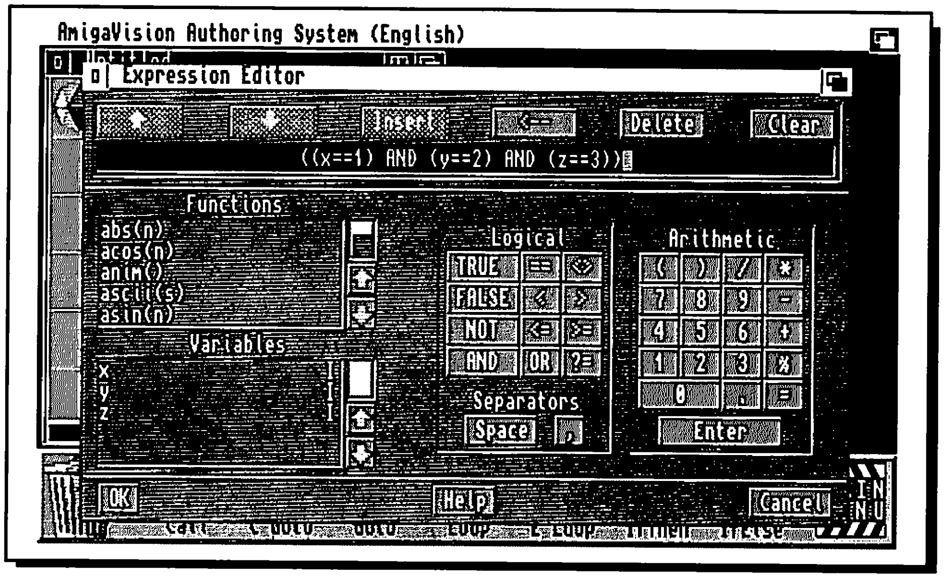
In many cases, you can combine multiple logical expressions. For example, instead of three If-Then commands, you could achieve the same result using one If-Then that contains complex expressions:

IF ((x==1) AND (y==2) AND (z==3)) THEN A=X+Y+Z

Within this command, three expressions are enclosed in parentheses and separated by the logical operator AND. Because of the way AND works, only if all three parenthetical expressions evaluate as true will the entire expression be true. If one or more are false, the entire expression is false. Figure 16-8 shows the AmigaVision Expression editor with the above expression defined in a single icon.

Another command useful in program decisions is the Conditional Goto. This command allows you to change the current execution point of the program to another location in the flow, basing the decision to branch on how an expression is evaluated.

For example, suppose you are writing a course that combines several different multimedia options, allowing the user to select text information, pictures, narration, or even video. If different parts of your program handle each option and an



((x==1) AND (y==2) AND (z==3))

Figure 16-8
Combining several expressions using AND in the Expression editor

iconic menu interface controls user selection, depending on what the user selects, program execution will jump to the section of the program that generates the option. Here is a simple example of such a structure in traditional programming code (note that while the branching directives here are GOTO commands, they are intended to demonstrate how Conditional Goto icons are used):

```
WaitUser:
 UserResponse=="Text" GOTO DisplayText
 UserResponse=="Sound" GOTO PlaySound
 UserResponse=="Picture" GOTO SlideShow
 UserResponse=="Video" GOTO PlayLaser
GOTO WaitUser:
```

In AmigaVision you can create such a program loop using a Wait Mouse icon, four Conditional Gotos, and one unconditional Goto icon, as in Figure 16-9. When the user selects one of the options, program execution jumps to the Module icon that starts that routine. When the routine finishes, control jumps back to the menu loop via another Goto icon.

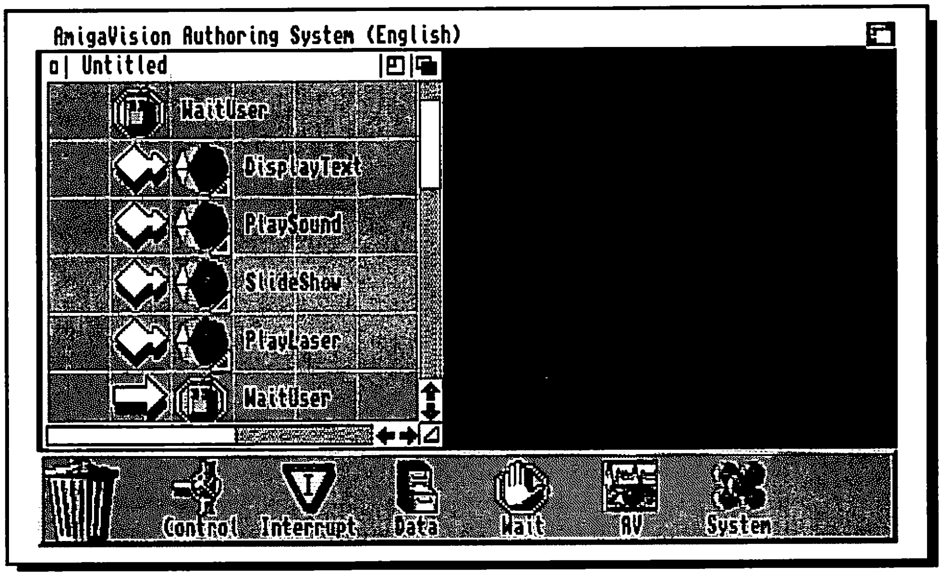


Figure 16-9
Using Conditional Goto to create a program loop

Please keep in mind when you are using Conditional Goto or Goto that these commands are very easily overused, especially by new programmers. When overuse occurs, the underlying logic of the program becomes very difficult to follow and mistakes and bugs become difficult to trace. This is not to say that you should avoid using these commands in your programs, but rather that you should use them when you cannot find another way to accomplish what you need to. In Figure 16-9, for example, it would probably be better to use subroutines accessed via a Call instead of using a Conditional Goto.

Loops

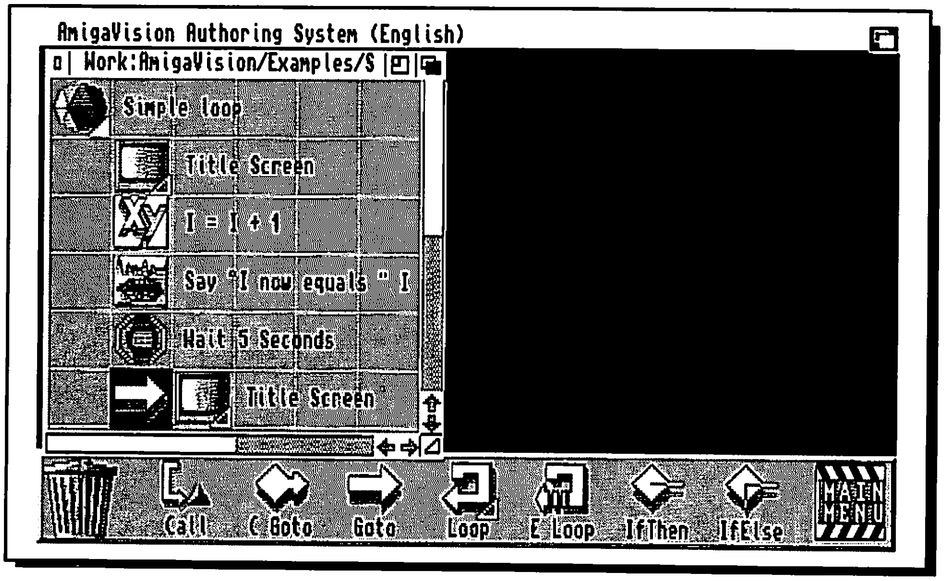
A loop is a section of code that is repeated one or more times. A useful spot for a Loop command is in a program's main menu. Each time the user selects a menu option, the program executes a subroutine via a Call command. When the subroutine is finished, execution returns to the area of the program that contains the menu, and loops through this section until a subroutine is needed.

The simplest form of a loop can be accomplished with an unconditional Goto command. The program loop is bounded by the Goto icon, which jumps back to an earlier command. Here is an example using traditional programming commands:

```
Main menu:
        Display screen
        Play music
        User selects
Goto Main menu
```

An AmigaVision program using this structure will work the same as the example above. A sequence of icons ending in a Goto icon will cause program execution to jump back to the main menu section.

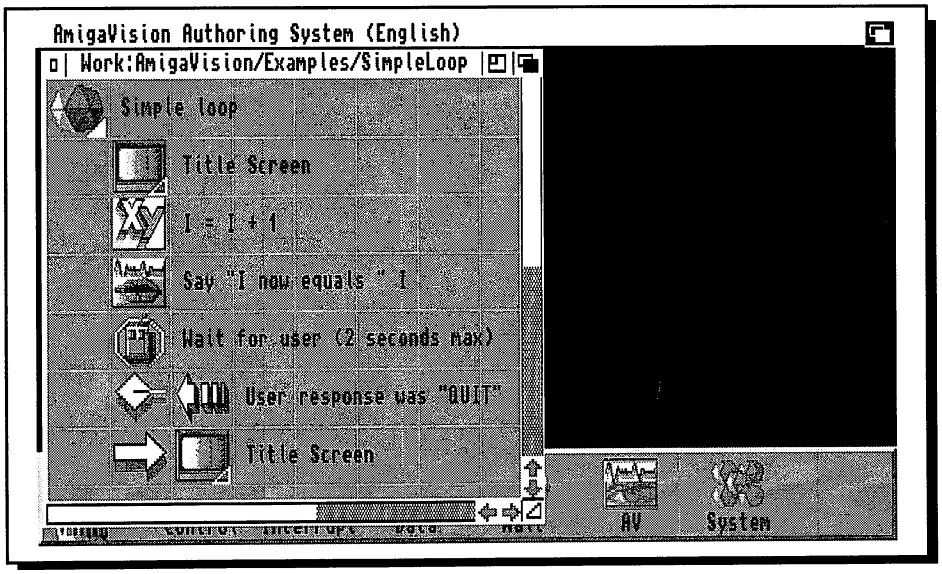
Figure 16-10 shows a simple loop using the Goto icon. The program displays a title screen, adds incrementally to the variable *i*, uses the speech command to "speak" the value of *i*, pauses a couple of seconds, and then jumps back to the title screen and starts again. This loop will repeat over and over again, and everything will stay the same except the value of *i*, which will increase.


