The UltraMacros Primer: How to Use TimeOut UltraMacros

by

Mark Munz

Forward by UltraMacros author
Randy Brandt

Published by the National AppleWorks Users Group

The UltraMacros Primer:
How to Use TimeOut UltraMacros

by
Mark Munz

Forward by UltraMacros author
Randy Brandt

Published by the National AppleWorks Users Group

The UltraMacros Primer: How to Use TimeOut UltraMacros

PUBLISHED BY
National AppleWorks Users Group (NAUG)
Box 87453
Canton, Michigan 48187

Printed and bound in the United States of America.
ISBN 0-9620807-3-X

To Mom and Dad

Table of Contents

Preface

The UltraMacros Primer began as two articles on UltraMacros that I wrote for the NorthWest Apple Pickers (NWAP), my local Apple users group. Before long, I got a call from Cathy Merritt of NAUG, expressing interest in the articles and asking if I could write a series of five articles on UltraMacros for the ***AppleWorks Forum***. I agreed.

That telephone call was the beginning of a long and exciting relationship. Now, 15 months later, this book is complete.

My original goal was to help UltraMacros owners become UltraMacros users, and that remains the purpose of this book. I tried to write a comprehensive guide to UltraMacros that would meet the needs of both UltraMacros novices and experienced users. I hope you find these efforts worthwhile.

Acknowledgments

Many people have made direct or indirect contributions to the development of this book. I hope to name everyone who helped me, but apologize in advance for those whom I omit.

First, I thank my mom and dad. It was their support of my computer interests that lead to my getting the Apple IIe computer on which I wrote half this book.

Second, I thank the Washington-based NorthWest Apple Pickers (NWAP) Users Group. It wasn't until I joined the group that I was able to share my computer knowledge with others. Thanks Doug, Jim, Kent, Marvin, Murray, Scot, and Susie for all your encouragement. Kent Hayden deserves an extra thanks because he caught several early macro typos and let me know, even after I left Washington for the warmer climes of southern California.

Then there is Randy Brandt, the author of UltraMacros. Without Randy's encouragement and without UltraMacros itself, it would be impossible to write this book.

Now we're at the NAUG part of my story. I thank Cathy Merritt for noticing me in NWAP, my local user group's newsletter and for all her help editing this book.

I also thank William Marriott for his helpful editing and constructive criticism of the early chapters of the book.

And thanks to Nanette Luoma for the excellent layout and design work she did on the book.

Finally, I give a special thanks to Dr. Warren Williams. Warren provided the support and encouragement I needed throughout this 15-month process. There were times when I was ready to give up on the whole thing. His guidance and patience are tremendously appreciated. At times I felt his encouragement and cajoling managed to squeeze the impossible from me.

Forward

I first met Mark Munz on the Pro-Beagle technical support bulletin board about two years ago. A sophomore computer science major, he impressed me with his grasp of UltraMacros programming and insightful questions about AppleWorks and TimeOut. Soon the standard online response to "How do I do this?" questions was "Give Mark Munz a few hours; he'll figure it out!"

Mark began developing several interesting patches to AppleWorks, and seeing the potential in them, I offered to market them through JEM Software. In exchange for the rights, I would pay him royalties and see what I could do to connect Mark with Beagle Bros. Before long JEM was distributing Late Nite Patches and Mark was packing his disks and moving from Washington to California.

I've enjoyed collaborating with Mark on several projects, such as MacroTools II, The AW 3.0 Companion, MacroEase, and TimeOut TextTools. He has written code for TimeOut HelpScreens (on the PowerPack disk), for TimeOut TeleComm, and has been a source of many useful ideas for improving AppleWorks. His innovative macros have surprised me with the potential uses of TimeOut UltraMacros and I used many of his ideas while writing UltraMacros 3.0. Writing powerful macro programs like Mr. Invoice has taught him virtually all there is to know about UltraMacros.

In September of 1988 Mark began sharing his macro knowledge in a monthly column in the ***AppleWorks Forum***. His Macro Primer articles detailed everything from how beginners could use the new commands in UltraMacros to how programmers could use the powerful if-then-else logic. Guiding the aspiring UltraMacros programmer step-by-step, this column quickly became the standard reference for anyone who wanted to maximize AppleWorks' efficiency.

The UltraMacros Primer is based on the Macro Primer column, with additional chapters that teach even more. In this book you will learn how to get started using the built-in features UltraMacros adds to AppleWorks, how to record your first macro and write your first compiled macro. Once you've mastered those techniques, you will study how to clearly document and organize your macros, how to create task files, how to use variables, how to write looping macros, and how to write conditional macros that do things you'd never imagined before starting your training.

The Macro Primer has already taught many aspiring UltraMacros programmers; this fine book, based on those columns, will help many other UltraMacros owners discover the power to be the best AppleWorks users they can be.

Randy Brandt
San Diego, CA
October 1989

1: Getting Started with Macros

This chapter describes how to use UltraMacros to enhance the power of AppleWorks. By the end of this chapter you should know the capabilities of UltraMacros and how to use its built-in commands and macros. Future chapters will describe how to record your own macros, how to create and use Task Files, and how to use the programming language built into UltraMacros.

A "macro" is defined in Webster's New Collegiate Dictionary as "a single computer instruction that stands for a sequence of operations". For example, it usually takes several keystrokes to tell AppleWorks to print a word processor document (Apple-P, RETURN, RETURN, RETURN). With a macro, a single keystroke combination (Solid-Apple-P) can perform the same job.

Macros have two obvious advantages. First, they save keystrokes. You can tell AppleWorks to memorize any set of keystrokes and then "invoke" or "play back" that macro whenever you need those words or commands in your document. Second, macros speed up AppleWorks. Macros immediately respond to AppleWorks questions and menus. There is no waiting for user input.

Macro Programs for AppleWorks

There are three programs that give you macro capabilities with the current version of AppleWorks. Beagle Bros' UltraMacros modifies AppleWorks and adds macro capabilities directly to the program. DiversiKey, from Diversified Software Research, and MacroMate, from Roger Wagner Publishing work only on the Apple IIGS, but add macro capabilities to most text-based programs that run on that computer. DiversiKey and MacroMate are not specific to AppleWorks and do not have all the power of UltraMacros, but these enhancements also work with Multiscribe, Bank Street Writer III, most spreadsheet programs, and other Apple II programs.

Of the available alternatives, I favor UltraMacros. It is the most powerful of the programs and it offers a number of features not available with the other two alternatives. In this book, I specifically describe how to use UltraMacros, although you can duplicate some of the functions I describe with the other programs.

UltraMacros' Additions to AppleWorks

UltraMacros enhances AppleWorks in five ways:

1. It adds new AppleWorks commands.
2. It includes a set of built-in macros for AppleWorks.
3. It gives AppleWorks the ability to memorize and replay any set of keystrokes you enter.
4. It lets you create task files so you can automate any task or set of operations.
5. It offers a complete programming language you can use to create menu-driven applications or automate complex tasks.

In this book I will describe how to use each of these features of UltraMacros.

UltraMacros and the Keyboard

The source of UltraMacros' power is its ability to recognize keystroke combinations that are typically ignored by AppleWorks. For example, AppleWorks does not differentiate between the Open-Apple and Solid-Apple keys. (The Option Key on an Apple IIGS functions like the Solid-Apple Key on the Apple IIe and IIc.) When you use AppleWorks, you can use the Open-Apple and Solid-Apple keys interchangeably; both keystrokes initiate the same commands.

UltraMacros "teaches" AppleWorks to differentiate between the Open-Apple and Solid-Apple key combinations. With UltraMacros installed, Open-Apple, Solid-Apple, and Both-Apple key combinations invoke different commands. In addition, UltraMacros recognizes Control-Key combinations. Thus, each key on the Apple keyboard can be used to generate additional AppleWorks commands. Throughout this book, I will use the

abbreviations "oa-", "sa-", "ba-", and "ctrl-" to designate Open-Apple, Solid-Apple, Both-Apple, and Control-Key keystrokes.

Built-in Commands

UltraMacros adds new commands to AppleWorks. For example, UltraMacros adds a Date Command. Once you install UltraMacros, you can press <sa-'> and AppleWorks will immediately type the current date in the format "September 1, 1990". Typing <sa-"> invokes the Date2 Command, which types the date in the format "09/01/90".

Two other useful commands are the "UC" (Upper Case Convert) and "LC" (Lower Case Convert) commands. Once you install UltraMacros, you can put the cursor on any lower case letter and issue the Upper Case Convert command (<oa-:>). AppleWorks will replace that character with its upper case equivalent. Similarly, the LC command (<oa-;>) will convert an upper case letter into lower case.

Figure 1.1 lists the commands UltraMacros adds to AppleWorks.

Built-in Macros

In addition to the built-in commands, UltraMacros includes a set of pre-defined macros that are automatically available with AppleWorks. For example, <sa-A> invokes a macro that takes you to the Add Files Menu from anywhere within AppleWorks. Once you install UltraMacros, you can add a file to your desktop from anywhere in AppleWorks by pressing <sa-A> and selecting the file from the disk catalog.

Figure 1.2 lists the Solid-Apple macros built into UltraMacros and *Figure 1.3* lists the other macros built into UltraMacros. Additional information about each macro appears in the AppleWorks word processor file entitled "Macros Ultra" on the UltraMacros disk.

Figure 1.1: Commands UltraMacros Adds to AppleWorks

Keystroke	*Description*
sa-.	Jump to first space after cursor.
sa-,	Jump to first space before cursor.
sa-'	Enter date in long form (September 1, 1990).
sa-"	Enter date in short form (09/01/90).
sa-=	Enter time in 12-hour format (1:42 pm).
sa-+	Enter time in 24-hour format (13:42).
sa-Return	Find the next Carriage Return character (word processor only).
sa-^	Find the next formatting or "normal" caret (^) (word processor only).
oa-X	Start recording a macro.
oa-Delete*	Delete character under the cursor.
oa-:	Convert character at cursor to uppercase.
oa-;	Convert character at cursor to lowercase.
oa-!	Summon insert cursor.
oa-@	Summon Zoom Out mode (formatting and formulas hidden).
oa-ctrl-W	Increment character at cursor (i.e.: "a" becomes "b").
oa-ctrl-A	Decrement character at cursor (i.e.: "b" becomes "a").

* Built into AppleWorks 3.x. Not an UltraMacros 3.x function.

Keystroke Macros

The UltraMacros <oa-X> command adds another feature to AppleWorks; the ability to record keystroke macros. When you press <oa-X>, UltraMacros starts memorizing each key you press. You can then define those keystrokes as a new macro and replay the keystrokes upon command.

Figure 1.2: Solid-Apple Macros Built into UltraMacros

Keystroke	*Description*
sa-A	Add files to desktop.
sa-B	Begin a memo.
sa-C	Center line of text.
sa-D	Delete word under cursor.
sa-F	Find text and clear old search text.
sa-G	Go to special marker.
sa-H	Go to home cell in spreadsheet.
sa-I	Indent three characters.
sa-J	Enter your address.
sa-K	Calculate page breaks, then find a page.
sa-L	WP: left justify; SS: change entry's label layout.
sa-M	Set special marker.
sa-N	Sort column in spreadsheet numerically.
sa-0	Indent zero.
sa-P	Print file; add date if spreadsheet or data base report.
sa-Q	Go to next file on desktop.
sa-R	Change a printer option.
sa-S	Save and remove a file.
sa-U	Undo last "UltraMacros delete".
sa-Y	Delete line.
sa-Z	Delete to end of file.
sa-9	Delete last line in file.
sa--	WP: insert subscript codes; SS: shrink column width.
sa-/	WP: force a page break; SS: copy a label or value.

You do not need to be able to program to use this valuable feature of UltraMacros. You tell UltraMacros to memorize your keystrokes, do your work as usual, and tell the program to repeat those keystrokes upon command. I will describe how to capture, compile, replay, and save keystroke macros in Chapter 3.

Figure 1.3: Other Macros Built into UltraMacros

Keystroke	*Description*
sa-Left Arrow	Go to beginning of word processor line, or go to first column of spreadsheet, or jump left one word in a data base category.
sa-Right Arrow	Go to end of word processor line, or go to last column holding data in a spreadsheet, or jump right one word in a data base category.
ba-+	WP: Insert superscript codes; SS: Expand column width.
sa-/	WP: Force a page break; SS: Copy a label or value.
oa-<space>	Insert a space, even in strikeover mode.
sa-ctrl-A	Sort a spreadsheet column alphabetically.
sa-ctrl-B	Boldface word under cursor.
sa-ctrl-C	Close a letter.
sa-ctrl-F	Find next forced page break.
sa-ctrl-L	List all files on current drive.
sa-ctrl-N	Create a new AppleWorks word processor file.
sa-ctrl-O	Delete next caret (^).
sa-ctrl-P	Begin data base phone log program.
sa-ctrl-	Exit AppleWorks, ignoring all changes.
ba-ctrl-S	Save all desktop files to current disk and exit.

Task Files

Task files are sets of macros you can use to perform a specific task. For example, you can set up a task file that boots up AppleWorks, concatenates ten different spreadsheet files into a single file, prints that summary spreadsheet, saves the summary on disk, and quits AppleWorks.

Although it's easy to use a task file, creating one requires some background. I will describe how to create task files in Chapter 6.

The Programming Language

Unbeknownst to many UltraMacros users, UltraMacros also offers a true AppleWorks programming language. The language has its own set of variables, the capability for loops, a powerful if-then-else logic command set, and other sophisticated features that are common to programming languages, but are unusual in a macro program.

The UltraMacros programming language supports more than 50 commands, in addition to the standard commands already available in AppleWorks. Chapters 4 through 17 describe the UltraMacros programming language and teach you how to use that language. You will find that this powerful language will let you do spectacular things with AppleWorks.

Conclusion

In this chapter I introduced some of the features of TimeOut UltraMacros. Next, you will learn how to install UltraMacros on your working copy of AppleWorks.

2: How to Install UltraMacros

This chapter describes how to install UltraMacros and its accessory programs on your working copy of AppleWorks.

ok for
AppleWorks versions
less than 3.0

Every TimeOut enhancement consists of two components: The TimeOut program that installs on your working copy of AppleWorks, and a series of modules that add features to AppleWorks.

TimeOut's duties include loading applications such as SuperFonts, Graph, SideSpread, Macro Compiler, and Macro Options. Then you can call them from inside AppleWorks by pressing the Open-Apple-Escape Key and selecting from a menu.

No matter which TimeOut enhancement you buy, you must first install TimeOut in AppleWorks. Installing TimeOut is a one-time process. Once you install TimeOut, you can add additional TimeOut modules to your collection of applications; TimeOut will automatically recognize your new enhancements.

The UltraMacros package includes three components: (1) the TimeOut program that is on every TimeOut disk, (2) the UltraMacros program itself, and (3) a series of TimeOut modules that compile macros, display macros, and add the other features available from UltraMacros, such as adjusting the mouse response and the cursor blink rate.

While the TimeOut program modifies AppleWorks and takes some desktop space when you load AppleWorks, the UltraMacros program does not change your disk copy of AppleWorks. UltraMacros renames your disk copy of AppleWorks from APLWORKS.SYSTEM to APLWORKS.SYS, but makes no other changes to your disk copy. Instead, UltraMacros installs itself in AppleWorks each time you run it. In addition, since important UltraMacros abilities such as Macro Compiler and Macro Options are TimeOut applications, UltraMacros uses little or no desktop memory, thus leaving more room for the many features built into the program.

Installing UltraMacros: An Overview

Installing TimeOut and UltraMacros is a four step process:

1. Prepare backup copies of your UltraMacros and AppleWorks disks.
2. Prepare a TimeOut Applications Disk that contains your TimeOut modules.
3. Install TimeOut and UltraMacros on your working copy of AppleWorks.
4. Configure TimeOut and UltraMacros for your own system.

Work on Backup Copies

Start by making backup copies of both your working AppleWorks disk(s) and your UltraMacros disk. If you do not have a disk copy program, boot your computer with the UltraMacros disk and use the backup option available in the TimeOut Installer. Follow these steps to access that copy program:

1. Boot your computer with the UltraMacros disk and press the Return Key when requested.
2. With the TimeOut screen on your display, select choice #1, "Continue with the TimeOut Installation".
3. With the Installation Menu on the screen, select choice #2, "Manual".
4. With the Manual Installation Menu on the screen, select choice #2, "Make a backup disk". Then follow the on-screen prompts to make backup copies of your UltraMacros and working AppleWorks disks. If you use 5.25-inch disks, make certain you back up both sides of the disk.

Where to Put Your Applications

If you use 5.25-inch disks, you should collect all your TimeOut modules on a single disk. If UltraMacros is your only TimeOut disk, you can use your backup copy of UltraMacros as your TimeOut Applications Disk. If you own additional TimeOut modules, you should format a blank disk and use either the TimeOut Installer or a file copy program to copy all the files with names starting with "TO." onto that disk. This is your "TimeOut Applications Disk". You can format the disk and copy files from the TimeOut Installation Menu described above. Assign the name /TIMEOUT to this disk during the formatting process.

If you use 3.5-inch disks, you should use the TimeOut Installer or a file copy program to create a subdirectory named /TO on your AppleWorks disk. Then you should copy all your TimeOut modules into that subdirectory. (Information about ProDOS subdirectories is beyond the scope of this book. For more information about subdirectories, see the article entitled "How to Install an Operating System and Organize Your Hard Drive" in the June 1989 issue of the ***AppleWorks Forum***.)

Hard disk users should establish a subdirectory called /TO within their AppleWorks directory and copy all the TimeOut modules into that subdirectory.

Installing TimeOut and UltraMacros

Although TimeOut and UltraMacros are separate enhancements, the TimeOut Installer will install both programs at the same time.

Follow these steps:

1. Boot your computer with the UltraMacros disk.
2. Press the Return Key to skip past the startup screen.

3. The next screen provides some information about using the menus. Once again, press the Return Key.

The next step is to decide whether you want to use automatic or manual installation.

As its name implies, automatic installation automatically installs TimeOut and UltraMacros onto your copy of AppleWorks. You simply follow the prompts and answer the questions that appear on the screen.

Because automatic installation is a subset of manual installation, I will describe the manual procedures in some detail. Then I will describe the differences in the procedures for automatic installation. Proceed as follows:

4. Select "Manual" from the Installation Menu. TimeOut will present the following Manual Installation Menu:

    1. Read manual updates
    2. Make a backup disk
    3. Format a disk
    4. Catalog a disk
    5. Create a subdirectory
    6. Install TimeOut
    7. Copy applications
    8. Quit

5. Select choice #1 and read the manual updates. These notes tell you about any changes in the UltraMacros manual. The update information is in an AppleWorks file called "Notes" which you can load into AppleWorks. I suggest you keep a printed copy of the update with your UltraMacros manual.

6. If you have not backed up both your UltraMacros disk and your working copy of AppleWorks, use option #2 on the Manual Installation Menu to make those backups.

7. If you use 5.25-inch disks and have not prepared a separate Applications disk, format that disk now. Then use the Copy Applications option on the Manual Installation Menu to copy all the files with names starting with "TO." onto that disk.

    If you use 3.5-inch disks, insert a copy of your working AppleWorks disk and select choice #5 to create a subdirectory on this disk. Call the subdirectory /TO. Then select choice #7 and copy the TimeOut modules onto your AppleWorks disk.

    If you use a hard drive, select choice #5 and create the subdirectory /TO in your AppleWorks directory. Then select choice #7 and copy the TimeOut modules into the /TO subdirectory on your hard drive.

Install TimeOut

8. Now it is time to enhance your copy of AppleWorks with the TimeOut program and install UltraMacros on the AppleWorks disk. Select choice #6, "Install TimeOut".

9. The "Sort Menu?" option asks if you want TimeOut to alphabetize the list of modules on the TimeOut Menu. If you indicate "No", the TimeOut Menu lists your TimeOut modules in the sequence they appear on your Applications Disk. Enter whichever option you prefer and press the Return Key.

10. TimeOut asks if you will have more than one Applications Disk. Under most circumstances, you will answer "No".

11. Now you must specify where to store your TimeOut modules. I suggest you enter the following settings:

    A. Owners of a single 5.25-inch drive: Select "Slot/Drive" and choose Slot 6, Drive 1 as the location of your Applications Disk. When you boot up AppleWorks,

TimeOut will indicate it is getting errors trying to read the Applications Disk. You can then insert your TimeOut Applications Disk and choose "Try again".

B. Owners of two 5.25-inch drives: Select "Slot/Drive" and choose Slot 6, Drive 2 as the location of your Applications Disk. Insert the Applications Disk in Slot 6, Drive 2 when you start AppleWorks; TimeOut will read your disk at startup. Remove the drive when you want to load files onto the desktop, but re-insert the Applications Disk before you run a TimeOut application.

C. Owners of 3.5-inch drives: Select "ProDOS Pathname" and enter the pathname /APPLEWORKS/TO as the path to the TimeOut modules on your AppleWorks disk.

D. Owners of hard disk drives: Select the "ProDOS Pathname" option and specify the path to the /TO subdirectory.

Where is AppleWorks?

12. The Installer then asks the current location of your AppleWorks disk. The program needs to find AppleWorks so it can add TimeOut to the appropriate AppleWorks program files. Respond by telling the installer where AppleWorks resides at this moment, not where it will be when you run AppleWorks.

    A. Owners of a single 5.25-inch drive: Indicate AppleWorks is in Slot 6, Drive 1. You will have to do some disk swapping later.

    B. Owners of two 5.25-inch drives: Insert your AppleWorks Startup Disk in drive 2 and indicate AppleWorks is in Slot 6, Drive 2.