AmigaVision Authoring System (English)
Work:AmigaVision/Examples/S
Call
C Goto
Goto
Loop
E Loop
IfThen
IfElse
MAIN MENU

```
Simple loop
    Title Screen
    I = I + 1
    Say "I now equals " I
    Wait 5 Seconds
    Title Screen
```

Figure 16-10
A simple AmigaVision loop using Goto

This program has a problem, however. There is no provision to exit the loop, and unless you had started it from the AmigaVision program editor, you would not be able to exit the loop without breaking out of the program or possibly even resetting your computer. The golden rule in setting up loops is to always allow some sort of exit. Figure 16-11 shows the previous loop modified to include an exit option. The time-delay icon has been replaced with a Wait Mouse icon that adds a Quit button (which was created in the Object editor) to the title screen. An If-Then icon checks for a user response of "QUIT" and, with an Exit icon as its partner, shuts down the application if the user has clicked on Quit.



AmigaVision Authoring System (English)

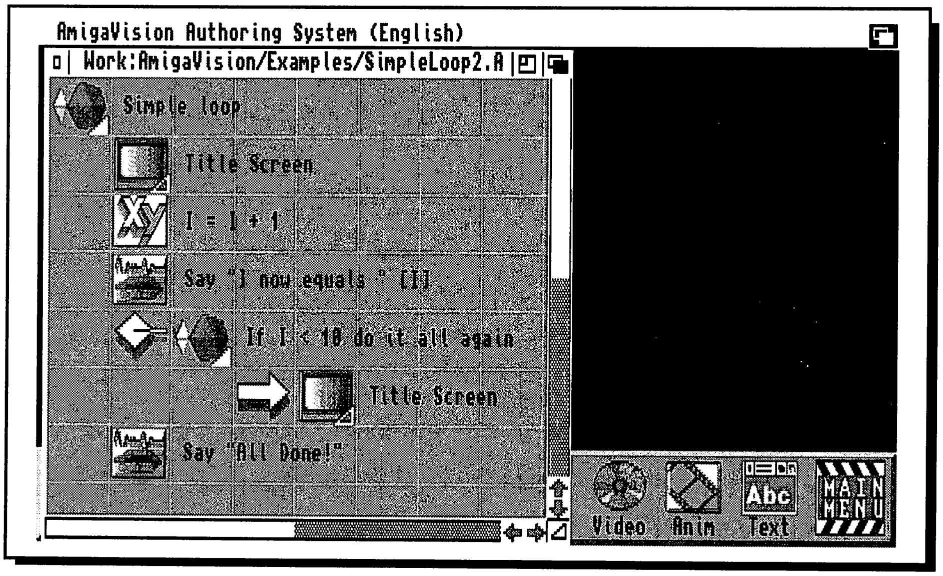
Work:AmigaVision/Examples/SimpleLoop

```
Simple loop
    Title Screen
    I = I + 1
    Say "I now equals " I
    Wait for user (2 seconds max)
    User response was "QUIT"
    Title Screen
```

Figure 16-11
A simple loop with an exit

Another way of defining an exit for a loop is to use a conditional statement that checks to make sure an expression is true before continuing. If the expression is false, the loop ends. In Figure 16-12, the loop will continue until the variable *i* reaches ten, at which point program execution will continue past the conditional If-Then command.

You could replace the If-Then with a Conditional Goto, and specify that program branching occur only if *i* is less than ten. The advantage of using If-Then is that it allows you to execute additional commands before jumping back to the title screen.



AmigaVision Authoring System (English)

Work:AmigaVision/Examples/SimpleLoop2.A

```
Simple loop
    Title Screen
    I = I + 1
    Say "I now equals " [I]
    If I < 10 do it all again
        Title Screen
    Say "All Done!"
```

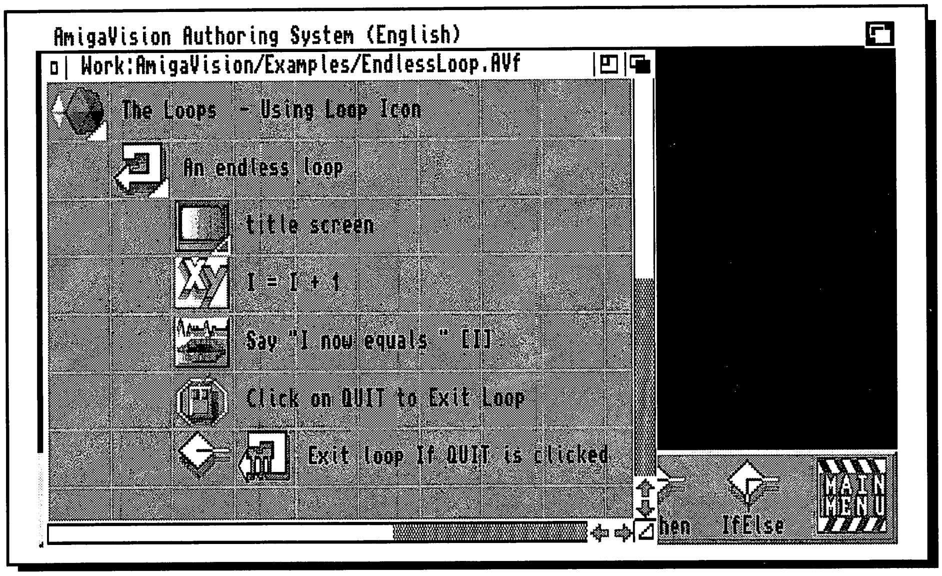
Figure 16-12
Limiting the duration of a loop based on the value of a variable

To make using loops easier, AmigaVision contains a Loop command that incorporates three modes: Conditional, Counted, and Endless. The Loop icon itself is a parent, and will play through its children over and over until something forces it to end. This "something" can be the satisfaction of a condition for exit, the attainment of the end of a counted loop, arrival at an Exit Loop icon, or merely a jump out of the loop via a Goto or Conditional Goto command.

An Endless Loop has no parameters. Like the example in Figure 16-10 that uses a Goto without any form of exit, the Endless Loop will go on forever. To select the type of loop you want, double-click on the Loop icon. When the Loop requester appears, click on the multistate gadget below the Memo field until it displays your choice of modes: Endless, Counted, or Conditional. Figure 16-13 is an example of an Endless Loop: The program will run through all its children over and over until the user exits the loop, ends the program, or resets the computer.

A more traditional form of loop is the Counted Loop. This is essentially the same as the For..Next loop found in BASIC or C. It works by stepping through a defined sequence of numbers by a specified increment. Here is an example of a counted loop using programming code.

```
For i = 1 to 100 Step 1
        Say i
Next i
```


AmigaVision Authoring System (English)
Work:AmigaVision/Examples/EndlessLoop.AVf

The Loops - Using Loop Icon
An endless loop
title screen
I = I + 1
Say "I now equals " [I]
Click on QUIT to Exit Loop
Exit loop If QUIT is clicked

Figure 16-13
An Endless Loop

This example loop will repeat 100 times, counting from one to 100 by ones. Each time through this loop, the program will "speak" the value of the variable *i*, which is linked to the current value of the loop counter.

To create the same counted loop in AmigaVision, double-click on the Loop icon to open the Loop requester. Then click on the Loop Mode multistate gadget until it reads Counted. In this mode the four command gadgets below, labeled Start, Stop, Step, and Var become active. These parameters control the starting and stopping values of the loop, the increment size of the counter, and which variable (if any) to link to the loop counter (Figure 16-14).

In Figure 16-14, the loop counter will start at one, increase by one each pass through the loop, and repeatedly execute the child icons until the counter reaches 100. Because a variable (*i*) has been linked to the loop via the Var gadget, the value of that variable will be set equal to the counter during each repeat sequence. The first time through the loop, *i* will be equal to one, the next time two, the next three, and the last time it will be equal to 100. In many cases, command icons in the loop will use that variable for some purpose.

The loop does not have to start at one or end at 100. Legal values are any positive or negative numbers. And the step value can be any positive or negative value. However, if you specify these parameters using constants (numbers) entered via

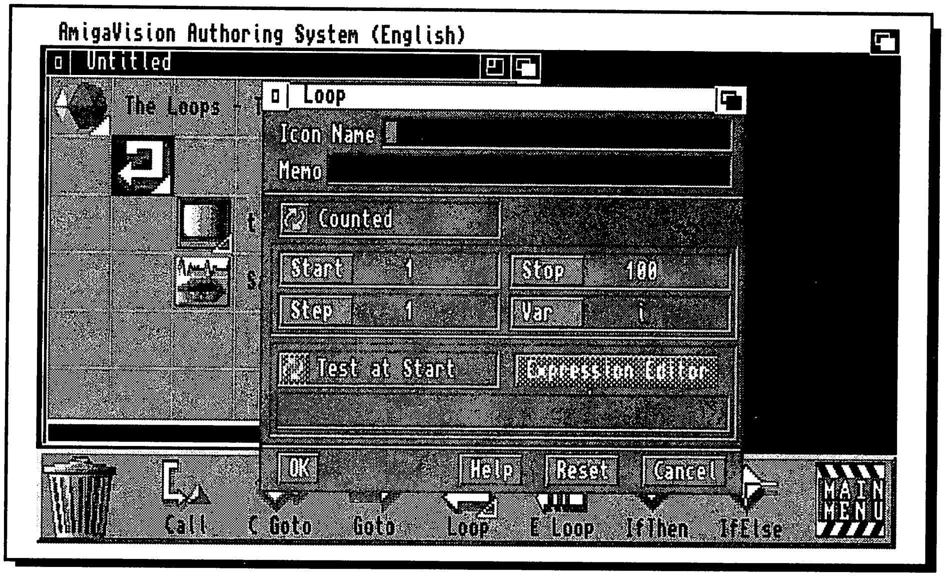


Figure 16-14
A defined counted loop

the Specify Value requester, they are limited to integers (whole numbers). It is also legal to use variables for the Start, Stop, and Step parameters, and these can be floating-point numbers. Here, in conventional code, are some sample counted loops:

```
for i = 1 to 1000 step 5
for j = -100.25 to 100.50 step .25
for k = 100 to -100 step -x
for l = a to b step c
```

An even more powerful form of loop is the Conditional Loop, which does not use such parameters as Start, Stop, or Step, but instead uses an expression to determine whether it should continue or exit. Figure 16-15 shows a defined Loop requester in Conditional Loop mode. Notice that there is a command button labeled Expression Editor at the lower right of the requester. Clicking on this opens the Expression editor, wherein you can enter an expression to be used as the condition for exit. When you exit the editor, the expression you entered will be displayed in the Loop requester, in the field just below the Expression Editor button. In Figure 16-15 the expression is (i+j) < (k-l), and as long as this expression remains true the loop will continue. If this expression proves untrue (in other words, if i+j <> k-l), the loop will automatically exit.

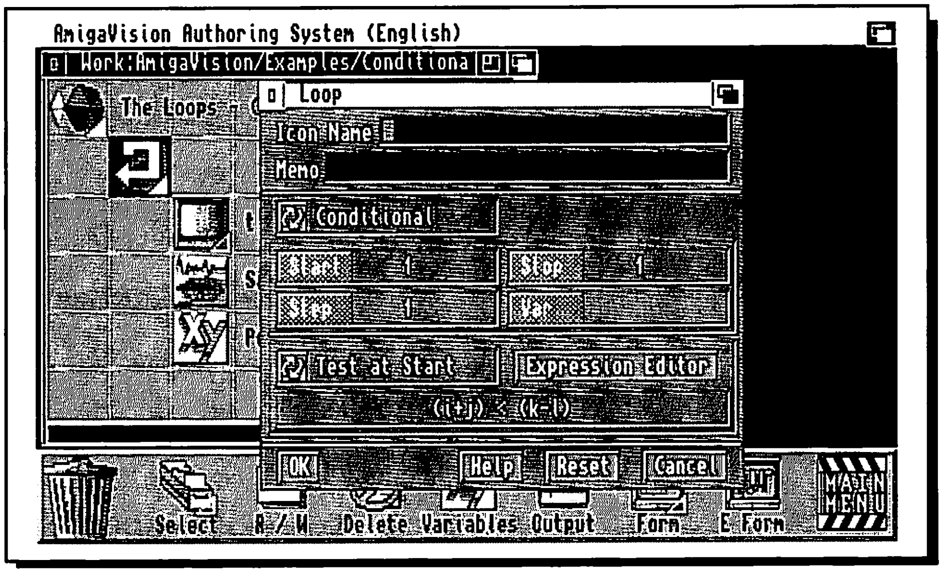


Figure 16-15
The conditional loop requester

The conditional expression used for exit can be almost anything. It can be based on some numeric function or variable (as defined in Figure 16-15), the value of a string variable (name =="John"), the current frame of animation or video (frame == 60), a time duration (seconds <= 15), or a user response (Response =="Play Sound"). However, unless there is some compelling reason not to, you should provide some way out of the loop besides this conditional exit.

On the Loop requester is a multistate gadget that is available only when the loop is in Conditional mode. This gadget toggles between Test at End and Test at Start. These modes determine where in the loop the conditional expression is evaluated, and can have a significant impact on how your program works.

Test at Start evaluates the expression at the beginning of the loop. If the expression is found to be true, the loop will continue; if it is false, it will exit the loop and program execution will proceed to the sibling beneath the Loop icon. If the expression evaluates as false the first time through the loop, none of the children of the loop will be executed even once. So, when using Test at Start mode, you should be aware that there is always a possibility that the children of the loop will never be encountered.

Choosing Test at End instructs AmigaVision to evaluate the conditional expression at the end of the loop. This means that it will evaluate the expression only after it executes all the children of the loop. In this situation, the loop will always execute at least once, regardless of the value of the expression when the loop is encountered. Which of these modes you use depends on the needs of your program, but it is essential you understand the difference.

Because loops can be nested — meaning that a Loop icon can be the child of another Loop icon—small loops can quickly become large and time-consuming to execute. Nesting is a powerful programming option, but you should take care when using them. Consider a triple-nested counted loop that is set to execute 100 times. The total number of times the outer loop will execute is 100, but the middle loop will execute 100*100, or 10,000 times, and the inner loop will execute 100 times that, or one million times! Use nested loops with care, as the number of passes can quickly become very large, even with relatively small numbers for each loop.

Modules and Subroutines

AmigaVision is a fairly structured language, and one of the best methods of creating structured programs is to break the program down into small modules or subroutines that can be more easily defined and implemented. The main program then often becomes essentially a series of calls to these subroutines. This type of structure can be considerably easier to deal with than other types when problems crop up, because the problems will likely be confined to one of the subroutines. Further, many subroutines will be generic enough to be useful in other programs, saving you even more development time. In fact, after working with AmigaVision for a while, you may find yourself with a library of routines that you can just drag into your new programs and modify as needed.

AmigaVision has two icons that are used to define subsections of your program. These are the Module and Subroutine icons.

The Module icon is very useful for organizing your program. As a Parent icon it can contain a large number of commands, including other Module icons. I use it in many programs as a method of separating different sections of my code into logical or functional groups. Figure 16-16 shows an example of using the Module icon to isolate various functions of your program. All of the Module icons in this illustration actually contain many child icons, but these are kept out of sight when not needed by telescoping them into their Module icon.

The Module command also allows you to create variables in the same manner as the Variable icon. However, variables created by the Module command icon are local variables, meaning they are available to only the children of that particular Module icon. Contrast that with the global variables created with the Variable icon which are always present anywhere in the program.

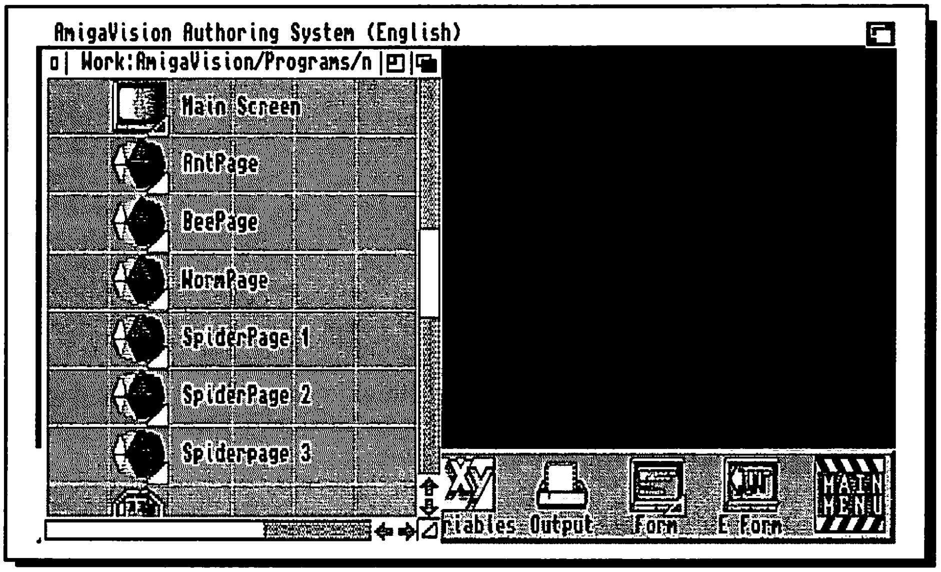


Figure 16-16
Using Modules to organize your programs

Local variables are initialized when the Module (and its children) are executed. When the group of icons on the Module complete and program control continues with the next sibling of the Module, these local variables lose their definitions and value. When and if the Module is reexecuted they are all created fresh again. Remember — local variables are reinitialized when the Module or Subroutine icon with which they were created are executed again.

Modules are essential when you want to create more complex functions for such decision command icons as If-Then and If-Then-Else. Since these commands are allowed only one companion, using the Module icon as that companion allows you to add a multitude of "partners" to the decision command — as children of the Module.

While Module icons are actually part of the main program, Subroutine icons (and their children) are separate entities. These icons are, or at least should be, accessed only by the explicit action of a Call icon.

Subroutines are just what their name implies — routines subordinate to the main program. The most efficient way to use routines that will be needed several times throughout your program is to set them up as subroutines, so you can call them over and over.

The Subroutine icon, like the Module icon, can be used to create local variables. Also like the Module icon, the children of Subroutine icons have access to their own local variables as well as the global variables of the main program. Remember, if you want your variables to be local, create them with the Module or Subroutine icons. Variables created with the Variable icon are global, regardless of where in the program they are created!

The types of subroutines you will create will be as varied as your programs. However, some types of functions are fairly common and readily adapt themselves as subroutines. Figure 16-17 shows an example of just such routines. At the bottom of the flow chart are three subroutines designed to control a laser-disc player. The subroutines are labeled Show One Frame, Show N Frames, and Play Sequence. These routines were created for a multimedia course I developed that was based on a laser disc containing many images and video segments of the NASA Voyager spacecraft and their planetary encounters. Because there were so many different image possibilities, these three routines were created to handle just about any contingency I might need.

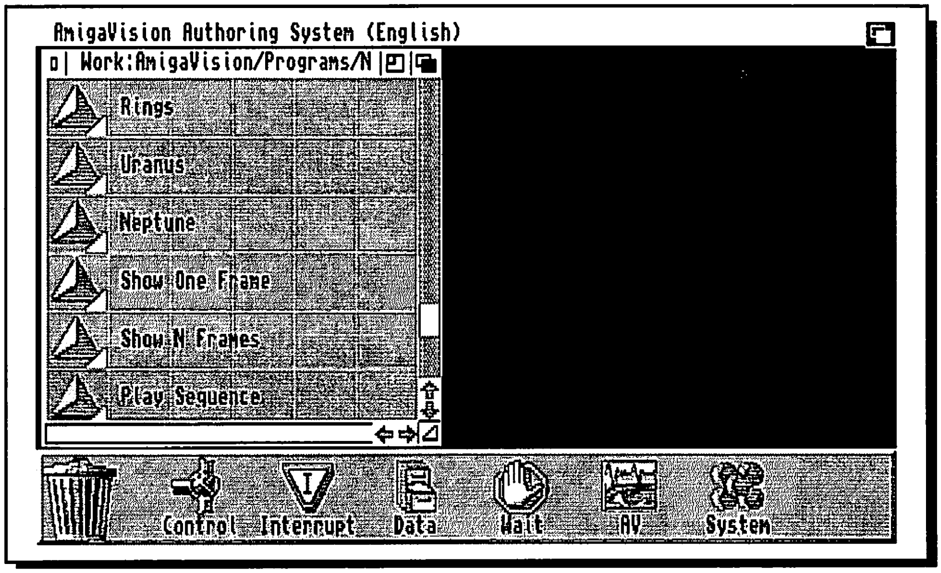
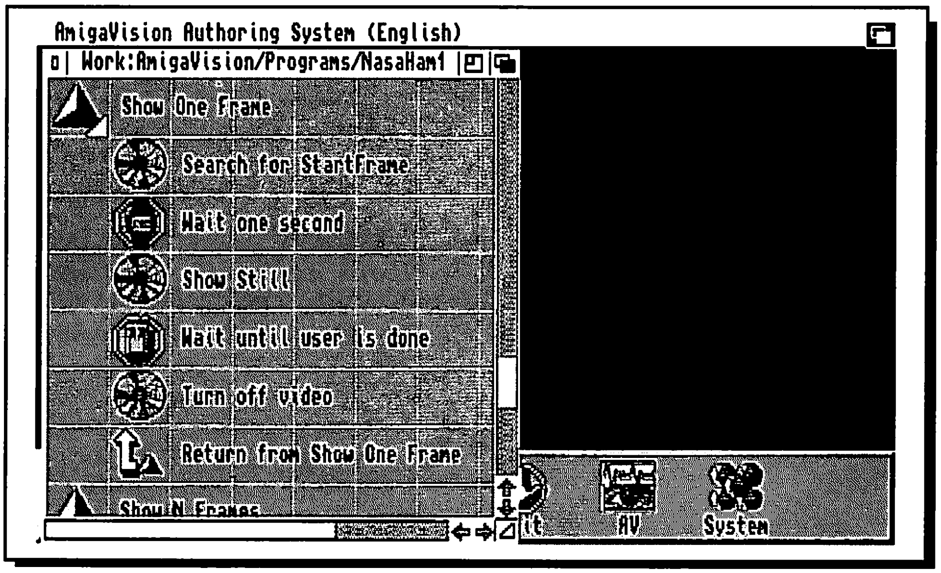


Figure 16-17
A set of subroutines for laser-disc control

Show One Frame (Figure 16-18) is called with a global variable called StartFrame defined with the frame number of video I want to display. The routine first uses the variable StartFrame as the parameter for a search. Then, when the frame is found, the laser-disc player is set to Still mode. The video image remains on screen until the user clicks an Exit button with the mouse, then the video signal from the laser-disc player is turned off and the routine returns to the main program.