    C. Owners of one 3.5-inch drive: Select the "Slot/Drive" option and indicate that AppleWorks is in the 3.5-inch

drive (usually Slot 5, Drive 1). You will have to swap disks during the installation process.

D. Owners of two 3.5-inch drives: Insert your AppleWorks disk in the second drive and indicate the slot and drive location of the program disk, usually Slot 5, Drive 2.

E. Hard drive owners: Specify the ProDOS Pathname to the file APLWORKS.SYSTEM.

13. Insert your AppleWorks disk in the appropriate drive and press the Return Key. If you use the same drive for both TimeOut and AppleWorks, the Installer will tell you when to swap disks.

14. The Installer displays the message "Installing TimeOut" as it modifies your copy of AppleWorks and adds the file ULTRA.SYSTEM to the disk. Then the program displays the message "TimeOut and UltraMacros are successfully installed" (earlier versions of the Installer display the message "TimeOut is successfully installed"). If the Installer encounters a problem installing TimeOut or UltraMacros, the program displays the reason. Two common problems are:

    A. Disk is write-protected: The Installer needs to write on the disk; you cannot install TimeOut or UltraMacros on a write protected disk. Remove the write protection from your AppleWorks disk and press the Return Key.

    B. Disk is full: If you get a "Disk is full" message, remove some files from your working AppleWorks disk. This error will not occur if you use a fresh copy of AppleWorks.

15. Now quit the Installer. This brings you to the ProDOS Program Selector (Bird's Better Bye) which lets you choose the system file to run next.

Automatic Installation

Automatic installation is similar to manual installation, except that TimeOut makes more assumptions and asks fewer questions.

Follow these steps to use the automatic installation procedures:

1. Select Automatic Installation from the Installation Menu.
2. TimeOut asks if you made a backup copy of the UltraMacros disk. If you have a backup, select "Yes", otherwise choose "No". If you do not have a backup, the installation program will help you make that backup now. This will require some disk swapping for single disk drive users.
3. TimeOut presents this short menu:

    1. Read manual updates
    2. Install TimeOut
    3. Quit

    Select "Read manual updates" and read the latest notes about UltraMacros. These notes are in an AppleWorks word processor file called "Notes" on the UltraMacros disk. Later, you should bring this file onto the AppleWorks desktop and print a copy of the notes to keep with your UltraMacros manual.

4. Select "Install TimeOut" to install TimeOut on your working copy of AppleWorks. The Automatic Installer assumes you want sorted menus and that you have only one TimeOut Applications Disk.
5. The Installer asks the location of your Applications Disk and AppleWorks disk. Respond as described for the manual installation process.
6. TimeOut installs itself on your working AppleWorks disk and then assumes you want to copy the TimeOut modules

onto a TimeOut Applications Disk. On-screen prompts guide you through this process.

7. Select the "Quit" option to quit the Installer.

Testing Your Installation

Now test your installation. Follow these steps:

1. A. 5.25-inch floppy disks: Boot your computer with the TimeOut-enhanced copy of AppleWorks in Drive 1 and the TimeOut Applications Disk in Drive 2. If you have a single drive, boot your computer with AppleWorks and insert your Applications Disk when TimeOut indicates it is getting errors trying to read your Applications Disk. Then select "Try Again".

    B. 3.5-inch disks: Boot your computer with your enhanced AppleWorks disk.

    C. Hard drive: If you use a program selector such as Easy-Drive or ProSEL, specify that you want to run ULTRA.SYSTEM, not APLWORKS.SYSTEM or APLWORKS.SYS to launch AppleWorks. Then launch AppleWorks.

2. The UltraMacros title screen should appear for a few moments, followed by the standard AppleWorks startup screen. Continue with your normal startup procedures and the TimeOut title screen will appear.

    If you get errors when booting from a 3.5-inch disk drive or hard disk, TimeOut cannot find the TimeOut modules in the location you specified. Check to make certain the TimeOut modules are in the correct subdirectory. If they are not, copy them there now. If they are already in the /TO subdirectory, you must re-install TimeOut and specify the correct location for the TimeOut modules.

3. When TimeOut finds your applications, it reads basic information from each module and displays the name of each application. If the name of a module does not appear on the screen, check to determine if you copied the appropriate TO. files onto your Applications Disk or subdirectory.
4. After TimeOut finishes loading, the AppleWorks Main Menu will appear on the screen. Users of single 5.25-inch disk drives will need to change disks during this boot-up process.
5. Press Open-Apple-Escape to invoke the TimeOut Menu. The number of items that appear on this menu depends on the number of TimeOut modules on your Applications Disk. If you use only UltraMacros, the TimeOut Menu will include at least the following:
    1. Data Converter
    2. Help 3.0
    3. Macro Compiler
    4. Macro Options
    5. Macros Ultra
    6. Utilities
6. Help 3.0 and Macros Ultra contain UltraMacros reference material you will find useful. If those choices appear on your screen, select either one now and read the material.
7. The TimeOut Menu can only display 30 modules at a time. If you have more than 30 TimeOut modules, TimeOut automatically sets up additional menus. Press the Tab Key with the TimeOut Menu on the screen and you will see the next set of TimeOut applications. Press Open-Apple-Tab to back up to the previous TimeOut Menu.
8. Press the Escape Key to return to AppleWorks.
9. Next, you will test if UltraMacros functions correctly on your system. Create a new AppleWorks word processor document and issue a Solid-Apple-' (Solid-Apple-Single Quote). UltraMacros should type the current date on the screen. If it does not, UltraMacros might not be installed correctly. Quit AppleWorks and then re-launch AppleWorks either by booting your computer with your TimeOut-modified copy of AppleWorks or by launching the file ULTRA.SYSTEM. Watch for the UltraMacros startup screen. If it does not appear, review the installation procedure and re-install UltraMacros on a new copy of AppleWorks.
10. Now you will use the Macro Options module to customize your copy of AppleWorks. Press Open-Apple-Escape to return to the TimeOut Menu and select "Macro Options". Then select "Other Activities". Explore the choices on this menu; they let you customize your UltraMacros-enhanced copy of AppleWorks.
11. Press the Escape Key repeatedly until you return to AppleWorks or until UltraMacros asks you if you want to save your changes.

Conclusion

Once you reach this point, you've successfully installed UltraMacros and TimeOut onto your AppleWorks disk. Now you are ready to learn how to use UltraMacros. In the next chapter I will start that process by describing how to create your own keyboard macros.

3: How to Create Keyboard Macros

This chapter describes how to create and use keyboard macros.

Now that you know how to use the commands and macros built into TimeOut UltraMacros, it is time to create your own macros.

Different Types of Macros

There are two ways to construct a macro:

1. By telling UltraMacros to memorize your keystrokes, or
2. By entering the macro as text in a word processor document and using the UltraMacros compiler to convert that document into a macro.

Keyboard macros are easy to construct, but are not as powerful as compiled macros. In this chapter I will describe how to develop and save keyboard macros. I will discuss compiled macros in later chapters.

How to Create a Keyboard Macro

Boot up an UltraMacros-enhanced copy of AppleWorks, and you are ready to record keystrokes and create a keyboard macro.

In this example, you will create a keyboard macro that returns you to the AppleWorks Main Menu from anywhere within AppleWorks. Follow these steps:

1. Enter <oa-X> to tell UltraMacros you want to record a macro.
2. The "Select macro key" prompt will appear. You must "assign" the macro to a key combination so you can later "call" or run the macro by pressing those keys. Press the capital letter "T" to indicate that you want to record the macro as <sa-T>.

   You can also assign macros to Both-Apple Key combinations. To create a Both-Apple macro, hold down the Open-Apple Key as you respond to the "Select macro key" prompt. (You do not need to hold down the Solid-Apple

Key; UltraMacros automatically assumes you pressed the Solid-Apple Key when you assign a name to a macro.)

If you change your mind and do not want to record the macro, press the Escape Key.

UltraMacros will check if the key you selected is already used to call another macro. If it is, UltraMacros issues the message:

```
Replace local macro  T?  No  Yes
```

This message indicates that the keystroke combination you want to assign to the new macro is already used to define another macro. If you respond "yes", UltraMacros will replace the former definition with a new set of memorized keystrokes. UltraMacros comes with a series of useful macros already installed in the system, so don't be surprised if you get the "Replace local macro?" message even though you never assigned a macro to that particular key combination. *[Ed:* Figures 1.1, 1.2, *and* 1.3 *in Chapter 1 list the keys already used by UltraMacros.]*

UltraMacros also tells you whether the previous macro is a "local" macro or a "global" macro. A local macro works in only one AppleWorks module. For example, you can define a local macro that only works in the spreadsheet module. That macro will not work when you use the AppleWorks word processor or data base modules. A global macro works in all three modules.

All keyboard macros are automatically global. If you want to develop local macros, you have to create an UltraMacros source file and compile it. The next chapter will describe those procedures.

3. The lower-right hand corner of the screen (where the "oa-? for Help" message usually appears) displays the message "Recording T". UltraMacros will memorize every keystroke

and store them in macro T. Enter your keystrokes slowly ... you do not want to make a mistake now.

If you use AppleWorks 2.x, the cursor appears frozen. You may not see the cursor at this moment, but it will reappear after you press a key. When the cursor returns, it does not flash. This means that UltraMacros is recording your keystrokes.

If you use AppleWorks 3.x, UltraMacros blinks the cursor and clicks the speaker each time you press a key.

If the cursor is flashing or if AppleWorks 3.x stops clicking the speaker, your Macro Table is full and UltraMacros cannot capture your keystrokes. I will describe how to resolve this problem later in this book. For now, if your Macro Table is full and if you still want to record your keystrokes, find a macro you don't use and assign your new macro to that keystroke combination. That will replace the earlier macro with the new set of keystrokes.

To create a macro that automatically returns you to the AppleWorks Main Menu from anywhere in AppleWorks, enter an <oa-Q> and press the Escape Key.

4. If you use AppleWorks 2.x, enter a <ctrl-@> (a Control-Shift-2) to tell UltraMacros to stop memorizing your keystrokes. Do not type an <oa-ctrl-@>; that enters a Control-@ into your macro.

   With AppleWorks 3.0 or later, a second <oa-X> command tells AppleWorks to stop recording your keystrokes.

   The message "Done macro T" will appear at the lower-right hand corner of the screen to indicate that recording has stopped.

You have now created your first macro; type <sa-T> and it will transport you to the AppleWorks Main Menu. This macro is

handy; it works almost anywhere inside AppleWorks and TimeOut (the exception is DeskTools' calculator).

Make Your Macro Permanent

At this moment, your keystroke macro is temporary; it is active only during the current AppleWorks session. If you quit AppleWorks, this macro will not be available when you return to the program.

You can make your macro "permanent" by adding the macro to the set of "default macros"; the macros that are automatically available when you start AppleWorks.

Follow these steps to make your new macro permanent:

1. From anywhere within AppleWorks, press <oa-Escape> and select "Macro Options".
2. Select choice #3, "Save macro table as default set". This option will save all the currently active macros (including the new global macro you just entered) as the default macro set that will be active when you return to AppleWorks.
3. If you use AppleWorks 3.x, answer "No" to the "Activate auto startup macro" prompt.

Conclusion

You should now know how to create a keystroke macro and make that macro permanent.

4: Introduction to Compiled Macros

This chapter explains how to use AppleWorks' word processor and the UltraMacros compiler to create more complex macros.

In the previous chapters you learned about the features UltraMacros adds to AppleWorks and how to prepare and use keyboard macros. This chapter introduces the powerful programming language built into UltraMacros.

By the end of this chapter, you should know how to write and "compile" an UltraMacros macro.

Compiled Macros vs. Keyboard Macros

Although keyboard macros are easy to create, they suffer from three limitations:

1. Keyboard macros are "global": you cannot develop keyboard macros that work in only one AppleWorks module.
2. It is difficult to create keyboard macros that require numerous keystrokes.
3. Keyboard macros cannot call for keyboard input nor can they include any "branching logic".

Although compiled macros are more difficult to construct, they offer the following advantages over keyboard macros:

1. Compiled macros are more powerful: They can look at what appears on the AppleWorks screen and make decisions based on rules you describe.
2. Compiled macros can perform operations not possible with keyboard macros. For example, a compiled macro can pause, wait for a keystroke entry, then automatically resume operation.
3. Compiled macros are easy to describe and share with others.

Getting Started

It is best to conceptualize macro writing as a five-step process:

1. Determine the steps required to do the job.

2. Break down each task into the keystrokes required to perform that operation.
3. Prepare an AppleWorks word processor document with the UltraMacros commands that correspond to those keystrokes.
4. Use the UltraMacros compiler to compile the macro.
5. Test the macro and correct any mistakes you discover.

Like all programming languages, UltraMacros has a set of commands and a structure you must follow. Since the commands correspond to familiar AppleWorks operations, it is relatively easy to learn and use this programming language.

Every macro consists of "text" and "tokens" you type into an AppleWorks word processor document. All AppleWorks activities can be represented by one or the other as follows:

Text: Anything you type at the keyboard other than AppleWorks and UltraMacros commands. Text includes the upper- and lowercase letters A through Z, numerals, and punctuation.

Tokens: Abbreviations for an UltraMacros or AppleWorks command, or any non-text keystroke. For example, the token <del> means "Press the Delete Key", <rtn> means "Press the Return Key", and <oa-P> means "Enter an Open Apple-P". Tokens begin with the "<" character and end with the ">" character.

Figure 4.1 lists the most frequently used tokens recognized by UltraMacros.

Figure 4.2 shows you how to use these tokens in a macro. The macro in *Figure 4.2* takes you to the Main Menu from anywhere within AppleWorks and automatically creates a new word processor document called "Untitled".

Figure 4.1: Common UltraMacros Tokens

Keystroke Tokens

<del>	Delete Key
<esc>	Escape Key
<rtn>	Return Key
<tab>	Tab Key
<left>	Left Arrow Key
<right>	Right Arrow Key
<up>	Up Arrow Key
<down>	Down Arrow Key
<spc>	Space Bar

Other Keystroke Combinations

These abbreviations, when combined with a keyboard character, form tokens representing several possible keystrokes and commands:

<oa- >	Open Apple Key
<sa- >	Solid Apple Key
<ba- >	Both Apple Keys
<ctrl- >	Control Key
<ba-ctrl- >	Both Apple Keys and Control Key

Examples:

<oa-p>	Print
<sa-h>	Solid Apple and H Keys
<oa-right>	Open Apple and Right Arrow Keys
<sa-ctrl-c>	Solid Apple Key, Control Key, and C

Beginnings, Endings, and Formats

Every macro must follow a carefully defined sequence. Here is the sequence for the simplest form of a macro:

1. The word "start" appears on a line by itself at the beginning of the word processor file.
2. A token to specify the keystroke that will "call" the macro.
3. A colon (:).
4. A token to specify the AppleWorks module(s) in which the macro will function.
5. Tokens and/or text constituting the macro.
6. An exclamation mark to indicate the end of the macro.

Now, you will prepare the macro in *Figure 4.2*.

Figure 4.2: Anatomy of a Macro

The macro below contains each of the elements necessary for a complete macro.

```
start
<sa-w>:<all>
<oa-q><esc>
<rtn>3<rtn><rtn>Untitled<rtn>!
```

Token

Text

The word "start" must appear at the beginning of any word processor file that contains macros.

Every macro begins with three elements:
- The token of the key that invokes the macro.
- A colon.
- A token that indicates where the macro can be used.

Tokens and text constitute the body of the macro.

An exclamation mark signals the end of the macro.

Writing a Macro

1. Boot a copy of AppleWorks enhanced with UltraMacros, and create a new word processor document from scratch. Name the document "Tutorial Macro". Remember that you write macros in word processor documents, so you will use this word processor document for the sample macro.

2. Type the word "start" on a line by itself and press the Return Key. *[Ed: In this tutorial, quotation marks appear around text you should enter. Do not type the quotation marks.]*

3. Every macro must begin with the token which names the key you will use to invoke the macro. You will invoke this macro by typing a Solid-Apple-W, so type "<sa-w>". Remember to type the left and right brackets; they specify you are entering a token.

4. Type a colon.

5. Type a token to indicate where the macro can be used in AppleWorks. The tokens that indicate the "domain" of a macro are as follows:

   <all> - everywhere in AppleWorks.

   <awp> - in the word processor.

   <adb> - in the data base.

   <asp> - in the spreadsheet.

   <ato> - in a TimeOut application (<ato> only works with AppleWorks 2.x. If you use AppleWorks 3.0 or later, see the <id#> command in Chapter 14 for information about how to write macros that work with TimeOut).

   <asr> - a subroutine macro. (<asr> only works with AppleWorks 3.x. See Chapter 15 for information about how to use <asr>.)

   Since you want the macro to be available at any time, type the token "<all>", then press the Return Key to start a new line on your screen. Although this Return has no impact on the operation of your macro, it makes the macro easier to understand and edit.

6. Next, you will enter the portion of the macro that returns you to the AppleWorks Main Menu. Since the keystrokes Open-Apple-Q followed by Escape take you to the Main Menu from anywhere within AppleWorks, you will enter the tokens representing those keystrokes.

   Type the tokens "<oa-q><esc>" and press the Return Key. Your screen should look like this:

```
start
<sa-w>:<all>
<oa-q><esc>
```

Now you will tell AppleWorks to "Add Files to the Desktop", add a new word processor file, and name that file "Untitled".

7. Issue an Apple-Q command, return to the AppleWorks Main Menu and write down the keystrokes necessary to create a new document with the name "Untitled". The keystrokes are as follows:

   ```
   Return, 3, Return, Return, Untitled, Return
   ```

   Note that each "Return" will be replaced by the token "<rtn>", but the number "3" and the word "Untitled" are text.

8. Issue an Apple-Q and return to the document "Tutorial Macro" that contains the macro. Add the tokens and keystrokes necessary to complete the macro. That is, enter the following line of text:

   ```
   <rtn>3<rtn><rtn>Untitled<rtn>
   ```

9. Type an exclamation mark to indicate the end of the macro. Your screen should look like this:

   ```
   start
   <sa-w>:<all>
   <oa-q><esc>
   <rtn>3<rtn><rtn>Untitled<rtn>!
   ```

10. Issue an Apple-S command to save this macro as a word processor file on your AppleWorks data disk. If anything goes wrong, you can load that file onto the desktop and edit the file.

Compile the Macro

Now you must "compile" the macro. This process converts the word processor document into a form that UltraMacros can use. With the word processor file containing the macro on the screen, proceed as follows:

1. Hold down the Open-Apple and Escape Keys to call the TimeOut Menu.
2. Select "Macro Compiler".
3. If you use AppleWorks 2.x, press the Return Key three times to accept the default settings for the Macro Compiler. If you use AppleWorks 3.x, enter an Open-Apple-Return with the "Compile a new set of macros" choice on the screen. That compiles the macro and returns you to the word processor file if the macro compiles correctly.

The compiler looks at the word processor document and converts your tokens and text into UltraMacros code. The compiler then tells you whether or not it encountered errors in the syntax of the macro. The Macro Compiler should report "Compiling macro W" and then display the message "No errors".

Your new macro is now "active". Press the Space Bar to return to the word processor, then press the Solid-Apple and W keys to invoke the macro. The macro quickly returns you to the Main Menu and creates a new word processor document called "Untitled".

Unfortunately, this macro is temporary and will disappear the next time you boot up AppleWorks. This macro also temporarily replaces all the macros built into UltraMacros. We will solve these problems in the next chapter.

Conclusion

You have now created and used your first compiled macro.

5: Understanding Compiled Macros

This chapter describes how to add custom macros to the set of default macros that come with UltraMacros.

Previous chapters described the advantages of compiled macros over keyboard macros, and how to write and compile your own macros. You learned that when you compile a macro you lose UltraMacros' built-in macros. In addition, when you quit AppleWorks, any macros you write are replaced by the macros originally built into UltraMacros. Fortunately, there are ways to avoid both these problems. This chapter describes how to store your macros for later use and how to write macros that become active when you start AppleWorks.

UltraMacros Lexicon

Here are some terms you will see in this chapter:

Macro: A collection of tokens and text which describe a command you want to add to AppleWorks. A "macro file" is any word processor document that contains macros.

Compiled Code: The machine-readable program created by the UltraMacros Compiler from a macro file. You cannot view or edit compiled code directly.

Current Macro Set: The set of compiled UltraMacros commands which are in memory and available for use.

Default Macro Set: The set of compiled UltraMacros commands which is stored on disk and is automatically loaded into memory when you start AppleWorks.

You can use the macro you wrote in Chapter 4 to explore these terms. Try this exercise:

1. Boot up a copy of AppleWorks enhanced with UltraMacros. When the UltraMacros startup screen appears, UltraMacros loads the default set of macros from disk into memory.

2. With the AppleWorks Main Menu on the screen, issue a Solid-Apple-A command. The command works because

Solid-Apple-A is part of the current set of macros that is created from the default set of macros at startup.

3. If you saved your Solid-Apple-W macro from Chapter 4 as a file on a disk, call it onto the screen now. If not, re-key the macro (see *Figure 5.1*) into a new word processor file.

Figure 5.1: Solid-Apple-W Macro

Enter this macro and save it as a word processor document:

```
start
<sa-w>:<all>
<oa-q><esc>
<rtn>3<rtn><rtn>Untitled<rtn>!
```

4. Press Solid-Apple-W. Nothing happens because the current set of macros has no Solid-Apple-W command. Macro files do not become new AppleWorks commands until you compile them.

5. Follow the steps outlined in the previous chapter and compile the Solid-Apple-W macro. When the compilation is complete, a message appears telling you that your macro replaced the current macro set.