AmigaVision Authoring System (English)

Work:AmigaVision/Programs/NasaHam1

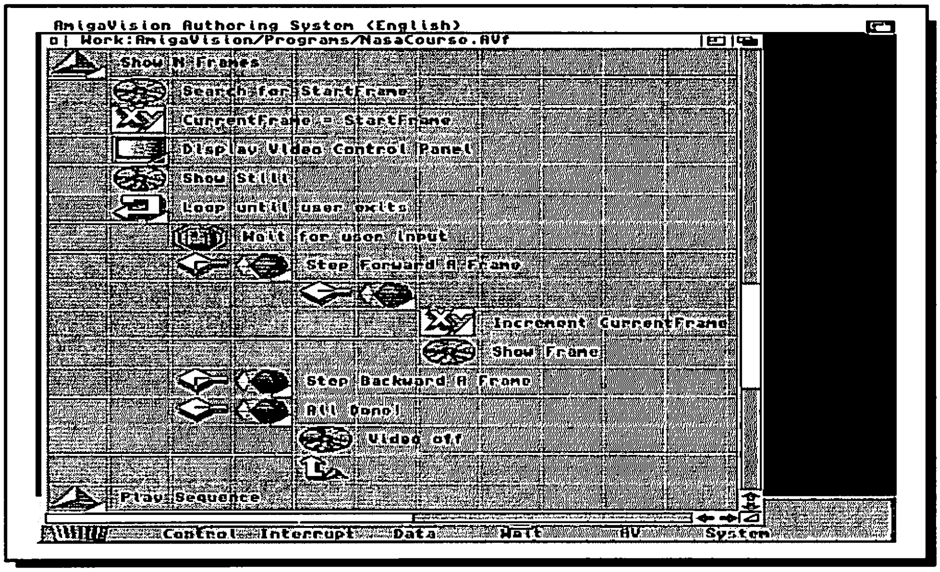
```
Show One Frame
    Search for StartFrame
    Wait one second
    Show Still
    Wait until user is done
    Turn off video
    Return from Show One Frame
Show N Frames
```

Figure 16-18
The Show One Frame laser-disc subroutine

Show One Frame is very useful when you need to show a single frame of video, but there are often times when you need to show a sequence of frames. An example from the NASA course is the moons of Jupiter. Each moon has a number of still images in a sequence that you can step through. Show One Frame could be used for this task if the variable StartFrame was initialized for each required frame and the subroutine called repeatedly. But that is a cumbersome method.

I wanted a more flexible routine capable of accepting two parameters, StartFrame and EndFrame, that could define a sequence of frames I wanted the user to be able to step through. Because I wanted my program to be an image browser, I needed the routine to be able to step backward in the sequence if the user decided to look again at a previous image. The routine also needed to be smart enough to limit itself to only those images in the sequence.

With these requirements in mind, I created Show N Frames (Figure 16-19). The two parameters, StartFrame and EndFrame, are global parameters that are defined by the main program. On entry, the video disc is moved to the location on the disc of StartFrame. A temporary variable called CurrentFrame is initialized to the value of StartFrame; all subsequent functions use CurrentFrame and not the global variables. The reason for this is that the value will change as the user steps forward and backward through the sequence, and I did not want to alter the value of global variables that might be needed in the main program.



```
AmigaVision Authoring System (English)
Work:AmigaVision/Programs/NasaCourse.AVf
Show N Frames
    Search for StartFrame
    CurrentFrame = StartFrame
    Display Video Control Panel
    Show Still
    Loop until user exits
        Wait for user input
            Step Forward A Frame
                Increment CurrentFrame
                Show Frame
            Step Backward A Frame
            All Done!
                Video off
Play Sequence
Control  Interrupt  Data  Wait  AV  System
```

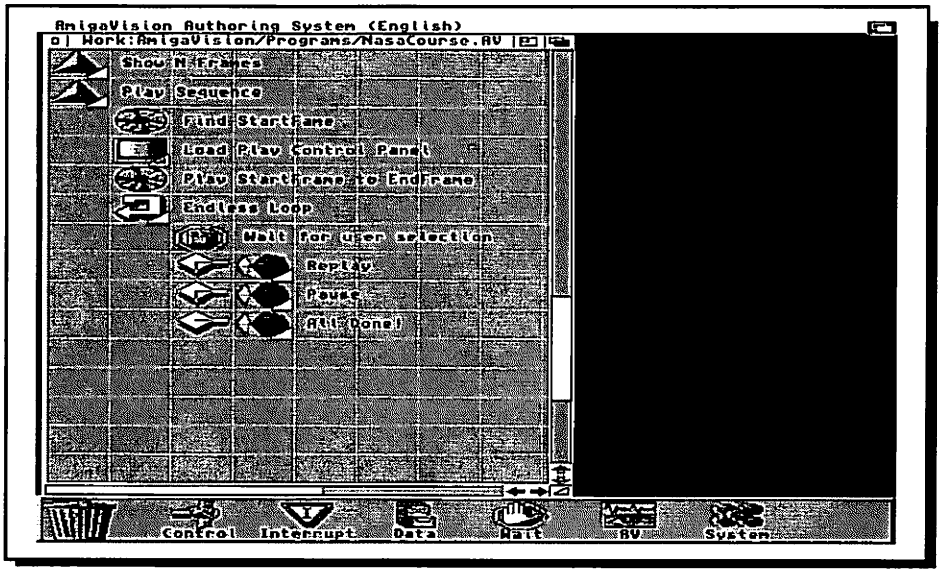
Figure 16-19
The Show N Frames subroutine

At this point in the routine, a computer image is displayed at the bottom of the screen over the video image. This image is a simple control panel with a series of buttons labeled StepForward, StepBackward, and AllDone. (Hot spots with the appropriate response strings are linked to the buttons so the user commands can be carried out.)

Once the image is displayed, an endless loop is executed and a Wait Mouse monitors the screen, waiting for one of the hot spots to be clicked. In response to a click, the laser disc is stepped forward or backward through the sequence. Each of the possibilities (StepForward, StepBackward, and AllDone) are handled by a series of If-Then-Else commands. With the step forward and backward options, additional checks are made inside the Module to make sure the user is not at the end or beginning of the sequence. For example, if the user is at the last frame in the sequence (EndFrame), no additional frames are available and a StepForward command would be ignored. Likewise, if the user is at the first frame of the sequence, a StepBackward command would be ignored. These decisions are made by comparing the value of CurrentFrame with the beginning and ending frame numbers of the sequence. After each successful StepForward or StepBackward, CurrentFrame is increased or decreased so that it always shows the proper position within the defined sequence.

The way out of this endless loop is the AllDone button. When the program detects that the the user has clicked on this button, the video image shuts off and a Return icon is executed, returning program execution to the main program — at the point immediately following the original call to the Show N Frames subroutine.

Finally, to round out this collection of laser-disc subroutines, I needed one to play a video sequence. This function is handled by the Play Sequence subroutine (Figure 16-20). Here, too, the global variables StartFrame and EndFrame are initialized by the main program, and then are used to control the laser disc in the subroutine. Like Show N Frames, the Play Sequence routine has a control panel that allows the user to replay the sequence, pause (freeze) the image so a frame can be examined, and an exit option to get the user out of the subroutine and back to the main program.


AmigaVision Authoring System (English)
Work:AmigaVision/Programs/NasaCourse.AV
Control
Interrupt
Data
Wait
AV
System

```
Show N Frames
Play Sequence
    Find StartFrame
    Load Play Control Panel
    Play StartFrame to EndFrame
    Endless Loop
        Wait for user selection
            Replay
            Pause
            All Done!
```

Figure 16-20
The Play Sequence subroutine lets you play a video sequence

One feature of subroutines not mentioned in your AmigaVision manual is the fact that they are recursive. This means that a subroutine can call not just other subroutines but can call itself as well! While this is a very powerful programming feature, it is one you should use with care as it is very easy to get stuck in a recursive subroutine!

Variables

Before wrapping up this chapter, I want to talk a bit about variables in AmigaVision. Most of this has been covered in other sections of the book, but is worth repeating here.

AmigaVision supports a number of variable types — string, floating point, Boolean, and date. We have seen how to use these in a number of ways, but there are still a few tricks that you can do that might not be quite obvious.

For example, to display a picture, you must enter its name in the Filename field of the Screen icon. However, it is possible to use a variable name instead of an explicit name in this field. Figure 16-21 shows a Screen requester that uses a variable instead of an explicit image name in the Filename field.