   Now exit the macro compiler and press Solid-Apple-A. Nothing happens. However, when you press Solid-Apple-W, the screen flickers and you get a new word processor file named "Untitled". This occurs because UltraMacros replaces the current set of macros when you compile a macro. Thus, Solid-Apple-W is the only command in the current macro set. The default macros are still stored on disk, but you cannot restore the default macro set unless you re-boot AppleWorks.

6. Issue an Apple-S command to save the word processor file that contains the Solid-Apple-W macro. Quit AppleWorks, then boot up AppleWorks again.

How to Access the Current Macro Set

As AppleWorks boots, the default macros again load into memory from disk and become the current macro set. You can demonstrate this by pressing Solid-Apple-W. Nothing happens because your new macro is no longer in memory. However, the Solid-Apple-A macro functions again, as do all the other built-in macros.

Since you cannot edit macros that are in compiled form, you cannot directly edit the current macro set. However, you can list the current set of macros into a word processor file and edit the macros in that file. Then you can compile the edited word processor document to replace the current macros with your custom set.

Here is how to generate a word processor file from the current set of compiled macros:

1. Create a new word processor document from scratch and assign any name to the file.

2. With the blank document on the screen, call the TimeOut Menu and select "Macro Compiler". Choose the option "Display Current Macro Set". The word processor document will now contain UltraMacros code.

The document on the screen should include some familiar elements. There are words that look like part of a letterhead, tokens like <awp> and <rtn>, and the exclamation marks which signal the end of a macro. However, the listing is in a somewhat different format than you saw in the last chapter. For example, instead of the format:

```
<sa-p>:<awp>
<oa-p><rtn><rtn><rtn>!
```

you see

```
P:<awp><oa-p rtn rtn rtn>!
```

The macros on the screen are in a condensed format that differs from the syntax you learned in Chapter 4. Note these differences:

1. The part of the macro which associates a macro with a keystroke is abbreviated when the keystroke is a Solid-Apple combination. For example, the token <sa-p> at the beginning of this macro becomes "P".
2. "<" and ">" marks only appear at the beginning and end of a series of tokens. UltraMacros replaces the unnecessary symbols with colons or spaces.
3. The macro is condensed; there are no lines or spaces to make the macro easier to read.

The Macro Compiler displays the current set of macros in an abbreviated format. Even though this listing looks different from the file originally used to create the macro, it is functionally equivalent and syntactically correct. However, this efficiency comes at a loss of clarity. Output generated by the Macro Compiler is usually more difficult to read than the original version of a macro.

How to Customize the Current Macro Set

The word processor file on your screen contains a representation of the current macro set. You will now modify those macros. Follow these steps:

1. Use the Apple-N command to name this file "Original Set". Then issue an Apple-S command to save the file on disk.
2. Create a new word processor file from scratch called "Experiments". You will use this document to explore the current macro set.
3. With "Experiments" on the screen, press Solid-Apple-N. The name "Heather Brandt" appears.

Randy Brandt is the author of UltraMacros, and Heather is Randy's daughter. You will modify the Solid-Apple-N macro so it displays your name, not Heather's.

4. Use Apple-Q to return to the word processor file "Original Set".
5. Use Apple-F to find the string "Heather". AppleWorks will display the following macro:

```
N:<awp>Heather Brandt!
```

6. Replace Heather's name with your own.
7. Issue an Open-Apple-Escape, select the Macro Compiler, and compile the file. The compiled version of this modified file is now the current macro set.
8. Use Apple-Q to return to the "Experiments" document and type a Solid-Apple-N again. Your name will appear.
9. Use Apple-N to change the name of the "Original Macros" file to "My Macros", and save the file.

You can use these procedures to customize other macros in the default set. For example, the Solid-Apple-B macro starts a memo in a standard format. The Solid-Apple-J macro types a return address at the insertion point. You can scroll through this file and change the text of existing macros to your taste. When you want to see the effect of your change, re-compile the file "My Macros", use Apple-Q to return to the "Experiments" file, and test the macro.

Issue an Apple-S command to save the file "My Macros" whenever you make a significant change.

How to Add Your Own Macros

Now that you can modify the macros built into UltraMacros, I will describe how to add new macros to the existing macro set. The procedure is to write a macro in a word processor file, test

the macro, then use the clipboard to move the macro definition into a larger set of macros.

Follow these steps:

1. Create a new word processor file that contains the macro you want to add to UltraMacros. This could be a macro of your own design, or one listed in the ***AppleWorks Forum***. Issue an Apple-S command to save the word processor file on disk.
2. Use the Macro Compiler to compile the new macro. This replaces the current macro set, which you saved in the word processor file "My Macros".
3. Test the new macro. If necessary, edit the word processor file containing the macro, re-compile, and re-test the macro.
4. When the macro works as expected, use the Apple-C command to copy the macro text to the clipboard.
5. Use Apple-Q to return to the file "My Macros".
6. Make sure the key you want to use for your new macro does not conflict with one already in use. You can use AppleWorks' Apple-F command to search for the sequence that marks the beginning of a conflicting macro. In this example, search for the text "W:<". You will discover that Solid-Apple-W is not used by any of the macros built into UltraMacros.
7. Press Apple-9 to get to the end of the macro file.
8. Use the Apple-C command to copy the text from the clipboard to the end of the file "My Macros".
9. Use Apple-S to save the file "My Macros".

You can now compile the file "My Macros" and use all the built-in UltraMacros commands, your customized versions of the built-in macros, and the new Solid-Apple-W command.

How to Make Your Macros the Default Set

The word processor file "My Macros" is stored on disk, and the compiled version of those macros is in memory as the current set. However, the next time you boot AppleWorks, UltraMacros will load the default macro set from the disk; your customized set of macros will not be active. You would have to re-compile your custom set at the beginning of every session.

Here is how to convert the current set of macros into the default set that automatically becomes active when you start AppleWorks:

1. Modify the file "My Macros" so it contains the macros that meet your specifications.
2. Compile the file "My Macros". Your custom set loads into memory and becomes the current set.
3. Call the TimeOut Menu and select "Macro Options".
4. Select choice #3, "Save Current Macro Set as the Default Set".
5. Answer "No" when UltraMacros asks if you want to activate the startup macro.

UltraMacros tells you that the file ULTRA.SYSTEM now holds the customized macros. Now, if you quit AppleWorks, your custom set of macros will be active when you return to AppleWorks.

How to Annotate a Macro

Like most programming languages, UltraMacros lets you document your work. While there are many ways to add comments to a macro file, the convention is to include your comments within braces ({ }). These messages do not become part of the compiled macro.

Syntax Rules for Comments

Comments can appear after "start" in the file containing your macros or in the middle of a series of tokens. The correct format to include comments in the middle of a macro is " ... <esc {comment} rtn>... ". Comments may not contain exclamation marks, "<" symbols, or ">" symbols.

Your comments must appear between "<" and ">" brackets within the body of the macro. Otherwise, UltraMacros treats the comment as text. For example, the placement of the comment in macro <sa-a> below is correct. The comment in macro <sa-b> is not placed correctly.

```
<sa-a>:<all: {This comment is correctly placed} oa-q:esc>!

<sa-b>:<all> {This comment is not correct} <oa-q:esc>!
```

You can also insert comments after the word "start" and before the first token. Thus, you can annotate the macro in the previous chapter as follows:

```
start                   { Solid-Apple-W adds a blank word      }
                        { processor file to the desktop        }

<sa-w>:<all             { This macro is available everywhere   }
                        { in AppleWorks                        }

oa-q : esc              { Go to the Main Menu                  }

                        { Now, create a new file for the word  }
                        { processor called 'Untitled'          }

rtn>3<rtn><rtn>Untitled<rtn>!
```

Note that I inserted spaces (or tabs) to move the comments to the right-hand edge of the screen to improve the readability of the macro. However, as long as I follow the rule that comments must appear within a string of commands, UltraMacros ignores these spaces or tabs and handles these comments correctly.

When you use the macro compiler to display the current macro set, the screen shows the condensed version of the macro, without comments. For example, the macro above becomes:

```
W:<all><oa-q esc rtn>3<rtn rtn>Untitled <rtn>!
```

Although it is easy to envision the operation of this macro in its condensed form, imagine what happens when you create more complex macros. Be sure to save the annotated version of any macro file you intend to change in the future.

Conclusion

You now know how to convert the current macro set into a word processor document, how to customize those macros, how to add macros to the original set of macros, and how to replace the original macros with your enhanced macro set.

6: Introduction to Task Files

This chapter introduces task files, compiled versions of macro sets, that let you use different groups of macros with different jobs without re-compiling your macros.

You now know how to write, edit, and compile a macro, and how to customize the macros built into UltraMacros. By now, you probably have developed several macros to help with your work.

The previous chapter described how to use the AppleWorks word processor to store macro sets. While that chapter suggested that you store each macro set in a separate word processor document, that is an inefficient way to manage different sets of macros. Every time you want to change between macro sets, you must bring the appropriate word processor file onto the desktop and compile the macros stored in that file. For example, you might create a set of macros to help you manage a complex spreadsheet. Obviously, it is not efficient to compile the word processor file containing the spreadsheet macros every time you want to use that spreadsheet. In addition, you must re-compile the standard macro set when you return to other AppleWorks activities.

While it is logical to structure macros into customized macro sets, the need to repeatedly compile each set of macros discourages this type of organization.

Task Files

Fortunately, UltraMacros gives you a way to compile macro sets once and store the compiled macros for future use. That is the role of a "task file". A task file contains the compiled version of a set of macros.

Task files are an efficient way to store different macro sets. First, they occupy less disk space than the word processor file used to create the macro set. Second, they let you load macro sets into UltraMacros without forcing you to constantly recompile a word processor file. Thus, task files allow a smooth transition between different sets of macros.

You should understand task files even if you never develop personalized sets of macros because commercial UltraMacros enhancements often supply macros in task file format. In addition, if you are using AppleWorks 2.x, the programmers of these products expect to find your default macro set on disk as a task file.

How to Convert a Macro Set into a Task File

As explained in earlier chapters, only one set of macros can be active at a time. Each time you load a new macro set into UltraMacros, you replace the current set of macros. When you want to return to your standard set of macros, you must re-compile those macros from the word processor document in which you stored them.

Here is an exercise that has two purposes: First, it shows you how to convert a set of macros into a task file. Second, this exercise creates a task file that contains your standard macro set. When you complete this exercise, you will be able to load this task file directly into UltraMacros. That lets you avoid re-compiling the macros whenever you want to replace the current macro set with your standard macros.

Proceed as follows:

1. Boot an UltraMacros-enhanced copy of AppleWorks. The default set of macros will automatically load from disk and become the current set.
2. Enter an Open-Apple-Escape to call the TimeOut Menu and select "Macro Options".
3. Select choice #2 on the Macro Options screen, "Create a Task File".
4. UltraMacros now requests a name for the task file. Most commercial macros try to restore your standard macro set by looking for a task file called "DEFAULT.MACROS",

therefore you should assign that name to this task file; it contains your customized set of default macros.

(Task files follow the same naming conventions as any ProDOS file; they must start with a letter, can be up to 15 characters long, and cannot contain spaces or punctuation marks other than a period. In addition, task files cannot start with the letters "ULTRA".)

The Macro Options module now saves the current set of macros as a task file on the disk or subdirectory that contains the file ULTRA.SYSTEM. UltraMacros will prompt 5.25-inch disks users to replace their AppleWorks Program Disk with the Startup Disk. UltraMacros then displays the pathname of the new task file.

Your UltraMacros-enhanced AppleWorks program is unchanged by this process; it still contains your customized set of default macros as the current set. However, you now saved a binary file containing a compiled version of those macros on your AppleWorks Startup Disk. In the future, you can re-install that file as your current macro set without using UltraMacros' Macro Compiler.

Using Your New Task File

Now we will simulate a macro programming session. In this exercise you will create and compile a simple macro, thus replacing your default macro set. Then you will use the DEFAULT.MACROS task file to restore your standard set of macros. Proceed as follows:

1. Create a new word processor document and enter the following macro:

```
start
N:<awp>
This is a macro<rtn>!
```

This macro types the sentence "This is a macro".

2. Compile the macro.
3. Create a new word processor document and press Solid-Apple-N. The text "This is a macro" will appear in the document. This demonstrates that UltraMacros replaced your customized macro set with the new Solid-Apple-N macro.

Now you will use the DEFAULT.MACROS task file to restore the standard set of macros. Proceed as follows:

1. Issue an Apple-Escape and select Macro Options from the TimeOut Menu.
2. Choose "Launch a new Task" from the Macro Options Menu and select the file "DEFAULT.MACROS". This replaces the current macro set which contains only the Solid-Apple-N macro with your standard macros.

   If you use AppleWorks 3.x, UltraMacros offers you another option. Enter an Open-Apple-Return with "Launch a new Task" highlighted, and UltraMacros lets you enter the pathname to the task file. ULTRA.SYSTEM appears as the default pathname, and that file contains your original macro set, so you can either accept ULTRA.SYSTEM or enter a pathname to a different file.
3. Issue a Solid-Apple-N and your name appears in the word processor document. Your standard macro set, incorporated in the task file DEFAULT.MACROS, is now active. *[Ed: If the name "Heather Brandt" appears, you should review Chapter 5 "Understanding Compiled Macros", and customize the default macro set that comes with UltraMacros.]*

Special Qualities of Task Files

You probably know that ProDOS attaches a three-letter identifier to the disk catalog entry for every file. Text-only (ASCII)

files are labelled "TXT", AppleWorks data base files are labelled "ADB", and AppleWorks spreadsheet files are labelled "ASP".

System files get the label "SYS" and are special; they can be "launched" just like AppleWorks. "APLWORKS.SYS" and "ULTRA.SYSTEM" are two examples of "SYS" files. Task files are also "SYS" files. Thus, task files appear in the list of files presented by program selectors, such as Bird's Better Bye or Squirt. You can also open task files directly from the Apple IIGS Finder. In addition, you can load and run task files before you start AppleWorks.

These attributes add power to task files. For example:

1. You can decide which macro set will be active when you boot AppleWorks by launching the task file that holds the set you want. You do not have to start with the default set and then manually load a custom macro set.
2. A task file can hold macros that instruct AppleWorks to automatically load one or more files onto the desktop.
3. You can write a macro that loads selected files into AppleWorks, automatically updates each file, prints a report, then exits AppleWorks.

You will see examples of these features later in this chapter.

The Structure of Task Files

These three characteristics of task files place constraints on the structure of the word processor file you use to create a task file. Specifically, the first two macros listed in a task file carry special meaning, depending on the way you run the task file.

UltraMacros executes one of your first two macros every time you run a task file. If you use a program selector or the Apple IIGS Finder to launch a task file from outside AppleWorks,

UltraMacros automatically executes the first macro in the task file. Alternatively, UltraMacros executes the second macro in the task file when you load the file from within AppleWorks.

If you do not want to execute a macro when you load a task file, you can put two "dummy" macros at the beginning of the word processor file holding the macro code. The dummy macros do not perform any function other than to occupy space.

However, rather than constructing dummy macros, consider starting your task file with macros similar to those built into the UltraMacros default set. *Figure 6.1* depicts the beginning of that file. You can convert the UltraMacros default macro set into a task file because it includes two macros that let you launch the task file containing the default macros from within or outside of AppleWorks. If you launch the default macro set task file with a program selector or with the Apple IIGS Finder, UltraMacros runs the macro "Both-Apple-]" and displays "Default Macro Set Installed". If you launch the task file from within AppleWorks, UltraMacros runs the second macro, "Both-Apple-[". That macro exits the Macro Options menu and displays the same message.

Figure 6.1: First Two Macros from the Default Set

```
start

<ba-]>:<all><rtn rtn ba-[>!

<ba-[>:<all><msg ' Default macros installed. '>!
```

How to Use a Task File to Organize Your Work

Now that you know how to construct a task file, the following example demonstrates how to use these files.

Imagine that you are a teacher who uses AppleWorks to generate a mid-semester report for each student. You normally pre-

Figure 6.2: Sample Task File

This figure depicts three macros that load four files onto the AppleWorks desktop. These macros use advanced techniques described later in this book. The macros assume you have AppleWorks 2.x, a 3.5-inch or hard disk drive and an Apple IIGS or a clock card. If you use AppleWorks 3.x, delete two <rtn>'s from the macro <ba-]>.

```
<ba-]>:<all:rtn:rtn:rtn:rtn:
   goto ba-*>!
```

This macro runs when you launch the task file from outside AppleWorks.

```
<ba-[>:<all:oa-Q:Esc:rtn:rtn:
   goto ba-*>!
```

This macro runs when you launch the task file from within AppleWorks.

```
<ba-*>:<all:
   $Ø="Report.Merge":find:right:
   $Ø="Students":find:right:
   $Ø="Attendance":find:right:
   $Ø="Grades":find:right:
   rtn:oa-esc:
   $Ø="Macro Options":find:rtn:rtn:
   $Ø="Default.Macros":find:rtn>!
```

This is the main macro, which is called by the first two macros. It loads the required files onto the AppleWorks desktop in the same order as they appear in the Add Files Menu, then restores the default set of macros from disk.

pare these reports by booting up AppleWorks and loading four files onto the desktop: a data base file with the students' names and addresses, a gradebook template which shows students' mid-semester grades, an attendance spreadsheet, and a form letter with mail-merge areas.

You can use a task file to automate the loading of the files. Examine the sample macro in *Figure 6.2*.

Opening this task file from the Apple IIGS Finder or a program selector automatically invokes the first macro (Both-Apple-]). This macro pages through the AppleWorks startup screens and calls the main macro, Both-Apple-*.

Alternatively, if you load the task file from within AppleWorks, UltraMacros runs the second macro, Both-Apple-[.

This macro returns you to AppleWorks' Main Menu and invokes the main macro.

The main macro, Both-Apple-*, loads the four AppleWorks files onto the desktop and loads the task file DEFAULT.MACROS into memory. This leaves you in AppleWorks with four files on the desktop with the default macro set available for use.

Of course you can expand the role of this task file by enhancing it with custom macros for the gradebook files. For example, you can add macros which automate the printing, updating, and other manual operations that move data between files.

Summary

In summary, creating a new task file is a three-step process:

1. Create a word processor document listing the macro set you want to use in the task file.
2. Use the Macro Compiler to compile the macro set and store the compiled macros in memory; they become the current set.
3. Invoke Macro Options and create a new task file which contains the current set of macros.

You can use any task file either by "launching" that file before you enter AppleWorks or by making that file active by going to the Macro Options Menu when you invoke TimeOut from within AppleWorks.

Using AppleWorks 3.x, you can also launch a task file from within a macro. I will describe that process in Chapter 15.

Conclusion

You should now understand the purpose of task files and know how to create those files. You can use these skills to create your own task files and to manage task files available on commercial macro products.

7: UltraMacros Programming

This chapter introduces UltraMacros programming commands you cannot generate by capturing keystrokes.

You are at an important juncture in your understanding of UltraMacros. You now have the tools necessary to use the program to emulate AppleWorks keystrokes and commands. You can record keyboard macros, have a working knowledge of the structure and syntax of macros, can manage multiple sets of macros, and can create and manage task files.

Now you will learn how to write UltraMacros programs that get AppleWorks to do things that cannot be done from the keyboard. These programs can sense what AppleWorks displays, accept input from the person using the computer, display messages and instructions to the user, and make decisions based on criteria you determine.

UltraMacros programs add power and flexibility to AppleWorks. You can create "intelligent" spreadsheet templates which guide users through the entry of complex data, you can automate the process of backing up a data disk or hard disk, and you can add new commands and features to AppleWorks. The potential for UltraMacros-enhanced AppleWorks is impressive; advanced UltraMacros programmers can already get AppleWorks to control a video disk player, and there is even a macro that simulates an arcade-style video game.

Programming Requirements

This chapter will build upon the basics of UltraMacros that I described earlier. To write UltraMacros programs you must know how to compile and test macros, you must understand the syntax of a macro and the structure of a macro file, and you should understand the differences between tokens and text.

In this chapter I will describe the basics of writing an UltraMacros program. By the end of this chapter, you will know the structure of an UltraMacros program, how to display messages to the user, and how to accept user input.

UltraMacros Programming Basics

UltraMacros programs are macros that use commands that cannot be produced within AppleWorks. You type these macros into a word processor document just like other macros. Then you compile the macro and the program becomes available within AppleWorks.

UltraMacros programming involves four steps:

1. Decide what you want the macro program to do and how to do it.
   - What are the major tasks required?
   - What are the components of that task, and how should you handle errors and unexpected events?
   - What are the specific keystrokes and commands required for each task?
2. Create a macro within the word processor.
3. Test and debug the macro.
4. Compile the new macro set.

You will start with a macro you developed earlier, and will enhance that macro so it does things that are normally not possible with AppleWorks.

The macro in *Figure 7.1A* takes you to the Add Files Menu, indicates you want to create a new word processor document, and names that document "Untitled". You can create that macro either by capturing your keystrokes or by typing the macro into the word processor. *Figure 7.1B* depicts an enhanced macro that displays a message at the bottom of the screen, beeps the computer's speaker to get attention, and asks for user input.

Figure 7.1A: A Simple Macro

```
start
<sa-w>:<all>
        <oa-q><esc>
        <rtn>3<rtn><rtn>Untitled<rtn>!
```

Figure 7.1B: An "Intelligent" Macro

```
start
W: <all : oa-q : esc : rtn> 3 <rtn : rtn :
   msg ' Enter the name of the new word processor file and press Return' :
   bell :
   input :
   rtn :
   msg ''>!
```

Note: The macro in *Figure 7.1B* uses the abbreviated format for listing tokens. You should become familiar with reading and writing macros expressed in this format.