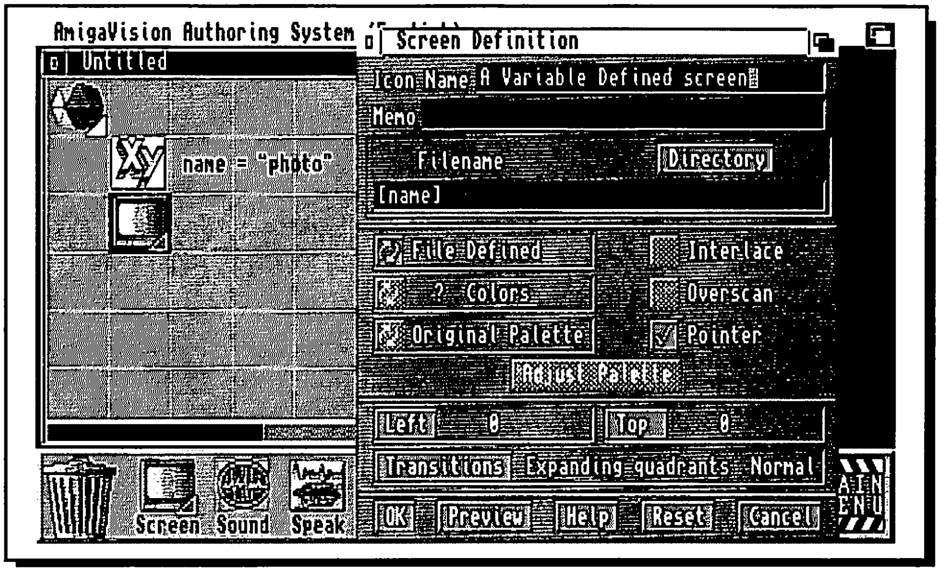
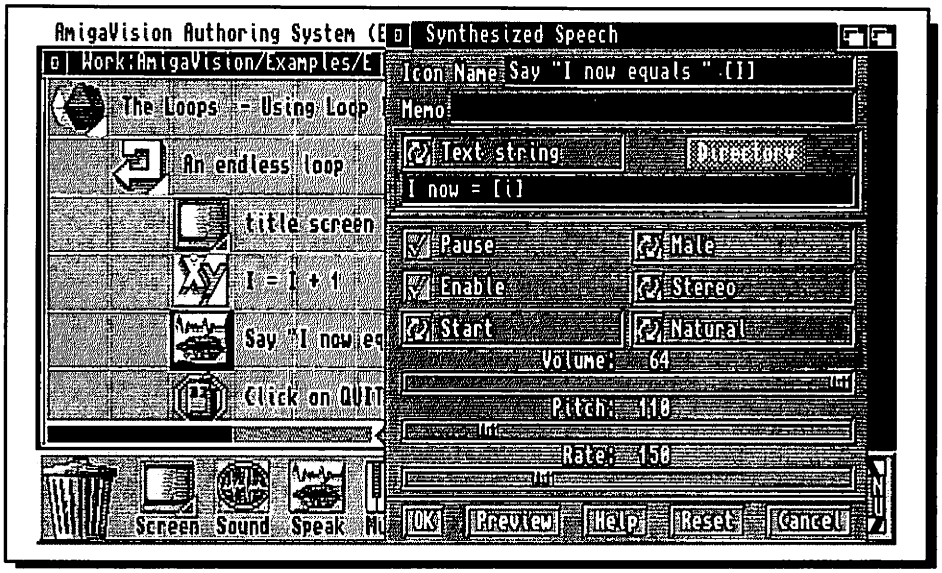

AmigaVision Authoring System
Untitled
name = "photo"
Screen Definition
Icon Name A Variable Defined screen
Memo
Filename
[name]
File Defined
? Colors
Original Palette
Interlace
Overscan
Pointer
Left 0
Top 0
Transitions Expanding quadrants Normal

Figure 16-21
Using a variable for an image file name

Notice that the variable name is enclosed in square brackets. These brackets tell AmigaVision to use the variable inside the brackets and not to use the Filename entry directly. If you are going to use this technique, make sure you set the multi-state gadget that controls resolution to File Defined mode.

One drawback to using variables in Filename fields is that in relocating applications, AmigaVision does not automatically move the variable-defined files with the rest of the application — unless you have defined them in a Resource icon.

You can use variables in the Filename fields of other icons as well. The Speech command for example is well suited to using variables. Figure 16-22 shows a Speech icon that uses both a constant text phrase "I now =" as well as a variable [*i*]. The result is that whenever this icon is executed, the Amiga will say "I now equals 1," or "I now equals 2," and so on, with the spoken value of *i* changing as the value of *i* changes in the program.



Icon Name: Say "I now equals " [I]

Memo:

Text string

I now = [i]

✓ Pause | Male

✓ Enable | Stereo

Start | Natural

Volume: 64

Pitch: 110

Rate: 150

Figure 16-22
Using a variable in the Speak icon

One weakness of the current version of AmigaVision is its lack of array variables. Arrays are groups of related variables that can be accessed in a sequential manner. For example, consider the following array of names.

Name(1) = "Lou"
Name(2) = "Sharon"

```
Name(3) = "Terra"
Name(4) = "James"
Name(5) = "David"
Name(6) = "Lisa"
```

This is an array of six elements. When we want to use one of these variables, we can reference it by its index number. Name(5) is equal to "David," while Name(2) is equal to "Sharon." Because the index numbers can be variables, arrays are often used in programs to have a logical method of accessing sequential information. For example, if i = 4, then Name(*i*) is equal to "James."

Since AmigaVision does not support arrays, programmers have resorted to using a database file to store information in an array-like manner. For example, you can create a series of records containing name and record-number fields. Then, when you want to use the 56th name in your array, you can select a database record that has the value 56 in its record-number field. By selecting this record you also have access to the other field in this record, so you can thus read and use the contents of the name field as well.

Because you cannot create a database "on the fly," this technique has some serious limitations. Consider the situation where you need to have an array of ten data elements that range in length from one to ten characters. With arrays this would be a breeze, but since we don't have them we have to be a bit more creative.

One solution I came up with is to create pseudo arrays using string variables. Because a string can be up to 255 characters long, and because AmigaVision provides a full complement of string-manipulation commands, it is possible to use a string to store a small array. To begin with, you must define a string with a null (empty) value:

```
Array = ""
```

Then you can add your array elements which, in this case, is a list of names. You would allocate a full ten characters for each name in order to make each element's starting position within the array a multiple of ten. To do this, you simply add enough spaces at the end of each name to make up the difference. (Of course, if any name is longer than ten characters, you would either have to chop off the extra letters or increase the size allowed for each element in the array.) From within a Variable icon, you would add the following ten expressions:

```
Array = Array + "Lou       "
Array = Array + "Sharon    "
Array = Array + "Terra     "
Array = Array + "James     "
Array = Array + "David     "
Array = Array + "Lisa      "
Array = Array + "Captain   "
Array = Array + "Data      "
Array = Array + "Hamster   "
Array = Array + "Goldfish  "
```

Notice that the length of the variable Array will increase by exactly ten characters with each assignment. In the end we will have a string exactly 100 characters long, containing the names of six people, two dogs, a hamster, and a goldfish.

If you were to examine this string, you would see that the word "Lou" begins at character one of the string, "Sharon" begins at the 11th character, "Terra" at the 21st, and so on until we get to "Goldfish," which begins on character 91.

If you think of these names as comprising an array that starts at element 0 (Array(0) = "Lou ") and ends at element 9 (Array(9) = "Goldfish "), a simple formula for accessing them begins to appear. Array element 0 begins at character 0*10+1, element one begins at character 1*10+1, element two at character 2*10+1, and so on until element nine, which starts at 9*10+1.

Since you know how long each element is (ten characters maximum) and how to find them (i*10+1), you can now use string functions to extract your array elements. The command you can use to extract the elements from your array is substr() .

Substr(), you will remember from our discussion of the Expression editor and its functions, allows you to extract a substring from a larger string. The command and its parameters are substr(source string, starting position, length). For example, if you wanted to extract the fifth element of your array and assign it to the variable Name, you could use the following expression:

```
Name = substr(Array,5*10+1,10)
```

This would give Name the value of the ten-character sequence starting at the 51st character in our array, which is the string "David ".

If you want to know if an array contains a specific string, you could use the function findstr(). This command will search a string for the specified substring, starting at a defined position. If it finds that substring, it returns the character number the string starts at. If it does not find it, it returns a zero.

Suppose you want to know if this array contains the string "Captain". Using findstr() to search the string named Array starting at the first character, we can find out if this name is in our string and, if so, where it starts.

```
Element = findstr("Captain",Array,1)
```

In this example, the variable Element would be assigned the integer value 71, telling us the string was found starting at the 71st character. If you wanted to get the substring, you could then use a command such as

```
Name = substr(Array,Element,10)
```

to read it directly. Alternately, you could use the value found in Element to calculate in which array element the match was found.

This ability to create an array, where none existed is merely an exercise in creative programming. All we did was use the existing commands to create something more powerful that was not built into the language. In fact, that is exactly what you do every time you write a program: You create something new that did not exist before you began. What's really amazing about programming is that the end result is greater than the sum of its parts!

Section Four
Appendixes

Third-Party Software and Hardware

The AmigaVision Version 1.70 Update

– A – Third-Party Software and Hardware

The AmigaVision software-authoring system is a sort of glue with which you can stick together a wide variety of data — text, pictures, animations, sound, music, and video — with a logical set of instructions to manipulate that data. The instructions are, of course, the program, which you create during the planning and implementation stages of software development. The other components — the data — are created either by you or by someone else using software or hardware external to AmigaVision. In this chapter I will discuss some specific packages that I have found to be useful as development tools and accessories for AmigaVision programmers and as beneficial course components for the users of your applications.

Software

Because AmigaVision supports the IFF-ILBM and ANIM5 standards that most Amiga graphics and animation software also supports, you can use picture and animation files generated by almost any such program in your AmigaVision applications. Most sound digitizers offer the option to save sounds in the IFF-8SVX format AmigaVision uses. Likewise, it is almost impossible to find a word processor or text editor that cannot generate AmigaVision-compatible files; most can save documents in ASCII format. Music software is different, however. Because of the strict requirements AmigaVision imposes for music scores and instrument files, you have a more limited choice in that area.

The following pages contain recommendations of software and hardware items that I think you will find to be useful and compatible with AmigaVision. Most of these, especially the software, are products I personally find useful for Amiga-Vision development. The list is not exhaustive; there are many other worthy products, and their omission here is not meant to imply that only those listed are appropriate.

2-D Graphics and Animation

In my opinion, the most useful graphics tool available to Amiga owners and AmigaVision developers is DeluxePaint III by Electronic Arts. This program supports all display modes except HAM (Hold And Modify) and multiple screen resolutions ranging from 320-by-200 to 704-by-480 overscan. Its animation support includes both full-screen and brush files. In addition, DeluxePaint III has reasonably good support for Amiga bitmapped fonts, including multicolor fonts. Its drawing tools are excellent, easy to learn, and intuitive to use. For anything except HAM images, DeluxePaint III is the program I reach for first.