Additional UltraMacros Commands

UltraMacros gets much of its power from a series of commands that go beyond those available in AppleWorks. The macro in *Figure 7.1B* uses three of those commands, the <bell>, <msg>, and <input> commands.

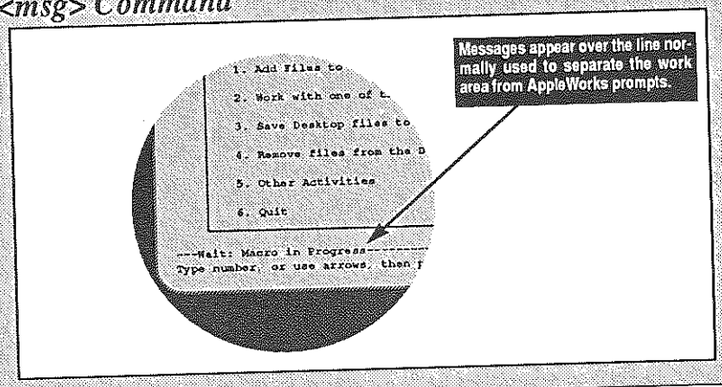
<bell>: The <bell> command beeps the computer's speaker either to draw attention or to notify the user of an error.

The syntax of the <bell> command is simple: <bell>. When UltraMacros encounters this token, it sounds the AppleWorks error buzzer and continues the operation of the macro.

The macro in *Figure 7.1B* uses <bell> to draw attention to the message on the screen.

You will use the <bell> command more often than you expect: UltraMacros mutes the standard AppleWorks error buzzer

Figure 7.2: Effect of <msg> Command



while running a macro. For example, you will use the <bell> command to notify users of mistakes or problems as they run a macro.

<msg>: The <msg> command lets you display messages at the bottom of the AppleWorks screen. (See *Figure 7.2* for an example of a message displayed from UltraMacros.) You can use this command to notify the user of the progress of a long macro, or to prompt the user when they must provide input to continue a macro.

The format of the <msg> command is <msg "Your message here">. Colons can replace the "<" and ">" signs. You are limited to one line of text in each message. The message appears in inverse video if you use single quotes (') and in normal video if you use double quotes (") .

Note in *Figure 7.1B* that you must use the <msg> command a second time to remove the message from the screen. Otherwise the message remains on the display until AppleWorks rewrites the bottom of the screen.

UltraMacros 3.x running under AppleWorks 3.x adds additional features to the <msg> command. For example, any text you

Figure 7.3: Control Codes that Work with <msg>

chr$	msg	description
Ø1	%A%	clear to end of the line
Ø2	%B%	clear current line
Ø3	%C%	clear page
Ø4	%D%	clear from cursor to end of page
1Ø	%J%	inverse text
11	%K%	normal text

Figure 7.4: Macros that Use <msg>

```
L:<all:msg %J% + " This prints inverse text ">!
M:<all:msg "Mousetext : " + &ABCDEFGHIJKLMNOPQRSTUVWXYZ&>!
```

enter between ampersands (&) will appear as the Mousetext equivalent of those text characters. You can also specify a control character by putting the character between two percent signs (%). For example, %B% specifies a Control-B.

Figure 7.3 lists the control code characters you can access with the UltraMacros 3.x <msg> command. *Figure 7.4* includes two macros that use these features. The availability of Mousetext and direct access to control codes adds useful power to the <msg> command in the latest versions of UltraMacros.

Chapter 15 describes additional message display capabilities available with UltraMacros-enhanced versions of AppleWorks 3.x.

Accepting Input from the User

The <msg> and <bell> commands are two ways you can communicate with the person using a macro. There are also commands that let you accept keystrokes entered by the user. For

Figure 7.5: Auto-Backup Macro

```
start
<ctrl-S>:<all : oa-S : bell :
  msg ' Insert your backup disk and press a key ':
  X=key : oa-S : bell :
  msg ' Replace your master disk and press a key ':
  X=key : msg ''>!
```

example, the macro in *Figure 7.1B* prompts the user to enter a file name and press the Return Key.

The <input> and <key> commands pause the operation of a macro and accept input from the user.

<input>: The <input> command pauses execution of a macro so the user can navigate around AppleWorks, use Open-Apple commands, and enter text at the keyboard. The <input> command ends and the macro continues operating when the user presses the Return Key.

Note that you must follow the <input> command with a <rtn> if AppleWorks also expects a Return to indicate the end of a user's input. For example, in *Figure 7.1B*, the user must enter a name for the new word processor file. At this point, both UltraMacros and AppleWorks are waiting for a press of the Return Key. The first press of the Return Key signals the end of the user's input in response to the UltraMacros <input> command. UltraMacros "traps" that Return Key and resumes operation of the macro. However, AppleWorks is also waiting for a press of the Return Key to signal that it should accept the new file name. Thus, the macro program must issue a <rtn>. It is the <rtn> sent from the macro that actually enters the new file name into AppleWorks.

<key>: The <key> command pauses a macro and tells AppleWorks to wait until the user presses any key. As you can see in

Figure 7.5, you usually precede the <key> command with a message that says "Press any key to continue". The keypress is "trapped" by UltraMacros the same way the <input> command traps the Return Key.

Note that in *Figure 7.5*, <key> always appears in an equation (in this case, the equation is <X=key>, but you can use any letter instead of the "X"). Versions 1.x and 2.x of UltraMacros let you use <key> alone to stop operation of a macro and await user input. However, starting with UltraMacros 3.0, <key> must appear in an equation, as in *Figure 7.5*.

Later chapters will describe more powerful applications of the <key> command.

If you use an UltraMacros-enhanced copy of AppleWorks 2.x, the AppleWorks cursor does not blink during <input> or <key> commands so you know you are in the middle of a macro. Under AppleWorks 3.x, each keypress causes UltraMacros to click the speaker when an <input> or <key> command is active.

Tutorial

Now you will construct a macro that combines the commands I just described. The macro will automatically store a copy of the data file on your desktop onto a backup data disk. You will type this macro into a word processor file and make this macro a part of your UltraMacros default macro set. Since you will add this macro to your default macro set, you will first use the Macro Compiler to create a word processor document that contains your current macro set.

Follow these steps:

1. Boot your copy of UltraMacros-enhanced AppleWorks and indicate you want to create a new word processor document "From scratch".

2. Go to the TimeOut Menu and use the Macro Compiler to list your current macro set into that word processor document. Save the word processor file on a disk and remove the file from the desktop to ensure it is not affected by your experiments.
3. Create another word processor file from scratch, and enter the macro that appears in *Figure 7.5*.
4. Use the Macro Compiler to compile the macro. This replaces the current macro with the single Solid-Apple-Control-S macro.
5. Run the macro. When you press Solid-Apple-Control-S, AppleWorks will save the current file twice, prompting you to swap disks after the first save operation. This macro is useful if you find yourself forgetting to make backups of important work.

Here are the steps required to add this macro to your default macro set:

6. Copy the entire macro, not including the word "start", to the AppleWorks clipboard.
7. In step #1, you saved a word processor file to disk with a listing of your default macro set: Load that file onto the desktop.
8. Use the Apple-9 command to move the cursor to the end of the macro listing. Press Apple-C to copy the Solid-Apple-Control-S macro from the clipboard into the listing of the default macro set. The word processor file now holds the listings for both your standard macros and the new Solid-Apple-Control-S macro.
9. Invoke the TimeOut Menu and use the Macro Compiler to compile the macro set. The combined set now is active as the current set.

10. Return to the TimeOut Menu and use the Macro Options Menu to make the current set of macros the new default set. The Solid-Apple-Control-S macro will then be available each time you start AppleWorks.
11. If you are using AppleWorks 2.x, use the Macro Options Menu to create a task file called "DEFAULT.MACROS". That will replace the old DEFAULT.MACROS file with one containing the Solid-Apple-Control-S macro.

The Solid-Apple-Control-S macro is not perfect. For example, it cannot react properly when the disk is full. In future chapters you will develop the skills necessary to write macros that can respond to unusual circumstances.

Conclusion

You now know how to use UltraMacros to program AppleWorks for automated operation and how to use the <bell>, <msg>, <key>, and <input> commands.

8: Introduction to Branching

This chapter describes how to create macros that can make decisions based on user input.

You now know two ways to prepare an UltraMacros macro: By capturing your keystrokes or by using the appropriate tokens to write a macro in the AppleWorks word processor. Until now, the macros you wrote were "linear"; they proceeded in a step-by-step fashion and did not react when the situation changed. For example, if you use the auto-backup macro illustrated in the previous chapter with a disk that is full, AppleWorks stops and issues a warning about the status of the disk. However, the macro does not recognize this condition and issues commands that are inappropriate for this unanticipated situation.

This chapter will introduce the concept of "intelligent" macros; macros that "branch", depending upon your response to a question. By the end of this chapter, you will know how to use the <getstr>, <key>, <if>, and <goto> commands to control a macro. Later chapters will build upon these techniques.

Two Sample Macros

Figure 8.1 contains two macros that automatically save a copy of an AppleWorks file on a backup disk. The macro in *Figure 8.1A* is the Solid-Apple-S macro we developed in the previous chapter. This macro instructs the user to insert a backup disk and then saves a copy of the current AppleWorks file onto that disk.

The macro in *Figure 8.1B* is similar, but adds "branching" capability to the Solid-Apple-S macro. The branching macro asks you a question, accepts your input, and makes a decision based on your response.

You should enter, compile, and run the macro in *Figure 8.1B*. *[Ed: See Chapter 4 for step-by-step directions on how to enter, compile, and run a macro.]*

Figure 8.1A: Auto-Backup Macro

```
start
<ctrl-S>:<all : oa-S : bell :
msg ' Insert your backup disk and press any key ':
key : oa-S : bell :
msg ' Replace your original disk and press any key ':
key : msg ''>!
```

Figure 8.1B: Enhanced Macro

```
start
<ctrl-s>:<all : oa-s :bell :
msg "File is saved. Do you want a backup also? (y/n)" :
$1 = getstr 1 : msg "" :
if $1 = "n" then stop : elseoff :
msg ' Insert your backup disk and press a key ':
key : oa-S : bell :
msg ' Replace your original disk and press a key ':
key : msg ''>!
```

The <getstr> Command

The first line of the Solid-Apple-Control-S macro in *Figure 8.1B* uses the <msg> (message) command I described in the previous chapter.

The second line, "$1 = getstr 1", contains two new UltraMacros tokens, <getstr> and $1. (<getstr> is an abbreviation for "Get String". In the lexicon of computer users, a "string" is any series of characters you can type at the keyboard. For example, "Hello" is a string composed of the characters "H", "e", "l", "l", and "o". "X" is a string composed of the character "X".)

The <getstr> command instructs UltraMacros to wait for an entry made by the user and to "capture" that text. The text is not sent to AppleWorks. (The last chapter introduced the <input> command. Unlike <getstr>, responses to the <input> command are sent directly to AppleWorks and are not used by UltraMacros.)

When you run this macro, the <getstr> command replaces the bottom line on the AppleWorks screen with a ">" symbol, indicating that UltraMacros is waiting for an entry. When the user presses the Return Key, the standard AppleWorks message reappears.

The <getstr> command can accept up to 60 characters of text. The number following the token tells UltraMacros the maximum number of characters it should accept. In this example, the user is supposed to type a single letter ("y" or "n"), so the correct format is "getstr 1".

How to Store User Responses

The statement "$1 = getstr 1" tells UltraMacros to store the user's response in a location called "$1" (described as "String One"). Whatever the user types in response to the <getstr> command is placed in the location named $1. You can then refer to location $1 to retrieve that information. For example, the command <print $1> sends the string stored in location $1 to AppleWorks.

UltraMacros has 10 storage locations for text information called $Ø, $1, $2, and so on, up to $9.

The next line in the macro, "if $1 = "n" then stop : elseoff" tells UltraMacros to check if the value stored in location $1 is the lower case letter "n". If $1 contains an "n", the instruction tells UltraMacros to stop this macro. If it is anything other than an "n", the macro continues uninterrupted.

This line contains an "<if> statement". In its simplest form, an <if> statement has four parts: the word "if", an expression, an outcome, and the word "elseoff". *[Note: UltraMacros-enhanced copies of AppleWorks 3.x allow either <endif> or*

<elseoff> to terminate an <if> statement.] The "outcome" occurs only if the expression is true. In this example, if the location labeled $1 holds the character "n", the macro stops. If the location contains any other letter (including a capital "N") the macro continues and begins the backup procedure.

The tokens <elseoff> and <endif> signify the end of an <if> expression. In this example, the role of the <elseoff> token is not immediately apparent because we are using the simplest form of the <if> expression. The function of <elseoff> and <endif> will become more apparent as we write more complex <if> statements in later chapters. For now, remember to follow every <if> statement with the token <elseoff> or, if you use AppleWorks 3.x, either <elseoff> or <endif>.

A More Sophisticated Approach

Just as there are many ways to complete a task in AppleWorks, there are an infinite number of ways to approach a problem with UltraMacros. *Figure 8.2* contains a set of three linked macros that demonstrate a more sophisticated approach to automating the process of saving an AppleWorks file on a backup disk. These macros examine the user's response to the <getstr> command. If the user enters an "N" or "n", the macro stops. If the user enters a "Y" or "y", the backup procedure continues. If the user makes any other entry, the macro displays the message "Press Y to back up your file; N to cancel", and lets the user enter a correct keystroke.

Figure 8.3 contains a flowchart that explains the logic of this macro.

As you can see from *Figure 8.3*, this macro has three "branches". The first macro, Solid-Apple-Control-S, saves the original file and displays the question about continuing with the backup process. Then macro Solid-Apple-Control-S calls the second macro, Both-Apple-1.

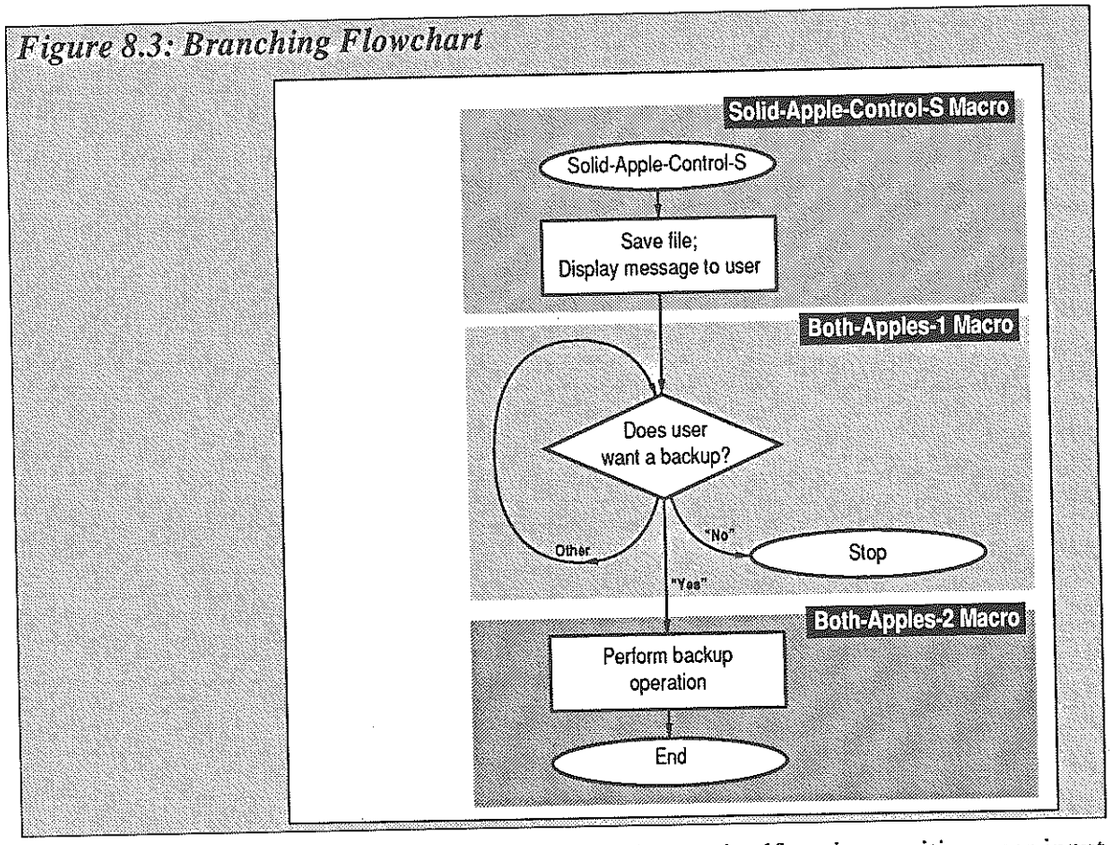
Figure 8.2: Macro Segmented for More Complex Branching

```
start
<ctrl-s>:<all : oa-s : bell :
  msg ' File is saved. Do you want a backup? (y/n) ' :
  goto ba-1>!

<ba-1>:<all : $1 = getstr 1 : msg '' :
  if $1 = "N" then stop : elseoff :
  if $1 = "n" then stop : elseoff :
  if $1 = "Y" then goto ba-2 : elseoff :
  if $1 = "y" then goto ba-2 : elseoff :
  bell : bell :
  msg ' Press Y to back up your file; N to cancel. ' :
  goto ba-1>!

<ba-2>:<all :
  msg ' Insert your backup disk and press any key ':
  X = key : oa-S : bell :
  msg ' Replace your original disk and press any key ':
  key : msg ''>!
```

There are two reasons you should segment these macros. First, the <goto> command must always refer to a macro. If you use a <goto> command, you will have to segment your macro. Second, there are essentially three tasks in this process. It is easier to conceptualize and write a program if you put each separate task in a different macro.

The line "$1 = getstr 1 : msg " " in macro Both-Apple-1 accepts a one-letter input from the user, stores that entry in location $1 and cancels the display of the earlier message. Then the macro tests the contents of location $1. If $1 contains an "N" or "n" the macro stops. If $1 contains a "Y" or "y", macro Both-Apple-1 turns over control to the macro Both-Apple-2. If $1 does not contain an "N", "n", "Y", or "y", the macro Both-Apple-1 continues. It sounds the bell twice, displays the message "Press Y to make a backup of your file; N

Figure 8.3: Branching Flowchart



or Escape to cancel" and starts itself again, awaiting user input in response to the <getstr> command.

The <goto> Command

Note the use of the <goto> command in the Both-Apple-1 macro in *Figure 8.2*. The combination of the <if> statement and the <goto> command gives UltraMacros the branching capability that provides much of the power in this programming language. You will use goto statements regularly when you write more powerful AppleWorks macros in later chapters.

The <key> Command

While the <getstr> command is easy to understand and is suitable for longer text entries, it is not the best way to accept the single keystroke response required in this macro.

Every key combination available on the Apple keyboard has a unique numeric value, and UltraMacros offers a <key> command, which waits for the user to enter a single keystroke and then captures the numeric value of that keypress. The table in Appendix A lists the numeric value of every key combination. UltraMacros then recognizes the user's keystroke and proceeds with the macro. As I mentioned in the last chapter, in AppleWorks 2.x the cursor does not flash while the <key> command awaits a user input.

As with <getstr>, UltraMacros does not automatically remember responses to the <key> command; you must define a location to store the numeric value of the keypress. For example, the statement "X = key" in the second line of macro Both-Apple-2 stores the keystroke in location "X". UltraMacros-enhanced copies of AppleWorks 2.x let you use either the syntax <key> or <n=key> to pause a macro. However, AppleWorks 3.x only recognizes the <n=key> syntax. You never have to use the value you store in location "n" with the <n=key> command, but you must use that structure for your statements.

Why the <key> Command Yields Numbers

You might wonder why the <key> command captures numbers, not the actual letters you press. That's because many of the keystroke combinations available on a computer keyboard cannot be represented visually on the screen or in print. For example, there is no visual image generated when you press the Escape Key or type a Control-A. The difficulty associated with representing these non-printable characters led the computer

industry to develop a standard numeric conversion system, called American Standard Code for Information Interchange (known by its acronym, ASCII). The ASCII system assigns a number to every key and key combination. Thus, the letter "A" generates a different value than a Control-A. Appendix A lists the numeric ASCII equivalent for every key combination available on the Apple keyboard. For example, the key combination Control-A has an ASCII value of 1. You will need this table when you write macros that use the <key> command.

There is another difference between UltraMacros-enhanced copies of AppleWorks 2.x and 3.x: Under AppleWorks 2.x, UltraMacros differentiates between entries of <oa-A> (upper-case "A") and <oa-a> (lowercase "a") in response to a <key> command. AppleWorks 3.x sees both entries as an upper-case letter "A". Note that this only occurs with responses that include Open-Apple keystrokes in combination with a letter of the alphabet.

Use <key> and <if> to Control Branching

Figure 8.4 depicts a series of macros that use the <key> command to capture the user's response and the <if> command to control the flow of the macros. These macros are more sophisticated than those depicted in *Figures 8.1* and *8.2*.

The <msg> command in the Solid-Apple-Control-S macro in *Figure 8.4* asks you to press a single key and then transfers control to the macro Both-Apple-1.

The <key> command in the statement "K = key" then pauses the macro and waits for the user to press a key. It stores the numeric value of that keypress in location K.

The macro Both-Apple-1 uses a series of <if> statements that take different actions depending on the key pressed in response to the <key> command. If you press the Escape Key (which

Figure 8.4: Macros Using the Key Command

```
start
<ctrl-s>:<all : oa-s : bell :
  msg "File is saved. Do you want to make a backup? (Y/N/Escape)" :
  goto ba-1>!

<ba-1>:<all : K = key : msg "" :
  if K = 27 then stop : elseoff :       { Was Escape pressed?       }
  if K = 78 then stop : elseoff :       { Was upper-case N pressed? }
  if K = 11Ø then stop : elseoff :      { Was lower-case N pressed? }
  if K = 89 then goto ba-2 : elseoff :  { Was upper-case Y pressed? }
  if K = 121 then goto ba-2 : elseoff : { Was lower-case Y pressed? }
  bell : bell :
  msg ' Press Y to make a backup of your file; N or Escape to cancel. ' :
  goto ba-1>!

<ba-2>:<all :
  msg ' Insert your backup disk and press any key ':
  key : oa-S : bell :
  msg ' Replace your original disk and press any key ':
  key : msg ''>!
```

has a numeric value of 27), the macro terminates and returns you to AppleWorks. If you press an upper- or lower-case "Y" or "N", the macro either cancels the save or transfers control to macro Both-Apple-2 that creates the backup of your file. If you press any other key, Both-Apple-1 sounds the bell twice, displays a message reminding you to enter an acceptable keystroke, and returns to the beginning of the same macro. The logic of this UltraMacros program corresponds to that of the macros in *Figures 8.2* and *8.3*.

A Comparison of the <getstr> and <key> Commands

As indicated earlier, the macros in *Figure 8.4* gain their power by using <key> instead of <getstr>. There are four important differences between the <getstr> and <key> commands:

1. The <getstr> command can capture a series of characters; the <key> command only captures a single character.
2. The user must press the Return Key when entering text in response to a <getstr> command. You do not have to press Return when responding to a <key> command.
3. The AppleWorks cursor flashes while awaiting a response to the <getstr> command. Under AppleWorks 2.x, the cursor does not flash during a <key> command. Under AppleWorks 3.x, the cursor flashes and the the speaker "clicks" when you enter keystrokes in response to <key> commands.
4. The <getstr> command accepts a series of text characters; the <key> command always generates a number. That number is the ASCII value of the key you enter. This forces the UltraMacros programmer to store the responses to <getstr> and <key> commands in different locations. UltraMacros stores text in locations $Ø through $9 (the dollar sign signifies a "string" location). UltraMacros can store up to ten separate strings of text.

   User input in response to the <key> command generates a number, and UltraMacros stores numbers in locations identified by any upper-case letter of the alphabet. UltraMacros can store up to 26 different numbers; one for each letter of the alphabet. You do not use a dollar sign to identify locations for numeric data. Thus, the entry in response to the <key> command in *Figure 8.4* is stored in location "K".

Conclusion

In this chapter, I described how to use the <getstr> and <key> commands to capture user input and how to use the <if> and <goto> statements to write macros that branch depending on that input. In the next chapter I will describe how to write macros that also accept input from the AppleWorks screen.

9: Macros that Read the Screen

This chapter describes how to create macros that branch based on information they read from the screen.

The previous chapter described how to write macros that "branch" depending on your response to messages displayed on the screen. This chapter describes commands that let you obtain information from the AppleWorks screen. By the end of this chapter, you will be able to capture information displayed on the AppleWorks screen and use that data in your macros.

UltraMacros offers several powerful commands that let you capture data from the AppleWorks display; this chapter describes three of them: The <screen> command reads the AppleWorks display and lets your macro branch depending on what is on the screen. The <cell> command captures the contents of any entry in a data base file or the contents of any cell in a spreadsheet. The <find> command tells UltraMacros to highlight a menu item or file you specify in AppleWorks listings.

The <screen> Command

Some AppleWorks commands work consistently throughout the AppleWorks modules. For example, when you type an Apple-Q, AppleWorks always presents the Desktop Index.

However, AppleWorks sometimes varies its response to your commands. For example, consider the Apple-R command in the data base module. When you issue an Apple-R, AppleWorks checks if a previous set of record selection rules are active. If record selection rules are in effect, AppleWorks asks if you want to "Select all records?". If no record selection rules are in effect, AppleWorks lets you specify the selection criteria.

This presents a problem when you write a macro. The macro must respond one way if record selection rules are already in effect, and respond a different way if rules are not in effect.

The <screen> command can read the AppleWorks prompts on the screen and lets you write macros that branch depending on the contents of the display. In this example, you can use the

<screen> command to check the bottom line of the screen and see whether or not it contains the prompt "Select all records?"

The <screen> command is similar in structure to the <getstr> command described in the previous chapter. If you want to let the user enter a five-character string, you use the command "$1 = getstr 5". Similarly, if you want to read five characters from the beginning of the third line on the screen, you use the command "$1 = screen 1,3,5".

The <screen> command requires three numbers to indicate where to start reading the screen and the number of characters to read from the screen. The AppleWorks screen consists of 24 lines of 80 columns each. The <screen> command requires that you specify the column you want to read from, then the line, then the number of characters you want to read. <screen 1,3,5> says "Go to the first character in the third line down from the top of the screen. Read in five characters starting from that position."

A Sample Macro Using <screen>

Figure 9.1 contains two macros that demonstrate how to use the <screen> command. You invoke these macros when you want to print either a data base report or a spreadsheet. The macros check to determine if you will print a report header at the top of the page. If you want to print a header, the macros automatically enter the correct date in response to the "Type report date or press Return" prompt. If you do not want to print a header, the macros skip over this step and start the printing process. Since you defined the first macro as an "asp" (AppleWorks spreadsheet) macro and the second macro as an "adb" (AppleWorks data base) macro, UltraMacros runs the appropriate one when you type Solid-Apple-P.

Figure 9.1: Auto-Print Macros

```
start
P:<asp:oa-P>A<rtn:
    $Ø=screen 1,24,4:
    if $Ø="Type" then oa-Y : date : rtn : elseoff :
    rtn>!
P:<adb:$Ø=screen 43,1,6 :
    if $Ø="CHANGE" then oa-P : rtn>1<rtn : elseoff :
    oa-P : rtn :
    $Ø=screen 1,24,4:
    if $Ø="Type" then oa-Y : date : rtn : elseoff :
    rtn>!
```

The Spreadsheet Macro

The first line in the spreadsheet macro (P:<asp: oa-P>A<rtn:) issues an Open-Apple-P command, responds "All" to AppleWorks' "Print? All Rows Columns Block" prompt, and then enters a Return to select your current printer.

The second line in the macro ($Ø=screen 1,24,4) tells UltraMacros to read the text in the first four columns at the bottom of the screen and store that text in location $Ø ("String Zero"). If you want to print report headers, the screen displays "Type report date or press Return:" on the bottom line and UltraMacros captures the string "Type". If you do not want to print headers, AppleWorks displays "How many copies?" at the bottom of the screen and $Ø will contain the string "How".

The third line in the macro (if $Ø="Type" then oa-Y : date : rtn : elseoff) tells UltraMacros to check to see if location $Ø contains the string "Type". If $Ø contains "Type", UltraMacros issues an Apple-Y command to yank out the previous entry. The <date> command followed by <rtn> then enters the current date. <elseoff> terminates the <if> command.

The last line in the spreadsheet macro enters a Return in response to the "How many copies?" prompt to accept the default value of one copy.

The Data Base Macro

The data base macro is more complex than the spreadsheet macro because the AppleWorks data base module uses the Apple-P command in two different ways. If you issue an Apple-P command in Review/Add/Change mode, AppleWorks takes you to the Report Menu screen. If you issue an Apple-P when you are defining a report, AppleWorks starts the report printing sequence.

The first line in this macro ($Ø = screen 43,1,6) captures part of the text from the top of the AppleWorks screen and stores that text in location $Ø. If you are in Review/Add/Change mode, location $Ø will now contain the string "CHANGE".

The second line in the macro (if $Ø="CHANGE" then oa-P:rtn>1<rtn:elseoff) checks to see if $Ø contains the string "CHANGE". If $Ø contains "CHANGE", the macro issues an Apple-P command to take you to the Report Menu, a Return to indicate you want to use an existing report format, the number "1" to highlight the first report format, and a Return to select that report format. If $Ø does not contain the string "CHANGE", UltraMacros skips to the "elseoff" and does not execute the Apple-P, Return, "1", or Return Key entries.

The next line (oa-P:rtn) tells UltraMacros to issue an Apple-P command and to issue a Return to select the highlighted printer from the Printer Menu.

The next three lines ($Ø=screen 1,24,4 : if $Ø="Type" then oa-Y:date:rtn:elseoff:rtn>!) are identical to the corresponding lines in the spreadsheet macro. They tell UltraMacros to enter the date if AppleWorks displays the string "Type" at the bottom of

the screen and then accept the default entry specifying the printing of only one copy.

These macros demonstrate how you can use the <screen> command to help UltraMacros determine the appropriate response for different situations within AppleWorks.

The <cell> Command

Although the <screen> command will read any text on the screen, it is difficult to use <screen> to capture data that is in an AppleWorks file. For example, sometimes AppleWorks displays a specific spreadsheet cell at one location on the screen; at other times that cell appears at another location.

UltraMacros offers the <cell> command, which captures the contents of the currently highlighted spreadsheet cell, the contents of the current entry in a data base record, or the contents of the current line in a word processor document. If you use the format "<cell>" in a macro, UltraMacros automatically stores the data in location $Ø. You can also specify any storage location from $Ø through $9 by using the format <$2 = cell>.

Figure 9.2 presents an example of a macro using <cell>. An explanation of that macro follows:

Imagine that you keep a record of your contacts with clients. Every time you call on a client, you enter the date and purpose of the contact. *Figure 9.3A* depicts an example of the data you might store in this AppleWorks data base file. The most current contact is at the top of the list. You also want to retain information about the three previous contacts.

The macro in *Figure 9.2* uses the <cell> command to capture the contents of each data base entry and move that entry into another location in the same data base record.

Figure 9.2: Sample <cell> Macro to Shift Entries

```
start
s:<adb :
  down : down : down : down : down :{ Move highlight to Prior Date2 category }
  down : down : down : down :
  $1 = cell : down : $2 = cell :     { Read Prior Date2 into $1; read Note 2 into $2 }
  down :                             { Move cursor to Prior Date3 category }
  oa-y : print $1 : rtn :            { Replace original data with date from Prior Date2 }
  oa-y : print $2 : rtn :            { Replace Note3 with data from Note2 }
  up : up : up : up : up : up :      { Move highlight to Prior Date1 category }
  $1 = cell : down : $2 = cell :     { Read Prior Date1 into $1; read Note 1 into $2 }
  down :                             { Move cursor to Prior Date2 category }
  oa-y : print $1 : rtn :            { Replace original data with date from Prior Date1 }
  oa-y : print $2 : rtn :            { Replace Note2 with data from Note1 category }
  up : up : up : up : up : up :      { Move highlight to Date/This Cntct category }
  $1 = cell : down : $2 = cell :     { Read Date/This Cntct into $1; Read Note into $2 }
  down :                             { Move cursor to Prior Date1 category }
  oa-y : print $1 : rtn :            { Replace original data with date from Date/This Cntct }
  oa-y : print $2 : rtn :            { Replace Note1 with data from Note category }
  up : up : up : up :                { Move cursor to Date/This Cntct category }
  oa-y : date2 : rtn :               { Delete old date and replace with current date }
  oa-y>!                             { Delete previous data in Note category }
```

How the Macro Works

The objective is to move the data from each date and note category into the next category, thus making room in the "This Cntct" and "Note" categories for information about your latest contact with the customer. The macro in *Figure 9.2* uses the <cell> command to capture the data in a category, then uses the <print> command to enter the data in the new location.

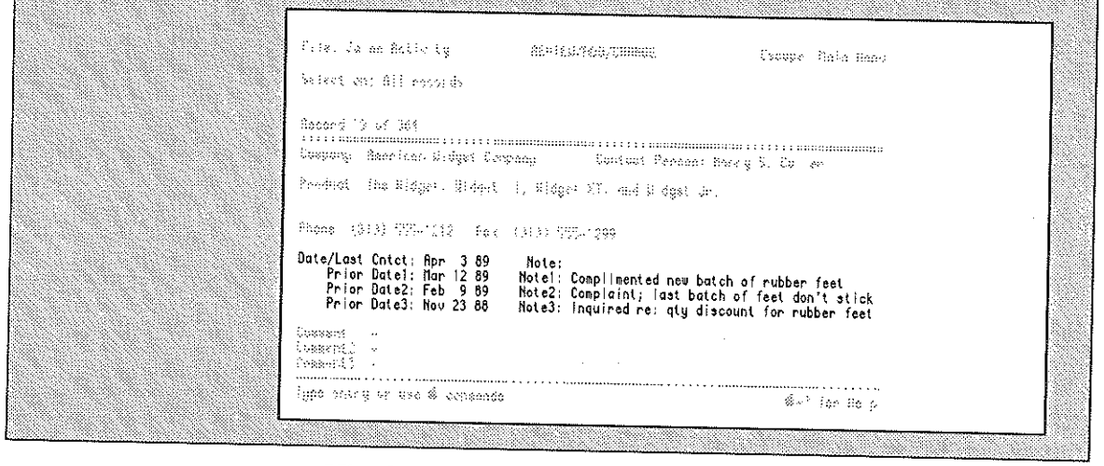
The first two lines in this macro consist of nine down arrow commands, which move the cursor to the tenth category, "Prior Date2".

The next line ($1 = cell : down : $2 = cell) uses the <cell> command to copy the entry in the "Prior Date2" category into memory location $1 and copies the entry in the "Note2" category into location $2. The fourth line (down:) commands

Figure 9.3A: Data Base File for Use with Sample Macro

```
File: Sales Activity          REVIEW/ADD/CHANGE          Escape: Main Menu
Selection: All records

Record 19 of 364
================================================================================
Company: American Widget Company        Contact Person: Henry S. Caller

Product: The Widget, Widget II, Widget XT, and Widget Jr.

Phone: (313) 555-1212   Fax: (313) 555-1299

Date/This Cntct: Mar 12 89     Note: Complimented last batch of rubber feet
    Prior Date1: Feb  9 89    Note1: Complaint; last batch of feet don't stick
    Prior Date2: Nov 23 88    Note2: Inquired re: qty discount for rubber feet
    Prior Date3: Sep  1 88    Note3: Requested 500 rubber feet for Widget XTs

Comment1: -
Comment2: -
Comment3: -
--------------------------------------------------------------------------------
Type entry or use ⌘ commands                                  ⌘-? for Help
```

Figure 9.3B: Data Base Record after Macro Execution



UltraMacros to move the cursor to the next category, "Prior Date3".

Line five (oa-y : print $1 : rtn) deletes the original entry in "Prior Date3" and prints the contents of location $1 into that category. Line six repeats that process; replacing the contents of Note3 with the text originally in Note2.

Figure 9.4: Spreadsheet Macro that Copies a Cell

```
C:<asp: cell : right : print $Ø>!  { Copy current cell to the right }
```

Line seven (up : up : up : up : up : up) moves the cursor up to the category "Prior Date1". Lines eight through eleven move the cursor to the appropriate categories, delete the original entries, and print the new entries in their place.

The macro repeats this process until it moves all entries to the following set of cells. The next to the last line (oa-y : date2 : rtn) removes the original entry from the "This Cntct" category and uses the <date2> token to enter the current date in the standard AppleWorks chronological format and moves the cursor to the "Note" category. The final line of the macro (oa-y>!) yanks out the original note, and terminates the macro. The user can then enter a note for the most recent contact with the client.

As you undoubtedly recognize, the <cell> command has numerous applications. For example, you can use <cell> to copy a single data base entry into many records, into different data bases, or even into different AppleWorks modules.

Enhancements to the <cell> Command

UltraMacros 3.0 and later offers two enhancements to the <cell> command in the spreadsheet module. First, earlier versions of UltraMacros do not always yield correct results when you invoke <cell> in the spreadsheet; UltraMacros versions 3.0 and later fix that problem. Second, UltraMacros 3.x captures the true value in a cell, not the rounded version or the format that appears on the screen. For example, <cell> captures $1,103.10 as 1103.1.

Figure 9.4 depicts a simple macro that copies the value in a cell to another cell. Under AppleWorks 2.x, this macro copies

the displayed value. Under AppleWorks 3.x, the macro copies the underlying true value.

To review: The <screen> command lets UltraMacros capture text that always appears at a stationary location on the AppleWorks display; the <cell> command captures data from an AppleWorks file even though that data does not always appear at the same location on the screen.

The <find> Command

Now imagine trying to write a macro that looks for a specific file on the Desktop Index, Disk Catalog, Report Menu, or Printer Menu. While the text you want to locate is on the screen, there is no easy way to use either the <screen> or <cell> commands to move the cursor to any specific item on the menu. UltraMacros offers the <find> command to help you select items from these AppleWorks menus.

<find> works differently from <screen> and <cell>. To use <find> you first store in location $Ø the string of text you want to find. Then you use <find> in your macro.

For example, imagine that the file "Sales Totals" is one of three files loaded onto the AppleWorks desktop, and you want to move to that file. The line "<oa-q : $Ø = "Sales Totals" : find : rtn>" calls up the Desktop Index, moves the highlight to the file "Sales Totals", then sends a Return keypress to AppleWorks, making that file active. If you want to be certain that a macro always uses the ImageWriter when printing a word processor file, you can incorporate the commands, <oa-p : rtn : $Ø = "ImageWriter" : find : rtn>, to print the file on the ImageWriter.

Another important use of <find> is to select a file from a catalog of files on an AppleWorks data disk. For example, if you use the file "Letter Template" with AppleWorks, you can use the <find> command to automate your use of that template. *Figure 9.5* lists a short macro that automatically loads the file

Figure 9.5: Using <find> to Load Templates

```
a:<all : oa-q : esc :                          { Go to the Main Menu }
  rtn : rtn :                                  { Add a File from the current disk }
  $Ø = "Letter Template" : find : rtn :        { Select and load template }
  oa-n>!                                       { Let user change the file name }
```

Figure 9.6: Macro that Launches Thesaurus in AppleWorks 3.0

```
T:<all: $Ø="Th" : oa-esc : find : rtn>!      { This macro will find TimeOut}
                                             { Thesaurus and run it.}
```

"Letter Template" and lets you rename the document before entering text.

UltraMacros 3.x Enhances <find>

UltraMacros 3.x adds four enhancements to the <find> command. First, <find> now searches all the TimeOut menus automatically. Second, <find> can now perform "partial finds". A partial find lets you search for a menu item that starts with the characters in $Ø. (Earlier versions of <find> require an exact match of the menu item with those characters.)

Macro <sa-T> in *Figure 9.6* demonstrates both these features. This macro sets the value of $Ø equal to "Th" and then searches the TimeOut Menu for the first item that begins with those characters. Thesaurus is the only TimeOut enhancement that starts with the characters "Th" in its name, so macro <sa-T> calls the TimeOut Menu and selects Thesaurus. It makes no difference if Thesaurus appears on your first, second, or third TimeOut Menu; <sa-T> will find and run that module.

Of course, you can use similar macros to select your favorite TimeOut modules with a single keystroke directly from AppleWorks.

UltraMacros 3.x's enhanced <find> command also uses variable Z to indicate whether or not a search is successful. If a search is successful, UltraMacros sets variable Z equal to one. If a search fails, UltraMacros sets Z equal to zero. You can then use the value of Z to branch to different segments of your macro.

Finally, unlike earlier versions of UltraMacros, the version 3.x <find> does not end a macro if the search fails.

The <ba-rtn> macro in *Figure 9.7* demonstrates these new features. This macro lets you enter any characters and tests to see if those characters appear at the beginning of the name of a TimeOut module. If <ba-rtn> finds those characters on the TimeOut Menu, it launches that application. Otherwise, <ba-rtn> displays the message "Not found" and terminates.

Figure 9.7: Macro that Launches Any TimeOut Module in AppleWorks 3.0

```
<ba-rtn>:<all:
    msg ' What module do you want? ' :
    $Ø = getstr 15 :                    { Get name of file to search for }
    oa-esc :                            { Call TimeOut Menu }
    find :                              { Search through all TimeOut menus }
    if Z=Ø                              { If Z=Ø...}
      then esc : stop : elseoff         { Stop }
    rtn>!
```

<find> from the Keyboard

UltraMacros also lets you use the <find> command from the keyboard. For example, follow these steps to use keystroke entries to find a TimeOut application:

1. Enter <oa-esc> to get the TimeOut Menu on the screen.
2. Press <oa-Ø>. This lets you enter text directly into variable $Ø. The ">" prompt will appear at the bottom of the screen.

Figure 9.8: Macro that Uses <date> and <time>

```
Z:<all: msg ' The date is ' + date + ' and the time is  ' + time>!
```

3. Enter the search string and press the Return Key. Under AppleWorks 2.x, you must enter the complete search string. Under AppleWorks 3.x, you only have to enter enough characters to make the search string unique; you do not have to type the complete TimeOut module name.

4. Press <sa-rtn> to invoke the <find> command and begin the search. UltraMacros 3.x looks through all the TimeOut menus (most users have only one menu, but if you have more than one TimeOut Menu, <find> "tabs" through those menus) and highlights the correct application. The command terminates if it does not find a match. If the search string is not unique, <sa-rtn> highlights the first module that matches the text in your string.

One important reminder about these UltraMacros 3.x enhancements: While UltraMacros 3.x is compatible with all versions of AppleWorks since version 2.0, these enhancements to the <find> command only work when you install UltraMacros on AppleWorks 3.0 or later. You do not get these features when you install UltraMacros on earlier versions of AppleWorks.

The <date> and <time> Commands

UltraMacros offers <date> and <time> commands that let you display the current date and time on the screen or enter that information into an AppleWorks document, data base, or spreadsheet. The <sa-Z> macro in *Figure 9.8* demonstrates a simple application of these commands.