When you do need HAM graphics support you have several choices. The Paint module of Electronic Arts' Deluxe PhotoLab supports all Amiga modes, including HAM. While not quite as good a paint package as DeluxePaint III, it is a reasonable choice when you want one program capable of handling all display modes.

Photon Paint 2.0 (MicroIllusions) is a HAM-only paint program. Besides a good complement of drawing tools, it offers specialized brush features that allow you to wrap a bitmapped brush onto a geometric or hand-drawn shape. It also supports some full-screen animation features, making it useful for editing small HAM animations.

SpectraColor (Aegis/Oxxi) is a new HAM program created by the authors of Photon Paint (in fact, some people consider it to be the latest incarnation of Photon Paint). Among its features is an animation system that includes animated HAM brushes. SpectraColor will prove important to those creating HAM animations for use in AmigaVision.

Digi-Paint 3 (NewTek), another HAM-only package, does not support animation or animated brushes and lacks some important drawing tools (such as the area fill tools found in SpectraColor and Photon Paint). It does, however, have a number of special drawing modes and tools that make it valuable for editing and creating HAM images. Example modes are RubThru, which reveals a background image as you draw, and Colorize, which allows you to change the color and shading of a bitmap without changing the picture itself. The Repeat tool repeats your last drawing action with a new set of parameters (new color, new draw mode, and so on).

My experience has shown that if you have a limited budget, your best bets for still 2-D work are DeluxePaint III and SpectraColor. If possible, add Digi-Paint 3 to your collection and you will be set for almost every eventuality.

DeluxePaint III and SpectraColor have strong animation features as well, and are essential for creating and editing the 2-D animations you use in AmigaVision. They are also useful for editing 3-D animations. Both programs support cel (page-flipping) and brush animation. You will likely want to edit or add to animations generated by other software before using them in your AmigaVision applications. For nonHAM files, DeluxePaint III is the best choice; in the case of HAM animations, SpectraColor is best.

Another good 2-D animation program is The Animation Studio (Walt Disney Computer Software). This software uses a display technique called "onion skin," in which each frame of the animation is somewhat translucent. This allows you to see several frames back while you draw on the current frame, and thus makes it easy to place and alter objects to achieve a smooth flow, as you check their positions earlier in the scene.

3-D Graphics and Animation

If you plan on creating three-dimensional models or animations for display in your AmigaVision programs, you have the good fortune of being able to choose from a large assortment of software.

Imagine (Impulse) is one of the best 3-D ray-tracing programs on the Amiga market. It was developed by the same people who created Turbo Silver, which was renowned for generating impressive images and animations. Imagine is a major step up from Turbo Silver, with a greatly improved user interface, powerful new object-morphing features, and exotic animation options. It has excellent object-texture control that lets you perform a range of operations, from mapping existing images around an object to defining the characteristics of an object's surface.

3D Professional (Progressive Peripherals & Software) is a 3-D modeling and animation program with a large number of tools and editors for generating objects and scenes that are difficult to create by hand. It includes lathe, profile, and conic object editors, a fractal-landscape and tree generator, as well as options for generating 3-D fonts from Amiga bitmapped fonts and turning bitmapped brushes into 3-D objects.

Sculpt-Animate 4D (Byte by Byte) is an aging but still powerful 3-D rendering and animation program. It has a reasonably easy-to-use interface and excellent animation options. Sculpt has only a limited number of surface textures, but those it does support it handles well.

Image Processing

Image processing involves manipulating existing computer images, converting them between display modes (such as converting HAM to Halfbrite mode), and changing their physical display size. For AmigaVision applications, you will often need to process both pictures and animations.

The best software package for the manipulation of single images has got to be The Art Department Professional (ASDG). It allows you to convert images between a variety of formats (including nonAmiga formats), and rescale pictures to any required size. It is an excellent, almost essential piece of software for developers.

For manipulating animations, there are two very good applications. Animation Station (Progressive Peripherals & Software) can merge multiple animations into one or convert an animation from one display format or resolution into another. It also offers such special-effects processing as scrolling, flipping, and pixel mosaics.

For adding even more flash to your animations, ANIMagic (Oxxi) can't be beat. Among its features are some that allow you to add a large number of digital effects, including Page Turns, Slides, Tumbles, Blinds, Spins, Wraps, Confetti, and more on your animations.

The importance of these three programs cannot be overstated. The Art Department Professional's fast processing and conversion techniques in particular pay for the cost of the software by saving you precious time that would otherwise be spent redrawing rerendering images.

Fonts

There are many commercial and public-domain sources for single-color bitmapped fonts that you can use in your AmigaVision applications. Another type of font that you should definitely consider using is Colorfonts — bitmapped fonts that are composed of several colors and conform to the standard endorsed by Commodore. The extra colors are used to simulate such effects as 3-D bevelling, metal, or chrome surfaces, or merely color accents.

One of the best sources of Colorfonts is the collection by Kara Computer Graphics. Kara offers a wide variety in several point sizes, most of which are large enough to be used as video titles. The company has a series of spectacular animated Colorfonts, in which each letter is actually a DeluxePaint III brush animation consisting of several frames. The characters in these sets appear to spin or write themselves onto the screen while light shimmers across them.

Many Amiga graphic programs support Colorfonts, and although AmigaDOS 1.3 does not support them directly, the new release of the Amiga operating system — 2.0 — does. Just in case, Kara Computer Graphics supplies a driver for its fonts. AmigaVision's Object editor and Text icon, when operated under the 2.0 operating system, can use a Colorfont as a text font. Use of Colorfonts, however, does not set the screen colors automatically; if you plan on using Colorfonts with AmigaVision, you should first set up your display with the appropriate colors.

TV Text Professional (Zuma Group) is an excellent program for generating attractive text screens. Its extensive display abilities include outline, drop shadows, and 3-D extrusion effects. It also has a powerful background-screen generator, which, when combined with one or more font styles, can lend a very professional appearance to your displays. TV Text Professional is one of the best screen editors around.

Sound

In my opinion, the best program in the sound category is AudioMaster II (Aegis/Oxxi). This package works with a variety of hardware sound samplers, and, on machines equipped with 68020 or 68030 processors, can sample sounds at over 40K per second — an extremely high rate that produces near-CD quality results. AudioMaster also has a number of special-effects and sound-processing features that make it very useful to the AmigaVision author.

Oxxi's newest version, AudioMaster III, has all the features of the original with much improved, faster sampling rates that can exceed 55K per second. It also supports sound sequencing, which means you can direct a sample to replay sections a given number of times. This can result in a major memory savings on samples that involve a lot of duplication.

AudioMaster III supports a variety of hardware samplers, including those from Sunrize, Mimetics, Star Sound, Applied Visions, and Microdeal, although not all of these allow sampling at the maximum rate.

Music

AmigaVision requires its music scores to be in the IFF-SMUS format, which is the standard file format for Amiga music. While a number of programs support SMUS for music scores, not all of these work with AmigaVision because some save their instruments in different formats. The safest choice in music software for AmigaVision is Deluxe Music Construction Set (Electronic Arts), which allows you to read and write files in both its own DMCS format as well as SMUS. There are a great many public-domain music scores and instruments in DMCS format that you can load into Deluxe Music Construction Set and then resave as SMUS files.

I have found that local bulletin boards and commercial network systems (American Peoplelink, CompuServe, GEnie, and so on) are great sources of DMCS scores and instruments. With a modem and terminal software, you can download these and convert them to SMUS files using Deluxe Music Construction Set. If you have access to an Amiga users group, you can also check its software library for AmigaVision-compatible music files.

Text Editors and Word Processors

Almost any text editor or word processor can save its files in ASCII format, and beyond the need for ASCII compatibility, there are no other requirements for use with AmigaVision. I suggest you use whatever text editor or word processor you are comfortable with.

Hardware

Chapter One outlines the hardware requirements for both using and developing AmigaVision software. In this section I will discuss some specific hardware products that can make creating applications easier. Many of these will give your applications more power and flexibility.

Video Digitizers

As an AmigaVision software developer, you will often find that the images you need cannot be readily drawn or rendered with graphic software, but that they do already exist in print or on some video source (TV, videotape, or laser disc). A video digitizer will let you incorporate high-quality digitized art into your programs quickly and easily. You can then modify these digitized images using a graphic program.

One of the least expensive, and most versatile, digitizers is Digi-View 4.0 (NewTek). This unit is a slow-scan color digitizer, which means it can capture still pictures only — not moving images. Using a relatively inexpensive black-and-white video camera, Digi-View digitizes a color image three times, once through each of three filters — red, green, and blue. The Digi-View software then combines these filtered images to produce a high-quality full-color screen display. Digi-View supports all Amiga display modes, from black-and-white to 4096-color HAM. For those on a budget, Digi-View is one of the best choices around.

If you need to capture moving images or do not want to bother with color filters, FrameGrabber (Progressive Peripherals & Software) is a very good choice. It allows you to grab an image from almost any still or video source in real time; it requires no slow scanning as does Digi-View. FrameGrabber's 2.0 software offers a fairly good selection of image-processing features, allowing you to manipulate the captured video image in a number of ways. It also has a very good animation option, which you can use to digitize a sequence of frames and turn them into an animation for use in AmigaVision. The animation controls also support time-lapse photography — grabbing a frame every *n* seconds, with *n* being anywhere from one second to many hours. In almost every way, FrameGrabber is a very good digitizer.

If you have the proper supporting equipment and money, and want the best in Amiga video digitizers, the Video Toaster (NewTek) is worth looking at. While it is principally a special-effects generator for videographers, the Toaster also is a high-quality 24-bit real-time frame grabber. With its companion paint software, you can save its frames as IFF-24 images, which you can later convert to Amiga modes via an image-processing program such as The Art Department Professional. In addition, the Toaster comes with a powerful 3-D rendering and animation program called LightWave3D, which can generate stills and animations. (Keep in mind that these still images and every frame of the animations must be converted from IFF-24 to an Amiga display format before you can use them in AmigaVision.)

The Video Toaster is currently available only for the A2000 and A2500 machines. (Although an A3000 version is supposedly being planned, no release date has been set.) Also, the Video Toaster requires a time-based corrector (TBC) in order to capture images from videotape. Using a camera (for live video), a laser-disc player, or a television does not require a TBC, however.

Audio Digitizers

An audio digitizer (sampler) can capture sound and convert it to a format that you can then load from disk and replay via software. In AmigaVision, digitized sound is controlled by the Sound icon or used as feedback in the Object editor. There are numerous audio digitizers for the Amiga, and just about any will work for you. Even though they all come with software to control the hardware, in most cases the supplied software is less capable than dedicated commercial sampling software such as AudioMaster III. If you plan to buy a dedicated software sampling package, make sure it supports your hardware.

Over the years I have used two Future Sound audio digitizers (Applied Visions). The first was the original model designed for the A1000, and the latest is the A500 model, which also works well with the A2000. Both of these digitizers proved reliable and capable of producing good-quality sound in use with AudioMaster III. Future Sound 500 offers both microphone and stereo RCA-line input, as well as a slider for controlling the input level.

Another audio digitizer I have used is A.M.A.S. (Microdeal), which also provides both stereo RCA-line and microphone input. It comes with three MIDI ports, which you can use for acquiring samples from MIDI-compatible devices. I found A.M.A.S. to be quite acceptable and capable of high-speed sampling.

Touch Screens

At the time of this writing, there are three different touch screens available for use with AmigaVision: the Elographics Touch Screen (Elographics), Future Touch (Amigo Business Machines), and MicroTouch (MicroTouch Systems).

Since this technology and support software is somewhat new to the Amiga, I made a request of all three companies for demonstration units to be used while this book was being written. Of those three, only Amigo Business Machines was able to supply me with a unit, so my experience with touch screens as related to AmigaVision is limited to the Future Touch system. (I would like to take this opportunity to thank Amigo Business Machines for its cooperation.)

The system Amigo loaned me consists of a modified 1084S monitor fitted with the company's controller card. It connects to the Amiga through either the serial port or the parallel port, depending on what your system requirements are.

The Future Touch monitor functions as does any other monitor — until the driver software is activated. After that, you can move the pointer either by touching the screen or moving the mouse in the usual way. Touching the screen makes the system react as if you had moved the mouse pointer to that location and clicked the left mouse button.

To make use of a touch screen in your AmigaVision applications, simply write your program to allow mouse control. Nothing else needs to be done except for activation of the software driver. You can do this either via your startup-sequence or from within your AmigaVision applications using the Execute icon to run the driver as an external program.

The mouse stays active once the driver is initiated, so the user of your software can use either input method if you so desire. If you want to make an application totally touch-screen controlled, it is a good idea to turn off the mouse pointer in your Screen and Animation icons during program development. That way, the pointer won't be jumping around as the user moves his or her fingers on the screen.

You can shut down the touch-screen driver software by passing a parameter with the Execute command. Thus, if your software has an option to quit and return to Workbench, you can turn the driver off by reexecuting it before the program finishes up.

I used the Future Touch system for several months and found it to be very compatible with AmigaVision and AmigaVision applications.

Genlocks

A genlock is a device that synchronizes incoming video signals with the computer's video, producing a single signal with computer graphics overlayed on the externally generated image. Because AmigaVision has very powerful video features in the form of its laser-disc control, a genlock is very desirable from the perspectives of both the AmigaVision developer and the end user.

The most common implementation of video in AmigaVision is when you are programming an interactive course using images supplied from laserdisc. In these situations you will usually combine the computer's graphics and the laserdisc video and display them on your RGB screen. Some genlocks, especially those designed for video use, do not have this RGB display feature, instead the combined signal is sent out only as a composite NTSC video signal. While this is exactly what you want for video recording, if you are creating a desktop multimedia-based application where the user is looking at the Amiga's RGB display, you will want the images to all appear on this screen in RGB mode and not on a composite monitor or TV in composite video mode.

Unless you are planning to record your genlock's output onto videotape, you might not want to spend a large amount of money getting an expensive industrial or broadcast quality unit. If you plan to display only the genlock's signal on the RGB screen, you might want to opt for a lower quality genlock that is nevertheless suitable for desktop presentations. I used this logic when I acquired a genlock for use with AmigaVision on my A2000. After looking around, I settled on the Commodore A2300. This device plugs into the A2000's internal video slot and outputs the combined signal directly onto the RGB display. It also has a composite video-out that allows me the option of recording the output. (While the composite output is not broadcast quality, it is adequate for home and low-level industrial use.)

If you do plan to use your genlock for serious video-recording work as well as desktop presentations, I recommend SuperGen (Digital Creations). This external genlock works with all Amiga models, supports the RGB output needed to display images on your Amiga monitor, and is suitable for industrial and broadcast recording. Its price is quite a bit higher than the A2300 (although it is lower than many other genlocks in its class), but considering the quality and the facts that it works on any Amiga and (in the case of the A2000 and A3000) it keeps your video slot free for other devices, it is well worth the expense.

Laser-Disc Players

AmigaVision supports a number of industrial-style laser-disc players and at least one videotape deck (the Sony Umatic). In addition, a number of companies are developing techniques to use consumer-style video-disc players via remote infrared controllers. My advice to AmigaVision authors shopping for laser-disc players is to be *sure* the system you intend to buy is compatible with AmigaVision. If possible, talk to others who have the same system before you buy.

The safest bet is to purchase one of the decks listed in your AmigaVision Video Configuration requester. My experience has been primarily with the Sony 1200, although I have also used the Sony 1500 and 1550 models. While the 1200 is one of the least expensive players AmigaVision supports, I have had good luck with it.

At this writing I have also been told that Commodore is working with NEC to produce an AmigaVision driver for NEC's PC-VCR. This SVHS videotape deck has its own form of time code for frame accurate positioning, and is supplied with a serial port for connection to and control through the computer. For more information on the availability of this and other AmigaVision video device drivers, check with your local Amiga dealer.

– B –

The AmigaVision Version 1.70 Update

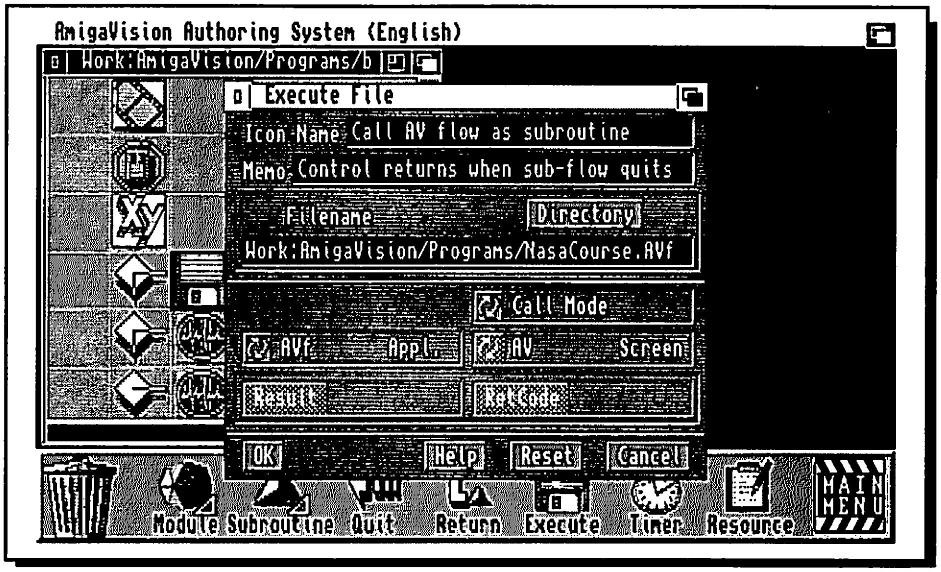
Just before this book went to press, Commodore contacted me with the news that a new release of AmigaVision was scheduled to occur in the next couple of months. They sent this new version to me,
in order that I might add its changes to the AmigaVision Handbook.

The major changes are the addition of an important new feature to the Execute Icon and a couple of new functions for use in the Expression editor. Along with these new features they fixed a number of obscure bugs that were in the 1.53G release.

Executing AmigaVision Applications

The Execute Icon has had a new option added to its requester. Now, in addition to supporting external programs using ARexx, CLI, or Workbench it offers the option of executing AmigaVision applications! This feature is called program chaining, and it means you can combine multiple AmigaVision programs into one larger, more complex program while only having part of the program in memory at one time, a very important consideration if you have memory limitations. You should understand that when you execute an AmigaVision application with the Execute command, you are not merely loading a new version of AmigaVision. It is the AmigaVision in memory that is actually running the new flow.

If you look at Figure A2-1, you will see the new Execute requester. Notice that the multistate gadget that controls the type of application being called, is now set to AVf Application instead of CLI, Workbench, or ARexx. Another new gadget above the screen gadget sets one of the two available modes (Call and Link) for launching AmigaVision applications. Finally, when you are using the AVf Application mode, the screen gadget is always set to AV Screen.


AmigaVision Authoring System (English)

Work:AmigaVision/Programs/b

Execute File

Icon Name: Call AV flow as subroutine

Memo: Control returns when sub-flow quits

Filename: Work:AmigaVision/Programs/NasaCourse.AVf

Call Mode

AVf Appl.

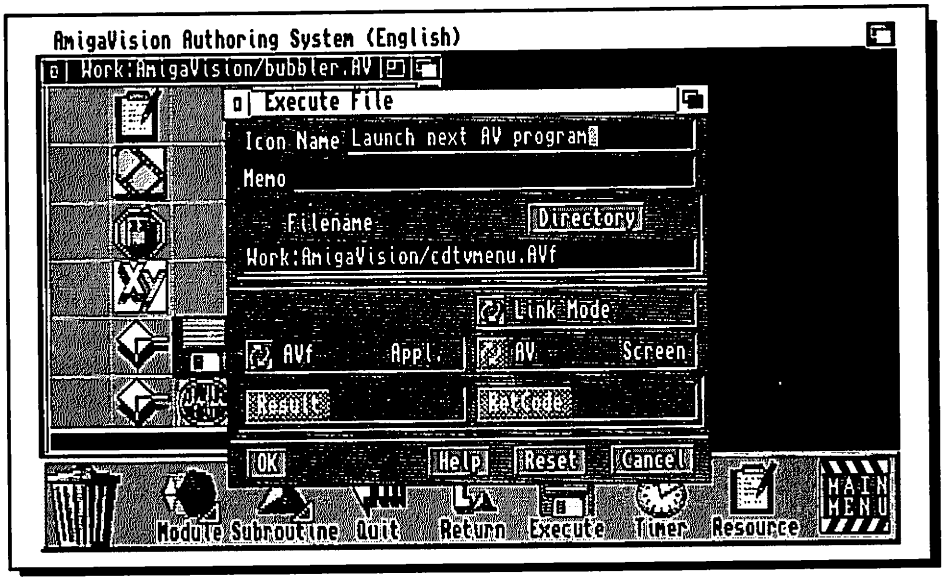
AV Screen

Figure A2-1
The Call Mode of the Version 1.70 Execute requester

The first mode is Call. It allows you to call another AmigaVision application from within an AmigaVision application. In this mode it treats the disk-based AmigaVision application as if it were a subroutine. When you execute the external AmigaVision application, all the global variables created within the calling program are available to the subroutine. And, if you create any new global variables within the disk-based AmigaVision you called, these will be available to the host program when the subroutine finishes and control returns to the calling program. Any interrupts in effect when the program is called remain in operation during the execution of the new program.