Versions 1.x and 2.x of UltraMacros reads the date you entered when you last booted AppleWorks. That is a problem for users

who do not turn off their computers or who work past midnight; in those instances, the date is not correct.

UltraMacros 3.x enhances the <date> command so it reads the date from your IIGS or ProDOS clock and returns the current date, even if you started AppleWorks days earlier. This avoids many problems associated with leaving the computer on past midnight. If UltraMacros does not find a clock, the <date> command follows the current practice of returning the date you last started AppleWorks and the <time> command displays 00:00.

Conclusion

You now know three ways to use information presented on the AppleWorks screen. You can use the <screen> command in combination with the <if> command to control the flow of a macro. You can use the <cell> command to extract data from AppleWorks data files. You can use the <find> command to navigate AppleWorks menus efficiently. You also know how to use the <date> and <time> commands to capture and display those values. Next, we will examine other ways to manipulate and use variables with UltraMacros.

10: Introduction to Variables

This chapter describes the concept of "variables" : How to define them, and how to create variables that utilize user input and customize documents.

The previous two chapters introduced numerous UltraMacros commands that let you control AppleWorks. This chapter moves away from discussions of specific tokens and commands and looks at an important theoretical construct; the topic of variables.

Variables

You already know that UltraMacros can store numbers and text in identifiable memory locations. We call each location a "variable" because it stores data that can vary. For example, memory location A is "variable A". Variables A through Z can each store any positive integer between zero and 65355.

UltraMacros can store up to 26 different numbers in variables A through Z, and ten strings of text in variables $Ø (pronounced "String Zero") through $9. Each "string variable" can store up to 80 characters.

Defining Variables

You "define" a variable by storing something in that memory location. For example, if you issue the UltraMacros command <$1= "Letter Template"> you define variable String One by saying it will contain the text "Letter Template".

You can always use UltraMacros' <msg> (message) command to view the contents of a variable. For example, the command <msg $1> displays the contents of variable $1.

Literal Definition of Variables

There are different ways to define variables. The most direct way to define a variable is to tell UltraMacros to store a specific value or text in the variable. *Figure 10.1* includes five macros that store specific values or text in variables.

Each macro in *Figure 10.1* defines a numeric or string variable and uses the message command to display the contents of that variable at the bottom of the AppleWorks screen. You should try to enter, compile, and run each macro.

Figure 10.1: Macros that Store Literal Values in Variables

```
1:<all: A = 1 : msg A>!
2:<all: B = $1Ø : msg B>!
3:<all: $Ø = "Text" : msg $Ø>!
4:<all: $3 = "You can use 'single quotes'" : msg $3>!
5:<all: $4 = 'Or "double quotes" in a message' : msg $4>!
```

Macro <sa-1> stores a value of 1 in variable A and displays the variable.

Macro <sa-2> is more complex. The dollar sign means two different things in UltraMacros. A dollar sign before a single-digit number usually signifies the name of a string variable. UltraMacros also uses dollar signs to indicate that a number is a hexadecimal number. You must examine the context to determine the meaning of the dollar sign symbol.

The hexadecimal number "$10" is equivalent to the decimal number 16. When you run this macro, UltraMacros will display the number 16 at the bottom of the screen.

Macro <sa-3> stores the string "Text" into the string variable $Ø and displays the contents of this variable at the bottom of the screen.

Note that you must use either single or double quotes before and after the text you want to store in the string variable. Everything between the quotation marks becomes part of the variable; e.g., <$5 = ' Hi there! '> defines variable $5 to contain two spaces, the letters "Hi there", the exclamation mark, and two more spaces. You can use either single or double quotation marks to define a string variable. If you use a single

quotation mark to start the definition, you must use a single quotation mark to end the definition.

Finally, note that a colon must follow the definition of every string variable.

Macros <sa-4> and <sa-5> show that you can place either single or double quotes inside a variable.

Commands that Define Numeric Variables

So far, we have defined variables by storing something directly into a memory location. A number of different UltraMacros commands also define variables. For example, *Figure 9.1* in Chapter 9 includes the UltraMacros statement: <$Ø=screen 1,24,4>. That statement defines variable $Ø so it contains the first four characters at the beginning of row 24 on the AppleWorks screen.

Figure 10.2 contains macros that use the UltraMacros <len>, <val>, <key>, and <posn> tokens to define numeric variables. You should try to enter, compile, and run these sample macros.

Figure 10.2: Tokens that Define Numeric Variables

```
A:<all: $1 ="Length of String" : Z = len $1 : msg Z >!
B:<all: $1 = "1Ø" : X = val $1 : msg X >!
C:<all: Z = key : msg Z >!
D:<all: posn X,Y : msg X : Z=key : msg Y>!
```

Macro <sa-A>: The <len> command tells UltraMacros to determine the number of characters in a text string. So the command <Z = len $1> says, "Store the number of characters in $1 in the variable named Z".

Macro <sa-A> starts by defining variable $1 so it contains the text "Length of String". Then the macro uses the <len> command to store the number of characters in "Length of String" in variable Z. (There are 16 characters in "Length of String"

because spaces count as characters in a text string.) Then the macro displays the value of variable Z at the bottom of the AppleWorks screen.

Macro <sa-B>: The difference between numeric and string variables is important in a programming language. For example, if you store the characters "255" in a string variable, you cannot do mathematics on the number; UltraMacros treats these characters as text, not as numbers. The <val> command tells UltraMacros to check if a variable contains a number in text format and to convert that text string into a numeric variable. In this case, <X = val $1> says "Convert the text in variable $1 into a value and store that number in variable X." If the variable starts with a number, <val> converts that number. If the variable does not start with a number, <val> stores a zero in variable X. See *Figure 10.3* for examples of what <val> stores under various conditions.

Figure 10.3: Converting Strings to Numeric Values

Contents of $1	Result of <val $1>
$1 = "Ten"	val $1 = Ø
$1 = "1Ø"	val $1 = 1Ø
$1 = " 1Ø"	val $1 = Ø
$1 = "15Hello"	val $1 = 15
$1 = "15 Hello"	val $1 = 15
$1 = "X1ØØ"	val $1 = Ø
$1 = "#3"	val $1 = Ø
$1 = "11 + 3"	val $1 = 11

Macro <sa-B> starts by defining variable $1 so it contains the numeric characters "10". Then the <val> command defines variable X to contain the number 10. The message command prints the number 10 at the bottom of the AppleWorks screen.

Macro <sa-C>: The <key> command generates the numeric ASCII value of any key you press (see Chapter 8). For example, pressing the Escape Key yields the ASCII value of 27.

Macro <sa-C> captures that keypress and stores the ASCII value in the numeric variable Z. The macro then displays the ASCII value of the keypress at the bottom of the AppleWorks screen. (See Appendix A for the complete list of ASCII key equivalents.)

Macro <sa-D>: The <posn> (position) token identifies the coordinates of the cursor position. <posn> generates two values you store in separate numeric variables; e.g., the statement <posn X,Y> stores the coordinates of the current cursor position in variables X and Y.

In the word processor, the <posn> token generates the column and line number of the current cursor position. In the data base, <posn> generates the category and record number of the cursor location. (Thus, you can use <posn> to insert record numbers automatically in a data base file.) In the spreadsheet, <posn> generates the column and row number of the current cursor position.

Macro <sa-D> uses the <posn> token to store the cursor coordinates in numeric variables X and Y. The macro then displays the value of variable X. When you press any key the macro displays the value of variable Y.

Manipulating Numeric Variables

UltraMacros lets you add, subtract, multiply, and divide numeric variables. For example, the expression <A = A * A> multiplies the numeric value of variable A by itself (i.e., squares the number) and replaces the number originally stored in variable A with the square of that number. Similarly, <Z = Z - 1> subtracts one from the current value stored in Z and replaces the original value in Z with the new value.

I will describe how to use these numeric manipulations to control program flow in later chapters.

Defining Numeric Variables with Expressions

You can also write numeric expressions that define new variables. *Figure 10.4* includes examples of expressions that define new variables.

Figure 10.4: Expressions that Define New Variables

```
E:<all: A = 1 : B = A + 1 : msg B>!
F:<all: A = A + 1 : msg A>!
G:<all: B = 8 : C = 5 : A = B + 10 - C : msg A>!
H:<all: $1 = "15" : Z = 10 + val $1 : msg Z>!
```

Macro <sa-E>: This macro uses the expression <A = 1> to define a new variable called "A", and then stores the value "1" in that variable. Then <B = A + 1> defines variable "B", and specifies that B will contain the value of A + 1. Finally, the macro displays the value of variable B (the number 2) at the bottom of the screen.

Macro <sa-F>: Macro <sa-F> adds one to the value stored in variable A and displays the value of A in the message area on the AppleWorks screen. The first time you run this macro, A is "empty" (i.e., it contains a zero), so the formula adds one to zero and yields a one. Each time you re-run the macro, the formula increments the value of A by one. A typical use of this formula is as a "counter" to control the flow of a macro.

Macro <sa-G>: This macro demonstrates UltraMacros' ability to define and manipulate numeric variables. Macro <sa-G> first defines variable B to contain the value of 8 and variable C to contain the value of 5. Then the macro adds ten to the value of B and subtracts the value of C from that sum. It stores the result in variable A and displays the value of A at the bottom of the AppleWorks screen.

Macro <sa-H>: Macro <sa-H> uses the <val> command described above. The macro stores the string "15" in variable

$1, then adds together the number 10 and the value of $1. That is, <sa-H> stores the value of 10 + 15 in numeric variable Z and displays the value of Z on the screen.

Rules for Numeric Operations in UltraMacros

Here are some rules to remember about UltraMacros' ability to handle numbers:

1. UltraMacros only handles positive integers between zero and 65355.
2. Since the program can only handle integers, UltraMacros discards any remainder when you divide two numbers.
3. UltraMacros performs all operations in a left-to-right order. Thus, <A = 2 + 3 * 10> stores the number 50 in variable A. In addition, UltraMacros does not let you use parentheses to control the order of the arithmetic operations. However, you can perform separate operations, store the values of those operations in different numeric variables, and then operate on those variables in a separate formula. For example, UltraMacros will not accept <A = 2 + (3 * 10)>, but will accept <B = 3 * 10 : A = 2 + B>.

Commands that Define String Variables

Figure 10.5 illustrates eight different commands that define string variables.

Macros <sa-I>, <sa-J>, and <sa-K>: Macros <sa-I>, <sa-J>, and <sa-K> illustrate the use of the <right>, <left>, and <mid> tokens. These commands let you abstract portions of a larger string and store the results in a new variable. For example, <$2 = right $1,4> examines variable $1 and stores a copy of the last four characters into $2.

Figure 10.5: Tokens that Define String Variables

```
I:<all: $1 = "Beagle Bros" : $2 = right $1,4 : msg $2 >!
J:<all: $3 = "Beagle Bros" : $4 = left $1,6 : msg $4 >!
K:<all: $5 = "Beagle Bros" : $6 = mid $5,2,5 : msg $6 >!
L:<all: Z = 99 : $1 = str$ Z : msg $1 >!
M:<all: $3 = chr$ 65 : msg $3 >!
N:<all: $2 = getstr 5 : msg $2 >!
O:<all: $1 = date : msg $1 >!
P:<all: $2 = time24 : msg $2 >!
```

Macro <sa-I> stores the text "Beagle Bros" in variable $1. Then the macro stores the four right-hand-most letters ("Bros") in variable $2. Finally, the macro displays the contents of $2 at the bottom of the AppleWorks screen.

Similarly, macro <sa-J> stores the text "Beagle Bros" in variable $3 and creates variable $4 with the six left-hand-most characters from variable $3 ("Beagle"). Then the macro displays the contents of variable $4.

Macro <sa-K> demonstrates an application of the <mid> command, a function which works only with AppleWorks 3.x. <mid> lets you abstract any part of a string, and is similar to the MID$ function in Applesoft BASIC.

The syntax for <mid> is

```
<$1 = mid $2,X,Y>
```

where the "$1" is the string variable that stores the result, "$2" is the string variable that contains the original text, "X" is the number of characters into the source variable where you want to start copying, and "Y" is the number of characters you want to copy from the variable $2.

For example, <$Ø = mid $1,5,10> extracts ten characters starting with the fifth character in variable $1. It stores those characters in variable $Ø.

The <sa-K> macro stores the text "Beagle Bros" in variable $5, then creates variable $6 and stores the first five characters starting with the second character ("eagle") in that variable. Then the macro displays the contents of variable $6.

Macro <sa-L>: Much as the <val> command converts strings into numeric values, the <str$> command converts values into strings. There are many uses for this feature; for example, you can use it to combine values and text into longer text fields.

The <sa-L> macro stores the number 99 in numeric variable Z, then converts the numeric value into the string "99" and prints the string at the bottom of the AppleWorks screen. While the 99 that appears on the screen looks like a number, it really consists of two text characters; you cannot do mathematical operations on that text.

Macro <sa-M>: <sa-M> uses the <chr$> token; the obverse of the <key> command. <key> generates the numeric ASCII value of any keystroke; <chr$> generates the keystroke that corresponds to any numeric value. For example, <$8 = chr$ 177> stores an Open-Apple-1 "character" in variable $8.

Macro <sa-M> stores the letter "A" in variable $3, then prints variable $3 at the bottom of the AppleWorks screen. You will need the ASCII conversion chart in Appendix A to use the <chr$> token.

Macro <sa-N>: Macro <sa-N> uses the <getstr> token originally described in Chapter 8. <getstr> lets the user enter a specified number of characters that UltraMacros stores in a string variable. For example, <$4 = getstr 7> lets the user enter up to seven characters and stores those characters in variable $4.

Macro <sa-N> lets the user enter five characters, stores those characters in variable $2, and displays those characters at the bottom of the AppleWorks screen. You usually use the <msg>

command to prompt the user to enter text that you capture with <getstr>.

Macro <sa-O>: UltraMacros supports two date functions: <date> gives the date in the format January 1, 1990, and <date2> uses the format 01/01/90. Macro <sa-O> captures the current date in variable $1 and then displays the date at the bottom of the screen. One application of the <date> token is in macros that automatically enter the current date in new data base records.

Macro <sa-P>: If you have a ProDOS-compatible clock or an Apple IIGS, UltraMacros offers two time functions: <time> displays the time in the format 6:35 pm; <time24> displays the time in the format 18:35. Macro <sa-P> enters the time in 24-hour format into variable $2 then displays the time at the bottom of the screen.

Note that while <date2>, <time>, and <time24> appear to generate numbers, these are really strings of text you must capture in string variables.

Manipulating String Variables

You know that UltraMacros can add, subtract, multiply, and divide numeric variables. UltraMacros can also manipulate string variables. I described some of these features earlier when I discussed using the <right>, <left>, and <mid> tokens to abstract strings of text from a longer string. In addition, I discussed how to use the <key>, <screen>, and <getstr> commands to capture text from the keyboard or screen. UltraMacros also lets you concatenate (or append) two strings using the "+" designator. *Figure 10.6* depicts two examples of concatenation.

Macro <sa-P>: This macro creates the variable $1 containing the string "Hello " (with a space after the "o"). The statement <$1 = $1 + "There"> appends the string "There" to the end of

Figure 10.6: Concatenating Strings with UltraMacros

```
P:<all: $1 = "Hello " : $1 = $1 + "There" : msg $1>!
Q:<all: A = 5 : $1 = "Number " + str$ A +
        " is the winner!" : msg $1>!
```

$1. The message command displays the entire contents of $1 ("Hello There") at the bottom of the AppleWorks screen.

Macro <sa-Q>: Macro <sa-Q> creates the numeric variable A that contains a value of five. It then creates the string variable $1 that contains the text "Number ", plus the string value of A (the character "5"), plus the text " is the winner!". The message command prints all of $1: "Number 5 is the winner!". Note that UltraMacros truncates any string variables more than 80 characters long.

Conclusion

In this chapter I described how to create numeric and string variables. The role of variables will become more evident in the next chapter, when I will discuss program flow and subroutines.

11: Executing Repetitive Tasks

This chapter describes how to use UltraMacros' capability to "loop" so you can repeat a series of macro commands.

AppleWorks users write macros to save time performing repetitive tasks. While the macros presented so far save time by reducing several keystrokes into one, none can perform iterative operations in a controlled manner. If you want a task done five times, you must press the Solid-Apple key combination five times.

This chapter will describe how to write macros that "loop"; macros that repeat themselves. You will also learn how to use the <begin>, <rpt> (repeat), and <onerr> (on error) commands to control loops.

Why Use Loops?

There are many situations where the ability to "loop" can simplify or speed up an operation. For example, you can use looping macros to move a specified number of lines from the word processor into a spreadsheet, or to enter a sequential serial number into every record in a data base. Both these examples require that you execute a repetitive operation a specified number of times.

<begin> and <rpt>

One way to write a macro that loops is to use the <goto> command. Consider the following macro that waits for your input and rings the AppleWorks error buzzer unless you press the Space Bar:

```
G:<all: msg ' Press Space to stop ':
   A=key:
   if A=32 then stop : elseoff : { ASCII 32=Space Bar }
   bell : goto sa-g>!
```

This macro uses the <goto> command to "call itself". You can achieve the same effect with the <rpt> (for "repeat") command as follows:

```
G:<all: msg ' Press Space to stop ':
   A=key:
   if A=32 then stop : elseoff :
   bell : rpt>!
```

The <rpt> command tells UltraMacros to jump back to the beginning of the macro. If you do not want to repeat the entire macro, you can use the <begin> command to specify a new starting point, as follows:

```
G:<all: msg 'Press the Return Key':
   A=key:
   begin:
   msg 'This message repeats. Press Space to stop':
   A=key:
   if A=32 then stop : elseoff :
   bell : rpt>!
```

This macro displays the message "Press the Return Key" and waits for any keystroke. Then the macro displays the message "This message repeats. Press Space Bar to stop." The message stays on the screen until you follow its instructions.

While it is easy to write macros that loop, the problem is getting the macro to stop looping. There are three common ways to tell UltraMacros to stop looping: Using a "flag", a "counter", or checking for "errors".

Controlling Loops with "Flags"

The Solid-Apple-G macros above use an <if> statement to exit the loop; each macro continues to loop until you press the Space Bar. The <if> command can check for any entry or variable you specify.

When the introduction of a special value causes the end of a loop, that value is called a "flag". Just as race cars exit the track when you wave a checkered flag, these macros exit their

loops when variable A contains the ASCII value 32 (a press of the Space Bar).

"Counters"

A second common way to control a loop is through the use of a "counter". You use a "counter" to perform a series of operations a specified number of times. For example, you can use a counter to insure that a macro inserts the current date in only five records or to copy a spreadsheet row into three separate spreadsheets.

In the race car metaphor, imagine cars that leave the track after a certain number of laps and the driver in this race must count each lap.

To write a macro that "counts each lap", you must set up a "counter". A counter is a numeric variable that you either increment or decrement each time the macro loops. Then you use an <if> statement to check for the current value of the counter.

For example, here is a macro that uses a counter to print a line of 50 asterisks across the screen:

```
H:<awp: A=Ø:
   begin
   >*<    { print an asterisk }
   A=A+1: { increment the value of A }
   if A<50 then rpt: elseoff>!
```

Each time the macro loops, it prints an asterisk and adds "1" to the value in variable A. The <if> statement allows the macro to continue until the value of A reaches 50; then it terminates the macro. The <begin> command specifies the beginning of the loop; without <begin>, the macro would set A to zero with each pass, causing the loop to continue indefinitely.

You can use counters to control loops when you know the number of times you want to execute the loop. Alternatively,

you can ask the user how many times to repeat the macro, as in the following example:

```
H:<awp: A=Ø:
   B=Ø:
   msg "How many asterisks do you want to print?":
   $1=getstr 3:
   B=val $1:
   begin
   >*<    { print an asterisk }
   A=A+1: { increment the value of A }
   if A<B then rpt: elseoff>!
```

In this example, the macro instructs the user to enter a number. UltraMacros stores the number as a "string" in variable $1, then converts the string to a value it stores in numeric variable B. Later, the <if> statement tests if variable A (the counter) is less than variable B.

Using "Errors" to Exit Loops

You can only use a counter to exit a loop if you know how many times you want to repeat a process. Unfortunately, the conditions in AppleWorks are so variable that you often do not know the exact number of times to repeat a loop.

For example, imagine you want to write a macro that loads the last file on a disk onto the desktop. You want this macro to work no matter how many files are on the disk. Yet there is no way to know how many documents are on each disk, so you cannot use a counter to tell AppleWorks when the highlight is at the bottom of the list of files.

When you load files onto the desktop, you get to the end of the list of files by pressing the Down-Arrow Key. If you press the Down-Arrow Key again, AppleWorks beeps. This is technically called an "error" because you pressed the Down-Arrow Key when you were already at the end of the list. AppleWorks

sounds a warning to tell you there is no place for the Down-Arrow Key to move the cursor.

Earlier in this book I mentioned that UltraMacros disables the AppleWorks error bell. However, UltraMacros is still aware that the "error" occurred, even though the error bell does not sound. While UltraMacros usually ignores the warning beep, you can instruct a macro to take any specified action when a beep occurs.

The <onerr> Command

UltraMacros offers an <onerr> ("On the occurrence of an error") command that "listens" for the silent error warning and terminates a looping operation. <onerr> offers three options:

<onerr stop>: <onerr stop> tells UltraMacros to stop the current macro when it detects an AppleWorks warning beep and returns control to AppleWorks or to the macro that called the current macro. More information about <onerr stop> appears in Chapter 12.

<onerr goto>: <onerr goto> tells UltraMacros to terminate the current macro and start another macro. For example, <onerr goto sa-X> says "When you detect the warning beep, terminate the current macro and start macro Solid-Apple-X."

<onerr off>: <onerr off> tells UltraMacros to ignore AppleWorks warning bells. This is the default setting for UltraMacros.

You can put <onerr> commands anywhere in a macro, but you will typically place them at the beginning of the macro to turn error checking on or off. An <onerr> command remains in effect until the entire macro terminates or until the macro encounters another <onerr> command. For example, the command <onerr stop> at the beginning of a macro instructs UltraMacros to stop the macro when an error occurs at any point in

its execution. The following macro uses <onerr stop> to simulate the standard Open-Apple-9 command in AppleWorks:

```
9:<all : onerr stop : down : rpt>!
```

This macro enters a Down-Arrow until AppleWorks beeps. Then the macro stops. You can use this macro at any time in AppleWorks; it adds the equivalent of an Open-Apple-9 command to any menu or list of files.

Alternatively, you use the <onerr goto> command to check for "errors" and turn over control to another macro. For example, here is a macro that presses the Down-Arrow Key until AppleWorks beeps and then displays the message "You reached the end of the list":

```
9:<all : onerr goto sa-w : down : rpt>!
w:<all : msg ' You reached the end of the list '>!
```

<onerr off>

As indicated earlier, UltraMacros normally ignores AppleWorks' error beep unless you use an <onerr> command. However, there are times you want part of a macro to respond to the beep and other parts of the macro to ignore the beep. You make your macro responsive to the warning beep with an <onerr stop> or <onerr goto> command. You can tell UltraMacros to return to its normal condition (i.e., ignore the warning beep) by issuing an <onerr off> command anywhere in the macro.

How to Use "Errors" Constructively

While we normally associate errors with mistakes that we should avoid, the errors intercepted by the <onerr> command are *useful* in your macros. These errors give you control over loops that are difficult to control with flags or counters.

When you write macros, you should equate "errors" with soundings of the AppleWorks bell, not with mistakes. You should anticipate when AppleWorks will sound the error bell, and the implications of ignoring it, stopping the macro, or branching when the bell sounds.

12: Introduction to Subroutines

You can simplify complex procedures by using "subroutines" — macros that serve as building blocks in larger macros. This chapter describes how to create and use subroutines.

The previous chapter discussed some aspects of program flow. You learned how to write macros that "loop" to perform a single task repeatedly, and how to control those loops with variables. You learned that instead of writing a macro that says "Print, print, print, and print again", you could say, "Print four times". Loops save space and are a flexible tool with many applications.

What Is a Subroutine?

While loops make it easy to repeat the same steps numerous times, they do not serve all your needs for repetition. For example, imagine a macro that includes a short procedure you re-use often. You do not want this procedure to repeat itself, but want to re-use the series of steps from time to time throughout the macro. That is the function of a "subroutine"; a macro containing steps you can "call" repeatedly from another macro.

Subroutine macros look like any other macro, but subroutines do not stand by themselves; they provide a "service" to other macros.

A Macro within a Macro

Just as you use macros to simplify AppleWorks, macros can use other macros to simplify their work. While a keyboard macro reduces a complex series of tasks to a single keystroke, a subroutine reduces a complex series of program steps into a single token.

For example, consider the following two macros:

```
A:<awp>-subroutine-!

B:<awp><sa-a>MAIN PROGRAM<sa-a>!
```

If you compile these two macros and enter a Solid-Apple-A, you will get the message, "-subroutine-". Type a Solid-Apple-B and you will get the message "-subroutine-MAIN PROGRAM-

subroutine-". When UltraMacros encounters a token that represents another macro, it transfers control to the second macro (the "subroutine macro") and when that macro is finished, returns to the original macro to continue where it left off. In this example, macro <sa-a> is a subroutine for macro <sa-b>.

In most programming languages, a subroutine is a part of the main program. However, UltraMacros' programming language uses "subroutine macros"; completely separate macros that you call from another macro. Usually you call a subroutine macro from the main macro, but any macro can call another, so you can have subroutines call other subroutines.

A Simple Subroutine

Imagine you want to write a macro that combines the contents of two AppleWorks word processor files into a single file. The task involves (a) loading both files onto the desktop, (b) creating a file named "Joined" to hold the concatenated version of the two files, (c) copying the contents of the first file into "Joined", and (d) copying the contents of the second file to "Joined".

Figure 12.1A contains a macro that performs this operation without using subroutines. The macro repeats several tasks: Using Apple-Q to select a file, loading the files onto the desktop, and copying the text into the new file. You have to re-type those lines, so the program is unnecessarily complex.

The macros in *Figure 12.1B* perform the same task and demonstrate how subroutines improve the structure of a macro.

Macro <sa-c> in *Figure 12.1B* uses two other macros, <sa-a> and <sa-x>. Both <sa-a> and <sa-x> use a third macro, <sa-f>. The <sa-a> macro, combined with <sa-f>, takes any file name stored in $Ø and puts that file on the AppleWorks desktop.

<sa-x> uses the <sa-f> macro to locate the correct file and moves the contents of that file into the file named "BOTH".

The macros in *Figure 12.1B* consolidate repeated tasks into subroutines. Once you know the function of the subroutine, you can easily understand the function of the main macro.

Figure 12.1A: Macro that Concatenates Two Files

```
c:<all :
     oa-Q esc : rtn : rtn:                    { Load first file onto desktop }
     $Ø = "First" : find : rtn:
     oa-Q esc : rtn : rtn :
     $Ø = "Second" : find : rtn:              { Load second file onto desktop }
     oa-Q esc rtn>3<rtn rtn>Both<rtn:         { Create file to hold both }
     oa-Q : $Ø = "First" : find : rtn:        { Copy text from first file }
     oa-1 oa-C>T<oa-9 rtn :
     oa-Q $Ø = "Joined" : find : rtn :        { Copy text into new file }
     oa-1 oa-C>F<
     oa-Q $Ø = "Second" : find : rtn:         { Copy text from second file }
     oa-1 oa-C>T<oa-9 : rtn :
     oa-Q $Ø = "Joined" : find : rtn :        { Copy text into new file }
     oa-9 oa-C>F<oa-1>!                       { Move to beginning of file and end }
```

Figure 12.1B: Macro that Uses Subroutines

```
c:<all :
     $Ø = "First" : sa-a : $Ø = "Second" : sa-a:       { Load files }
     oa-Q : esc : rtn>3<rtn : rtn>Both<rtn :  { Create file for both }
     $Ø = "First" : sa-x :                    { Add text from "First" into "Both" }
     $Ø = "Second" : sa-x :                   { Add text from "Second" into "Both" }
     oa-1>!                                   { Move to top of "Both" and end }

a:<all : oa-q : esc : rtn : rtn : sa-f>!      { Add file in $Ø to desktop }

x:<all : oa-q : sa-f :                        { Move to appropriate file }
     oa-1 : oa-c>T<oa-9 : rtn :               { Copy file contents }
     oa-q : $Ø = "Both" : sa-f : oa-c>F>!     { Move contents into "Both" }

f:<all : find : rtn>!                         { Select whatever file is in $Ø from list}
```

Why Use Subroutines?

Subroutines offer four advantages:

1. Clarity. It is easier to understand the flow of a macro; subroutines move the details out of the flow of the main program.
2. Easier debugging. It is easier to find errors when you "modularize" a program into subroutines.
3. Efficiency. Subroutines let you write shorter macros, saving space and reducing your typing.
4. Flexibility. Subroutines make it easier to enhance a program; you can "call" the subroutine in any additional steps you add to the macro.

Figure 12.1B demonstrates another advantage of subroutines: Variables established in the main macro hold their values when you call a subroutine. (*Figure 12.1B* defines the variable $Ø in the main macro, even though it doesn't use that value until the subroutine.)

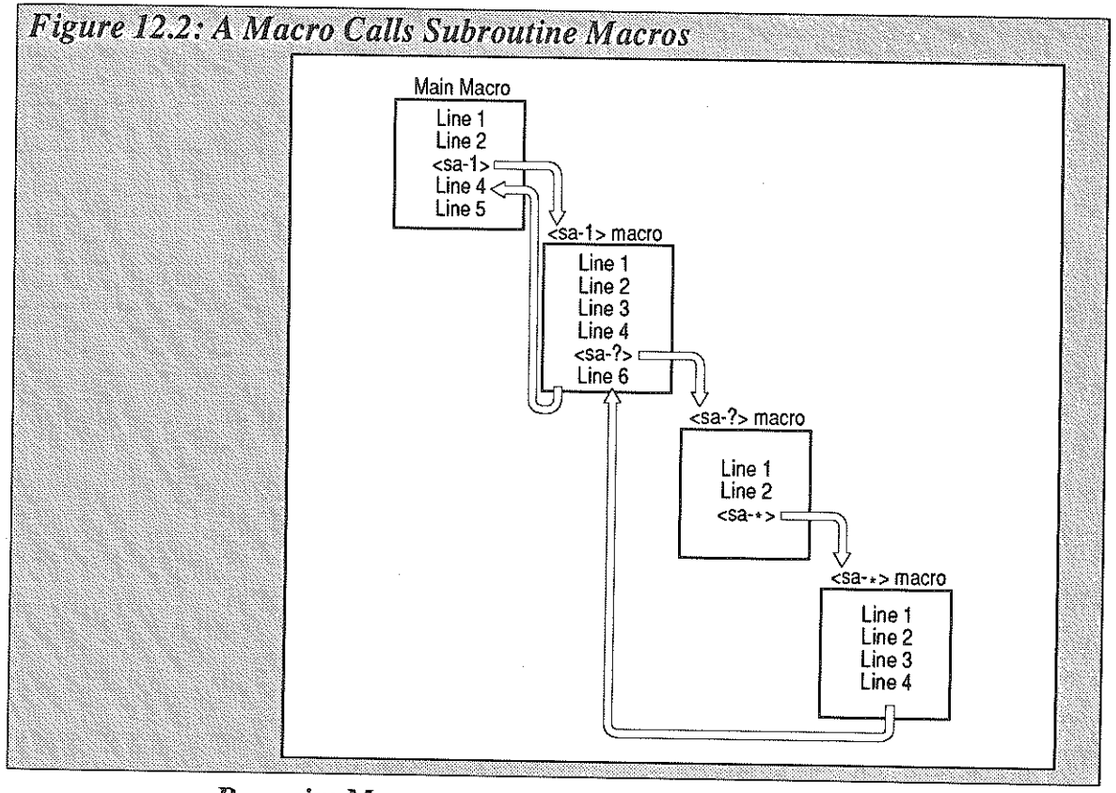
Subroutine Limits and Features

Each time you call a macro from within another macro, UltraMacros leaves the main macro, runs the new macro, and either returns to the main macro or to another macro called by the subroutine. Thus, a macro can call a macro which calls a macro which again calls another macro ... and so on.

UltraMacros can remember up to 18 locations and can step back up the macro "path" for 18 steps, so you can repeat this process up to 18 times in a single path and write macros up to 18 levels "deep". UltraMacros always runs the subroutine macro and then returns to the macro that called the subroutine.

Figure 12.2 depicts an example of a subroutine that is three macros deep. The main macro in *Figure 12.2* calls macro <sa-1> , a subroutine macro. <sa-1> calls another subroutine macro, <sa-?>, which calls <sa-*>. Control then returns to line 6 in macro <sa-1> and then back to the main macro. Line 4 in the main macro could call another subroutine macro that would start down another subroutine path; each subroutine path can be up to 18 levels deep.

Figure 12.2: A Macro Calls Subroutine Macros



Recursive Macros

Just as macros can call other macros, macros can call themselves. The macro then becomes its own subroutine. For example, consider this <sa-x> macro that prints a line of asterisks:

Figure 12.3: Using Recursion in Subroutines

Original <sa-left> Macro

```
<left>:<awp><oa-tab oa-tab oa-tab oa-tab oa-tab oa-tab
    oa-tab oa-tab oa-tab oa-tab oa-tab oa-tab oa-tab oa-tab
    oa-tab oa-tab oa-tab oa-tab oa-tab>!
```

Revised <sa-left> Macro

```
<left>:<awp:oa-tab oa-tab:sa-left>!
```

```
X:<all>*<sa-x>!
```

The macro prints an asterisk (*) and then calls itself. Since UltraMacros can remember up to 18 different locations, this macro prints 18 asterisks and stops.

The <sa-x> macro is a "recursive macro", a macro which calls itself. Each time the <sa-x> macro calls itself, it acts as a new subroutine. Thus, if you do not use logical statements to stop their operation, recursive macros repeat themselves 18 times and then stop.

It is rare to have recursive macros within the body of a main macro, but recursive macros make it easier to do repetitive tasks within subroutines. For example, *Figure 12.3* depicts the useful <sa-left> macro that is part of UltraMacros 2.x's set of default macros. This macro moves the cursor to the beginning of the line in an AppleWorks 2.x word processor document. (The Apple-. and Apple-, commands in AppleWorks 3.x serve an identical function, so UltraMacros 3.x does not include this macro. If you want to position the cursor at the beginning of a line in AppleWorks 3.x, include the command <oa-,> or <oa-.> in your macro.)

As you can see from *Figure 12.3*, the original <sa-left> macro consists of many <oa-tab> commands. *Figure 12.3* shows how you can use a recursive macro to perform the same task. The revised macro sends 36 <oa-tab> commands to AppleWorks;

two <oa-tab> commands per subroutine multiplied by the 18 times a macro can call itself.

It is apparent that you can use subroutines to make your macros more efficient and understandable.

An Application

Figures 12.4 through *12.8* depict a practical application of subroutine macros. This application analyzes a spreadsheet that compares each sales representative's actual sales with his or her quota of expected sales. The macro prints a congratulatory letter to employees who meet or exceed their quotas and encouraging letters to the employees who fall short of their quota.

Figure 12.4 depicts the operation of this application, and *Figure 12.5* presents the main macro and subroutine macros. *Figures 12.6* through *12.8* present the spreadsheet and word processor files you must create and load onto the AppleWorks desktop before you can run this macro.

How Subroutines Work in the Macro

To understand a macro, you must first try to picture the overall goal of the macro. In this case, the goal is to compare sales data with quotas in a spreadsheet and generate short memos to people who are above or below their quota.

Start at the top of the main macro and try to understand each step. Follow the macro when it sends you to a subroutine and try to understand the purpose of each subroutine. The subroutines are the short macros that follow the <sa-m> macro in *Figure 12.5*. Here is the function of each subroutine in this macro:

<sa-f> Switches to the main spreadsheet file from anywhere in AppleWorks.

Figure 12.4: Application that Generates Letters

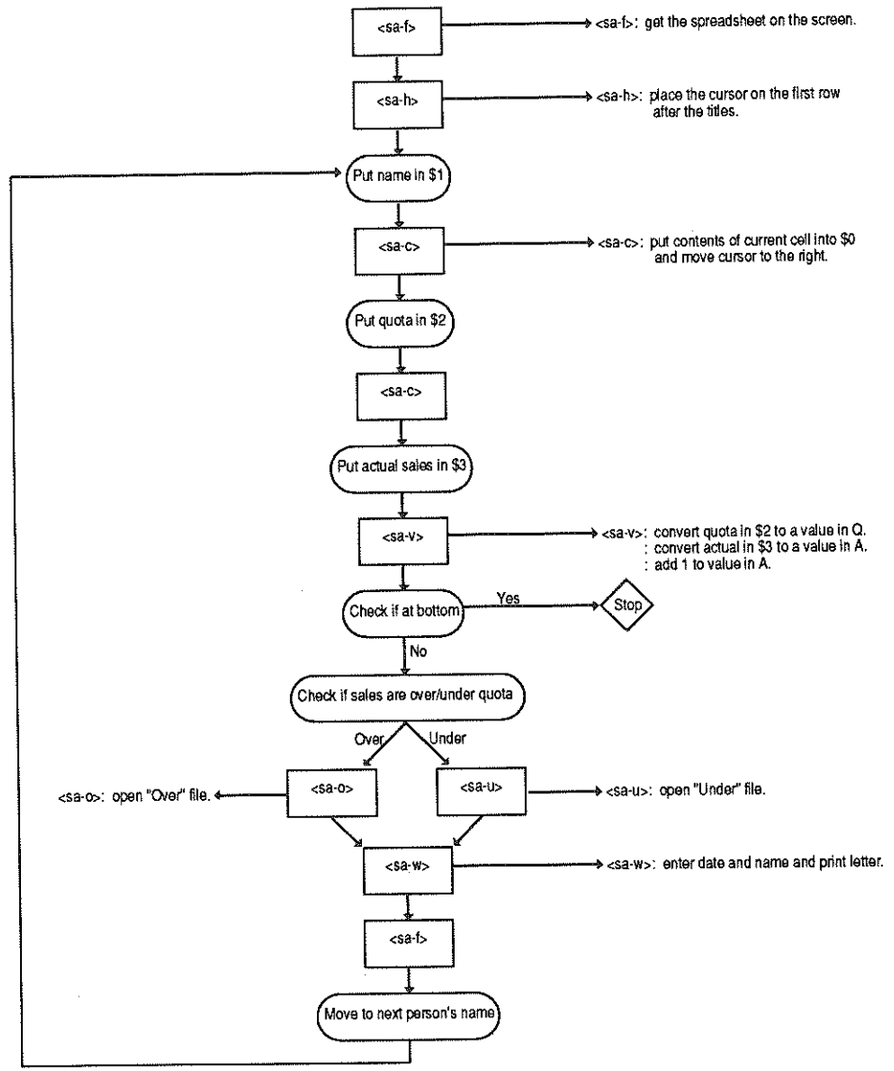


Figure 12.5: Report Generator Macro

```
M:<all:sa-f:                                    { Put spreadsheet on screen }
   sa-h:                                        { Put cursor on first name }
   begin :                                      { Start loop here }
   sa-c:                                        { Capture current cell in variable $ø }
   $1=$ø :                                      { Put name into variable $1 }
   sa-c: $2=$ø                                  { Put quota in variable $2 }
   sa-c: $3=$ø                                  { Put actual in variable $3 }
   sa-v:                                        { Convert $2 and $3 into values Q and A }
   if $1 = "--------------" then oa-1:stop:elseoff:  { Stop if end }
   if A < Q then sa-U : sa-W : elseoff :        { Print "Under" letter }
   if F > Q then sa-O : sa-W : elseoff :        { Print "Over" letter }
   sa-f:                                        { Go back into spreadsheet }
   oa-left : down:                              { Back to left side ... next person }
   rpt>!                                        { Repeat loop }

F:<all:oa-Q:                                    { Call up Desktop Index }
   $ø="Sales Summary" : find:rtn>!              { Search for spreadsheet file }

H:<asp: oa-f>c<oa-Y>A1<rtn:                     { Top of file }
   down : down>!                                { Skip title info }

C:<asp: call : right>!                          { Get current cell contents into $ø }
                                                { and move cursor to the right }
O:<all: oa-Q:                                   { Call up Desktop Index }
   $ø="Over":find:rtn>!                         { Search for Over file }

U:<all:oa-Q:                                    { Call up Desktop Index }
   $ø="Under":find:rtn>!                        { Search for Under file }

V:<all:Q = val $2 :                             { Q = Quota}
   A = val $3 :                                 { A = Actual}
   F = A + 1>!                                  { F = Adjusted Actual}

W:<awp : oa-1 : insert                          { Goto first line }
   rtn>DATE : <tab : date : rtn :               { Enter header }
   rtn>TO : <tab : print $1 : rtn : rtn :
   oa-P : rtn : rtn : rtn :                     { Print letter }
   up : oa-D : oa-1 : rtn>!                     { Delete header }
```

Figure 12.6: Sales Summary

```
File: Sales Summary                REVIEW/ADD/CHANGE

========A==========B=======C=======D=======E==========F======
  1|Salesperson          Quota   Actual
  2|-------------        -----   ------
  3|Larry Davis            400      301
  4|Cherri Dawson          150       54
  5|Elaine Grant           250      312
  6|Richard Huber          325      397
  7|Darel Jevins           400      408
       :                    :        :
 13|TOTALS                2900     2988
```

Figure 12.7: Sales Over Quota

```
File: Over                         REVIEW/ADD/CHANGE
====|===|===|===|====|====|====|====|====|====|=====|====|====|====|===

From:  Terry Wilson, area manager

You met your ambitious sales quota for the past three months; I congrat-
ulate you on your fine achievement. It is difficult to achieve these
goals and I hope you can continue your exceptional performance.

Sincerely,
ACME Widgets, Inc.
```

Figure 12.8: Sales Under Quota

```
File: Under                        REVIEW/ADD/CHANGE
====|===|===|====|====|====|====|====|====|====|====|====|====|===
From:  Terry Wilson, area manager

I am certain you were disappointed not to meet your sales goals for the
past quarter. From my observations, I believe you are a dedicated mem-
ber of the company and are working hard to contribute to our effort.
Please let me know if there is any way I can help you achieve your
goals.

Sincerely,
ACME Widgets, Inc.
```


<sa-h> Moves to the top of any spreadsheet file.

<sa-c> Reads the value of a cell into $Ø and moves the highlight one cell right.

<sa-o> Moves to the word processor file used for representatives meeting or exceeding their quota.

<sa-u> Moves to the word processor file for representatives under their quota.

<sa-v> Converts text read from spreadsheet cells into numeric values.

<sa-w> Places header information at the top of whatever document is current, prints the document, then deletes the header.

Make certain you always understand an operation before you proceed to the next instruction.