When a Called flow finishes (without encountering a Quit icon), program execution returns to the calling program. The memory used by the subroutine flow is freed up. Program execution will then continue at the icon immediately following the Execute icon that called the flow subroutine.

The Link mode (see Figure A2-2) works quite differently than Call mode. When an AmigaVision program calls another AmigaVision program using Link mode, the program that does the linking is removed from memory. Any variables in effect at that time are transferred to the linked AmigaVision program as global variables in the new program, and any display objects are removed from the screen. However, whatever screen image is in effect remains until the new program displays another screen.


AmigaVision Authoring System (English)

Work:AmigaVision/bubbler.AV

Execute File

Icon Name Launch next AV program

Memo

Filename Directory

Work:AmigaVision/cdtvmenu.AVf

Link Mode

AVf Appl.

AV Screen

Result

RetCode

Figure A2-2
The Link mode of the Version 1.70 Execute command

As far as the new program is concerned, the parent program that preceeded it was just a Variable icon that initialized a group of variable definitions. When this child program is finished, the entire flow is finished. There is no return to the parent program unless it is explicitly executed via the Execute command icon. This reexecution of the parent might occur in the case of a main menu program that merely executed child programs in response to a user selection. When the child programs are finished they would then reexecute the parent menu program.

Since each AmigaVision application can have certain default values defined, it is important to give that issue consideration when calling other AmigaVision applications. Here are the effects on those defaults of chaining AmigaVision programs together.

Close Workbench	Set by the first application.
Double Buffering	Set by the current applications defaults.
Overscan	Set by the current applications defaults.
Instruments	Set by the current application, unless it has no defined path. In this case the path is set by the previous application.
Initial Screen	Used by the first application only.
Feedback Sound	Set by the current application, unless it has no default feedback sound. In this case the sound is set by the previous applications default settings.
Audio Volume	Set by the current application unless the default feedback sound was set by the preceeding application. In this case the default volume is set by the preceeding application.

Functions

Three new functions have been added to the AmigaVision Expression editor's battery of commands. In addition, one function has been enhanced to support a new hardware option from Commodore.

Dec(n)

This command is the opposite of the inc(n) command already supported by AmigaVision. Dec(n) takes the value of the number or numeric variable inside the parenthesis and decreases it by one, then returns the value to a variable. It would be used in the following manner:

```
j=dec(n)
n=dec(n)
```

In the first example, the variable *j* would be assigned the value of n-1. In the second case *n* would be decremented by 1.

Defined(s)

This is a Boolean function that returns True if the variable has already been defined. You would use this function in a child program to see if a variable had been defined in the parent program. Note that if the name of the variable is stated as a literal, it must be enclosed within quotes.

```
Exists=defined(x)
```

Self(n)

This returns the file name and/or path of the current application. The parameter indicates what form to return. A value of zero indicates only the file name itself should be returned, while non-zero indicates you want the whole path name.

```
ParentProgram=self(1)
```

You would use the self(n) function in a parent program before calling the child. The variable you use to store the name would then be available to the child. This would be very useful in situations where you were using Link mode and would want to reexecute the original parent program.

Mouse()

The Mouse() function has been upgraded to return a response of "MMB" if the user clicked on the middle mouse button instead of the right or left mouse button. This was needed because Commodore Business Machines plans on releasing a new, more sensitive mouse that has three buttons instead of the two found on current Amigas.

Upgrading AmigaVision

If you are interested in getting an upgrade to your current version of AmigaVision, contact your local dealer for information. If you are not sure what version of AmigaVision you are using, select the About option from the Project menu. This will give you the information detailing the version of AmigaVision you are using.

Index

The *AmigaWorld* Video Library

It's like having a professional computer consultant at your side, 24 hours a day!

Animation Video Volume I

When the Editors of *AmigaWorld* canvassed the Amiga community looking for the best in Amiga animation, the response was overwhelming! Submissions poured in from Amiga master artists and super-talented readers. The result is a dramatic video featuring dozens of world-class animators. This video is quickly becoming a collector's item! Approximately 48 minutes in length. Available in NTSC.

Desktop Video Don't miss out while others get the inside angle on: Pre-production, production and post-production; Home and studio settings; Selecting video equipment and accessories; Edit-free shooting with your camera; Recording from the Amiga onto video tape; Selecting a Genlock and how to get the most from it; Tips and Tricks on how to improve your video skills; Adding special effects; *And lots more!*

The Musical Amiga Your Amiga has exquisite sound and music capabilities. Learn all the details of getting started with music. . . what you need to begin and playing music with existing software. We'll teach you digitizing and audio sampling and even give you an introduction to MIDI (Musical Instrument Digital Interface). This video is the best instructional tool for any Amiga owner interested in learning how to create music with their Amiga!

Amiga Graphics (Volume I)

There's no easier, faster or better way to learn how to create your own Amiga masterpieces! This hour-long video can teach you all you need to know about how to get started in graphics. . .paint programs. . . elements of design. . .creating an image. . .and lots more! Plus, three extensive sessions on FONTS, CLIP ART, and even DIGITIZING!

Getting Started With Your Amiga

This comprehensive, easy-to-follow video is packed with valuable information. Learn how to assemble your Amiga. . .how to use the Workbench. . .add a digitizer or genlock. . .how to use system utilities. . .and much, much more! Best of all, GETTING STARTED WITH YOUR AMIGA is there whenever you need a quick refresher on any aspect of Amiga computing!

AMIGA

Fill out coupon and mail to:
Video Library, PO Box 802, 80 Elm Street, Peterborough, NH 03458

or call 1-800-343-0728

✓ **YES!** Please send me the tapes I've selected below!

Qty.	Description	Price	Total
	Getting Started With Your Amiga	$29.95	
	Amiga Graphics (Volume I)	$29.95	
	Desktop Video	$29.95	
	The Musical Amiga	$29.95	
	Animation Video (Volume I)	$19.95	
		Subtotal	
		Shipping/Handling *	
		Total Enclosed	

Name ____________________

Address ____________________

City ____________ State ______ Zip ________

Payment method: [ ] Check

Charge my: [ ] MC [ ] VISA [ ] American Express [ ] Discover

Card # ____________________ Exp. Date __________

Signature ____________________

* Shipping/Handling Per Order:
1 Tape = $2.95
2 or more Tapes = $5.00

Available in VHS only. Make checks payable to *AmigaWorld* Video Library. Please allow 4–6 weeks for delivery. Foreign orders add $7.50 for air mail orders. Payment must be made in US funds which can be drawn on a US bank. PAL is available if specified. Add an additional $10.00 for PAL version plus $7.50 air mail delivery. IDG/Peterborough, publisher of *AmigaWorld*, is the licensed North American distributor of all above-mentioned videos. © 1990 by Razza Video USA. All rights reserved. Amiga is a registered trademark of the Commodoore Business Machines, Inc.

IDG Books Worldwide Registration Card --

AmigaWorld Official AmigaVision Handbook

Fill this out—and hear about updates to this book & other IDG Books Worldwide products!

Name ____________________

Company/Title ____________________

Address ____________________

City/State/Zip ____________________

What is the single most important reason you bought this book? ____________________

____

Where did you buy this book?

- ☐ Bookstore (Name ____________________)
- ☐ Electronics/Software Store (Name ____________________)
- ☐ Advertisement (If magazine, which? ____________________)
- ☐ Mail Order (Name of catalog/mail order house ____________________)
- ☐ Other: ____________________

How did you hear about this book?

- ☐ Book review in: ____________
- ☐ Advertisement in: ____________
- ☐ Catalog
- ☐ Found in store
- ☐ Other: ____________

How many computer books do you purchase a year?

- ☐ 1
- ☐ 2-5
- ☐ 6-10
- ☐ More than 10

How would you rate the overall content of this book?

- ☐ Very good
- ☐ Good
- ☐ Satisfactory
- ☐ Poor

Why? ____________________

What chapters did you find most valuable? ____________________

What did you find least useful? ____________________

What kind of chapter or topic would you add to future editions of this book? ____________________

____

Please give us any additional comments. ____________________

____

Thank you for your help!

☐ I liked this book! By checking this box, I give you permission to use my name and quote me in future IDG Books Worldwide promotional materials.

IDG BOOKS

Fold Here

Place
stamp
here

IDG Books Worldwide, Inc.
155 Bovet Road
Suite 730
San Mateo, CA 94402

Attn: Reader Response

AMIGA WORLD Official AmigaVision Handbook

"everyone will benefit from the hints, examples and undocumented features." —Cathy Godfrey, Authoring System Support Specialist, Commodore

CERTIFIED IDG TECHNICAL EDIT

Create presentations, multimedia programs, sophisticated applications with AmigaVision and this definitive guide and reference!

Commodore has developed a fantastic new tool for creating visual applications—AmigaVision. Users new to the package along with sophisticated multimedia mavens will find this guide invaluable. Three in-depth sections will bring out your multimedia talent immediately!

Section One gets you started with the basics:
- What Multimedia Is
- The Art of Authoring
- All the Basic Menus
- Common Requesters
- Program Editing Information

Section Two is a thorough command reference of Control Commands:
- Interrupts
- Database Icons
- Wait Icons
- Audio Visual Icons
- Module Icons

Section Three goes in-depth, with Editors, Tools & Programming:
- Object Editor
- Video Disk Controller
- Expression Editor
- Database Editor
- Complex Program Structures

Plus a Special Guide to Version 1.7!

Book Level: Beginner to Advanced

About the Author:
Louis Wallace is the Senior Editor, Technology at *AmigaWorld* magazine. He has been writing about the Amiga, its products, and the market since 1985, when he got his first Amiga. He has also contributed hundreds of articles to magazines across North America & Europe.

About the Cover Art:
Artist Jim Sachs created the cover illustration on an Amiga, using DeluxePaint 3.

Computer Book Shelving Category:
Computer Systems/Amiga

$24.95 U.S./$33.95 Canada/£22.95 U.K.

IDG BOOKS

IDG Books Worldwide
San Mateo, CA 94402
An International Data Group Company

ISBN 1-878058-15-0

52495

9 781878 058157

PRINTED ON RECYCLED PAPER

Cover Design by Owens/Lutter