How the Macro Works

The user enters a Solid-Apple-M to start the main macro; the <sa-m> macro controls the entire process and calls all the necessary subroutines. Study *Figures 12.4* and *12.5* carefully until you can understand the operation of this series of macros.

Other Applications

This application can serve as a model for many similar macros. For example, you could modify these macros to generate progress reports for students in a class or to generate different letters depending on whether a bill was 30, 60, or 90 days overdue.

Conclusion

In this chapter, I described how to use subroutine macros to improve your macros. Looping and subroutines let you write macros that perform repeated tasks efficiently. Use loops when you want to perform a repetitive operation one time after another; for example, to write a macro that repeats the Find Command until it finds a special character. Use subroutines when you need to perform a task often, but in different parts of the macro.

Subroutines let you write more efficient, understandable, and powerful macros.

In the next chapter, I will describe how to make full use of UltraMacros' important <if-then-else> statement.

13: IF-THEN-ELSE Statements

UltraMacros' support for IF-THEN-ELSE logic adds significant power to the branching capability of a macro. This chapter explains the syntax of UltraMacros' IF-THEN-ELSE statements and offers two practical applications of this branching capability.

Chapter 8 explained how to use simple <if> statements to control program flow. That chapter described only the simplest <if> statements; statements that allow a single branch in a macro. This chapter will describe how to use the IF-THEN-ELSE capability available in <if> statements to more fully utilize the branching power of TimeOut UltraMacros.

Simplest <if> Statement

In its simplest form, the <if> statement has four parts: The word "if", an "expression", an "outcome", and the word <elseoff> or, with AppleWorks 3.x, <endif>. The "outcome" occurs if the "expression" is true, otherwise the macro jumps past the <elseoff> of <endif> statement and continues as though the <if> statement was not in the macro program.

For example, consider the following macro:

```
a:<all: A=8: if A=8 then msg ' HELLO THERE ': elseoff>!
```

This macro stores the number eight in numeric variable "A", then uses an <if> statement to test if "A" contains an eight. If it does, the macro displays the message "HELLO THERE" at the bottom of the AppleWorks screen. If variable "A" contains anything but an eight, the macro terminates and does nothing. Since this macro always stores the number eight in variable "A", it always displays "HELLO THERE".

IF-THEN-ELSE Statements

While there are practical applications for simple <if> statements, you get increased power if you use the more complete form for these statements. In its complete form, the <if> statement has six parts: The word "if", an "expression", the "if true outcome", the word "else", the "if false outcome", and the word "elseoff". You can depict the syntax as follows:

```
<if test then iftrue: else iffalse: elseoff>
```

If the test is true, UltraMacros executes all the statements between <then> and <else> and skips past the <elseoff> token to continue. If the test is false, UltraMacros executes all the statements between <else> and <elseoff> and continues running commands after the <elseoff> token.

Note that only one outcome occurs for each <if> statement: UltraMacros either executes the statements between <then> and <else>, or executes the statements between <else> and <elseoff>.

The syntax of IF-THEN-ELSE statements should make the role of <elseoff> clear. <elseoff> terminates an IF-THEN-ELSE statement and tells UltraMacros where to resume after a branching operation.

Sample IF-THEN-ELSE Macro

Consider the following macro that demonstrates the operation of IF-THEN-ELSE logic:

```
b:<all: A=7: if A=8 then $1 = ' TEST IS TRUE ':
                    else $1 = ' TEST IS FALSE ':
                    elseoff: msg $1>!
```

This macro sets numeric variable "A" equal to seven and tests to see if variable "A" equals 8. If the test is true (which, of course, it is not), the macro stores the string " TEST IS TRUE " in variable $1.

If the test is false (which it is), the macro skips to the <else> token and stores the string " TEST IS FALSE " in variable $1.

In either case, the macro then continues after the <elseoff> token and displays the contents of $1 at the bottom of the AppleWorks screen.

Figure 13.1: Sample Application of <ifnot> Token

```
c:<all:$1='hello': ifnot $1='hello' then A=1:
                                        else A=Ø:
                                        elseoff:
      msg ' Variable "A" contains ' + str$ A>!
```

Figure 13.2: Different Tests Available with <if> and <ifnot>

```
if A=B          { Check if A equal to B }
if A>B          { Check if A greater than B }
if A<B          { Check if A less than B }
ifnot A=B       { Check if A not equal to B }
ifnot A<B       { Check if A greater than or equal to B }
ifnot B>A       { Check if A less than or equal to B }
```

Expressions in a Test

<if> statements can only test for values that are equal to, greater than, or less than another value. You cannot test for "not equal", "greater than or equal to", or "less than or equal to". To overcome this limitation, UltraMacros offers an <ifnot> command that follows the same syntax as the <if> statement.

The <ifnot> command checks whether a condition is false, rather than true. For example, consider the macro in *Figure 13.1*. That macro stores the string "hello" in variable $1. Then the macro tests to see if $1 does not contain "hello". If $1 does not contain "hello", the macro stores the number one in variable "A". If $1 contains "hello", the macro skips to the <else> token and stores the number zero in variable "A". The macro then displays the message "The number is" followed by the contents of variable "A" at the bottom of the screen. (Of course, $1 contains "hello", so this macro always stores a zero in variable "A".)

Figure 13.3: Macros with <if> and <ifnot> Logic

```
d:<all: $1="Munz": ifnot $1="Mark" then msg ' NOT MARK ': elseoff>!

e:<all: A=1Ø: begin: msg A: A=A-1:
        if A>Ø then rpt:
                else msg ' DONE ':
                elseoff>!
```

The combination of <if> and <ifnot> tokens lets you test for all possible relationships between variables. *Figure 13.2* summarizes the possible test combinations.

<if> and <ifnot> statements often result in a contorted logic. It sometimes helps to rearrange the words representing the commands and speak the rearranged sentence out loud. For example: < ifnot A=B > becomes "If A is not equal to B", and < ifnot B>A > becomes "If B is not greater than A". The oral expression of re-ordered <ifnot> statements makes it easier to understand the logic of these negatively stated expressions.

Let's examine two more applications of this logic before exploring more complex, practical uses for these commands.

Figure 13.3 presents two examples of macros with <if> and <ifnot> logic. Note that only the <then> command is required in an <if> command; the <else> statement is optional. You use <if> without <else> if you want a single branch in a macro; use <if> and <else> if you want two or more branches.

The first macro in *Figure 13.3* sets variable $1 to "Munz". The <ifnot> statement checks if $1 is not equal to "Mark". Since $1 contains "Munz", it is not equal to "Mark", and the expression is true; that is, "Munz" does not equal "Mark". Since the expression is false, the macro executes the statements between <then> and <elseoff>, and the message "NOT MARK" appears on the AppleWorks screen.

The second macro sets the value of variable "A" to ten. The macro then starts a loop that displays the value of "A", decrements that value by one, and tests if "A" is greater than zero. This loop needs a <begin> statement, otherwise the macro would re-set variable "A" to a value of ten each time it repeated the loop. When the value of "A" reaches zero, the expression <if A>Ø> is not true, so the macro displays "DONE" and stops.

Two Macros with IF-THEN-ELSE Logic

Now we will examine two practical applications of the <if> statement. The first macro assumes you entered a person's last name and first name into a single category. The macro divides that entry into two categories. The second macro searches for duplicate data base records and lets you delete one of those records.

These are the most complex macros in this book; they provide an opportunity to demonstrate the operation of subroutines, variables, <if> statements, and other concepts introduced in the earlier chapters. The macros appear in *Figure 13.4*. An explanation of the macros follows.

Divide a Data Base Entry

Imagine you have a data base with student names in a category called "NAMES". You entered the names in the format "lastname, firstname". Now you want to split the names into two categories. *Figure 13.4* contains the necessary macros, but first you must reorganize your data base file so it is compatible with these macros.

Follow these steps before compiling and running the macros in *Figure 13.4*:

1. Get the data base on the screen and change the display to multiple record layout.

Figure 13.4: Macros that Separate Names

```
s:<adb:                          { Macro only works in a data base file }
   onerr stop:                   { Stops the macro when it reaches the bottom }
                                 { of the file and the Down Arrow
                                 { causes an AppleWorks error beep }
   oa-1:                         { Go to the top of the file }
   begin:                        { Start the loop }
     $Ø=cell:                    { Store current entry in $Ø }
     $1=$Ø:                      { Copy string in $Ø to $1 }
     sa-k:                       { Calls the subroutine that splits the }
                                 { current category into two }
     oa-Y:                       { Clears the entry in that category }
     print $1 : rtn:             { Enter the last name }
     print $2 : rtn:             { Enter the first name }
     oa-tab: oa-tab:             { Move the cursor back to the first category }
     down:                       { Move to the next record }
   rpt>!                         { Repeat until reaching the end of the file }

k:<adb:                          { Subroutine that separates an entry }
  X=len $1:                      { Determine the length of the name }
  $2=right $1,1:                 { Capture the character under the cursor }
  if $2="," then                 { Check if current character is a comma }
     X=X-1: $1=left $Ø,X:        { Capture the string to the left of the comma }
     X=X+2: Y=len $Ø: Y=Y-X:     { Determine the length of the first name }
     $2=right $Ø,Y:              { Skip the space and comma and store }
                                 { the first name in $2 }
     else                        { If the current character is not a comma }
       X=X-1: $1=left $1, X:     { Reduce $1 by one character }
       rpt:                      { Return to the beginning of the subroutine }
  elseoff>!
```

2. You will need a blank category to hold the students' first names. If you do not have an extra category, create that category now.
3. Use the Apple-N command to name that category "FName".

Figure 13.5: Data Base File Ready to Accept First Name

NAMES	FName	Address1	Address2
Bennett, Tim		123 Anywhere Street	Apt. 4Ø2
Merritt, Emily		234 Elsewhere Ave	
Merritt, Jennifer		234 Elsewhere Ave.	
Merritt, Michael		234 Elsewhere Ave.	
Munz, Mark		8765 Porshe Road	
Smolarz, Joshua		66 South Lane	
Smolarz, Yvonne		66 South Lane	
Williams, Lisa		5678 Lake Street	Apt. 312

4. Use the Apple-L command to move the FName category next to the NAMES category. Your screen should look like the sample in *Figure 13.5*.
5. Issue an Apple-L command and set the cursor so it moves to the right when you press the Return Key.

Now you can enter, compile, and run the macros in *Figure 13.4*.

Here is how these macros work:

The task requires you to repeat a series of steps for each record in a data base. This example divides the macro into two segments. The main macro takes responsibility for cursor movement and the placement of the first and last names into the correct categories. The subroutine separates the entry of a person's name into two variables. IF-THEN-ELSE logic controls the operation of the subroutine.

Here's what happens when you invoke this macro:

1. The <sa-s> macro begins with an <onerr stop> command that tells UltraMacros to stop when AppleWorks sounds its error bell. In this example, pressing the Down Arrow Key after AppleWorks processes the last record normally sounds

the AppleWorks error bell. Now the <onerr stop> command "listens" for that error beep and terminates the macro. You can use the <onerr stop> command to terminate a macro when you want a macro to repeat itself until all records are processed.

2. The <oa-1> command puts the cursor at the beginning of the file.
3. The <begin> command defines the beginning of a loop. It tells UltraMacros where to start processing when it encounters the <rpt> command.
4. The macro stores the current entry in the NAMES category in variable $Ø.
5. The macro then duplicates the value of $Ø in $1, so the name is now in two locations; $Ø and $1.
6. <sa-s> then calls the <sa-k> subroutine.
7. <sa-k> determines the length of the name stored in $1, and stores the last character of that string in variable $2.
8. The macro tests if the character in $2 is a comma, the character that appears between the last and first names in our example. If $2 does not contain a comma, the macro skips to step #11 below. If $2 contains a comma, <sa-k> stores the characters preceding the comma in variable $1, thus replacing the full name previously stored in $1with the last name.
9. The macro then determines the length of the first name. The formula is:

   (length of total name - length of last name - X).

   The macro subtracts two from the length to allow for the comma and for the space between the last name and first name.

10. <sa-k> then captures the first name in variable $2 and returns to the <sa-s macro>. That is, it skips to step #12 in this step-by-step description.
11. If the character in $2 is not a comma, the macro executes the <else> portion of the logic. The macro decreases the value of "X" by one and repeats the process of storing the current character in $2 and determining if $2 contains a comma. This continues until the <sa-k> macro captures the comma in $2 and executes the <then> logic described in the steps 8-10 above.
12. With the last name stored in $1 and the first name in $2, control returns to the <sa-s> macro.
13. The <sa-s> macro issues an <oa-y> to yank out the current entry in the NAMES category and puts the contents of $1 in that category.
14. The macro enters a Return to lock in that entry. The Return also moves the cursor to the next category, the FName category.
15. <sa-s> copies the first name from variable $2 into the FName category and enters a Return to lock in that entry.
16. The macro issues two <oa-tab> commands to move the cursor back to the first category and then issues a Down Arrow command to move the cursor to the next record in the file.
17. If the cursor is already on the last record, the Down Arrow command will cause an AppleWorks error beep that is captured by the <onerr stop> command. (You issued the <onerr stop> command at the beginning of the macro.) If there are more records in the data base, the macro executes the <rpt> command which turns control back to the <begin> token and the loop repeats itself.

Macro <sa-s> uses both a loop and a subroutine. Operation of the loop stops when the macro issues a Down Arrow command with the cursor on the last record in the file. Operation of the subroutine stops when the <if> token encounters the comma between the first and last names in the entry in the data base file.

Nested <if> Statements

A discussion of IF-THEN-ELSE logic is not complete without mention of one other important variation of the <if> statement; the "nested <if>". A "nested <if>" consists of one or more <if> statements called by another <if> statement.

Figure 13.6 presents an example of nested <if> statements.

Figure 13.6: Example of Nested <if> Statements

```
<ba-1>:<all: msg 'Do you really want to? (y/n) ':
   $1=getstr 1: msg '':
   if $1="N" then Z=Ø:
       else if $1="n" then Z=Ø:
            else if $1="Y" then Z=1:
                 else if $1="y" then Z=1:
                      else bell: bell: rpt: elseoff>!
```

The macro <ba-1> starts by displaying the message "Do you really want to? (y/n)" and captures the response in variable $1.

The macro then executes one or more <if> statements to check whether you entered an "N", "n", "Y", or "y".

If $1 contains an "N", the macro sets "Z" equal to zero, jumps past the <elseoff> token and stops. If $1 does not contain an "N", the macro executes the <else> portion of the statement. The <else> portion contains another <if> statement that checks if $1 equals "n". If $1 contains an "n", the macro sets "Z" equal to zero and terminates. If it does not contain an "n", the macro checks to see if $1 contains a "Y". If $1 contains a "Y",

the macro sets "Z" equal to one. If it does not contain a "Y", the macro checks if $1 contains a "y". If $1 contains a "y", the macro sets "Z" equal to one. If $1 does not contain a "y", the macro sounds the bell twice and displays the "Do you really want to? (y/n)" question.

Since the <then> and <else> portions of every <if> statement can themselves include additional <if> statements, you can use nested <if> statements to generate an unlimited number of branches in your macros.

The next set of macros will use nested <if> statements, but first I must introduce two additional commands I will use in those macros: <ctrl-x> and <zoom>.

<ctrl-x> and <zoom>

<ctrl-x>: UltraMacros sometimes hides the cursor from view while running a macro. The <ctrl-x> token forces UltraMacros to display the cursor during operation of a macro.

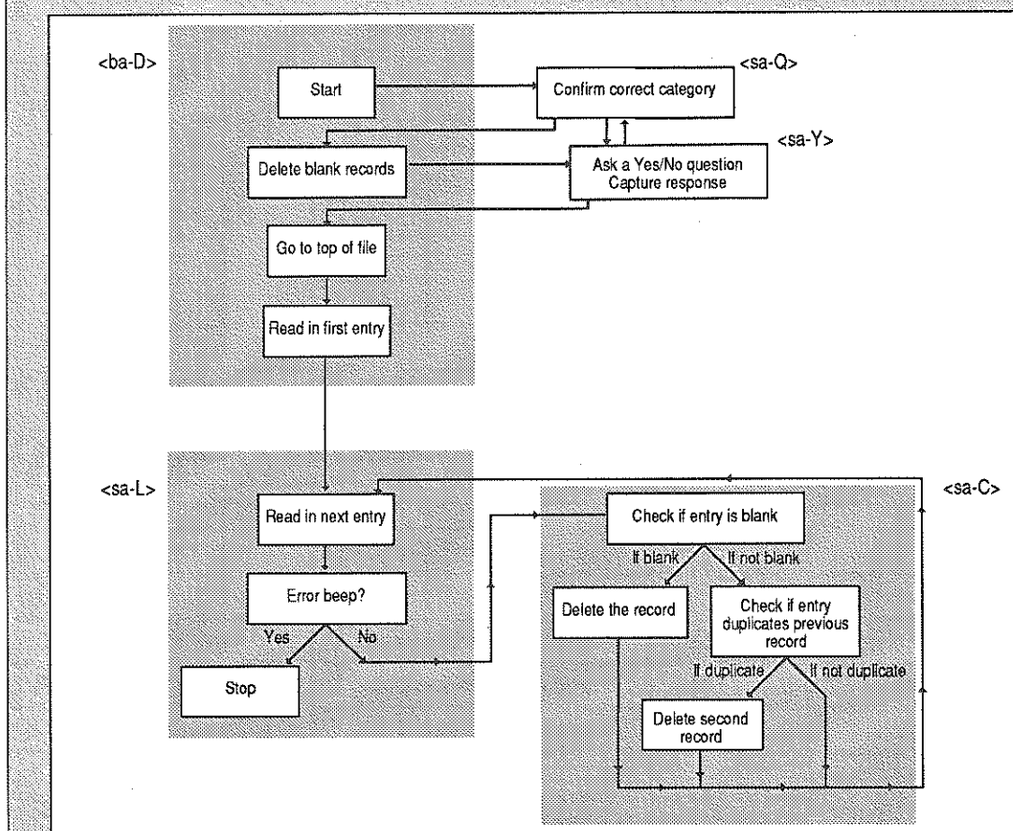
<zoom>: The <zoom> command forces AppleWorks to "zoom out". <zoom> hides the display of printer commands in the word processor, shows values rather than formulas in the spreadsheet, and displays multiple record layout mode in the data base module. You can follow this command with <oa-z> to show the printer options in the word processor, display formulas in the spreadsheet, or enter single record layout mode in the data base module.

Macros that Delete Duplicate Records

Now let's examine *Figures 13.7* and *13.8*, a series of macros that deletes duplicate records from a data base file.

It is easier to find errors and enhance a macro if you separate complex operations into smaller sub-tasks. Given the complex-

Figure 13.7: Flowchart of Macros that Delete Duplicate Records



ity of this task, this example "modularizes" the operation into a series of five subroutines as follows:

1. <sa-D> checks if you are testing the correct category and starts the process.
2. <sa-L> controls the loop that tests each record.
3. <sa-C> checks for and deletes blank records.
4. <sa-Q> checks if you are testing the correct category.

Figure 13.8: Macros that Delete Duplicate Data Base Records

```
[This is a macro you start from the keyboard. It double-checks the category and starts the operation.]

<ba-D>:<adb:
   Q:                                          { Check for correct category }
   $Ø=" Delete Blank Entries ": sa-Y: B=Z:     { Does user want to delete blank records? }
   oa-1:                                       { Put cursor at beginning of file }
   $1=cell:                                    { Read the first entry into $1 }
   down:                                       { Move cursor to the second record }
   goto sa-L>!                                 { Goto the loop macro }


[This macro controls the loop through the records.]

L:<adb:
   onerr stop:                                 { Stop at the end of the file }
   begin:                                      { Start the loop here }
     $9=cell:                                  { Read current entry into $9 }
     sa-C:                                     { Compare with previous record }
     $1=$9:                                    { Copy current entry into $1 }
     down:                                     { Goto next record }
   rpt>!                                       { Do comparison on the next record }


[This macro checks if current record is blank and deletes blank records if authorized.]

C:<adb:
 ifnot $9 = "" then                                          { Check if it's not a blank record }
              if $1=$9 then                                  { If the two records are identical }
                       oa-D: rtn: up:                        { delete the second record }
              else                                           { Do this if the record is blank }
               if $1=$9 then                                 { Is this a duplicate blank record? }
                        if B=1 then                          { User authorized deleting blank records }
                                oa-D: rtn: up>!              { Delete the record }


[This macro checks if the current category is the correct one.]

Q:<adb:
   zoom:                                                     { We need to zoom out for this task }
   ctrl-x:                                                   { Display the cursor }
   $Ø=' Is this the correct category to search for duplicates':
   sa-Y:                                                     { Ask a yes/no question }
   if Z=Ø then bell: bell: msg ' Aborted !!': stop:          { If answer is "no" abort the macro }
   elseoff>!
```

Figure 13.8: Macros that Delete Duplicate Data Base Records, continued

```
[This macro asks a yes/no question and captures the user's response.]
Y:<all:
   msg $Ø + '? (Y/N) ':                    { Display this message }
   ctrl-x:                                 { Show the cursor }
   X = key:                                { Capture the keystroke from the user }
   Z=Ø:                                    { Set default response to "No" }
   if X=27 then Z=Ø: else                  { Check for Escape Key press }
   if X=121 then Z=1: else                 { Check for "y" }
   if X=89 then Z=1: elseoff:              { Check for "Y" }
   msg ''>!                                { Clear the message line }
```

5. <sa-Y> handles the user's response to a yes/no question.

Figure 13.7 depicts the logic of the different subroutines in this set of macros. *Figure 13.8* presents the actual macros.

Enter and compile the macros in *Figure 13.8*, then follow these steps:

1. These macros assume that if two records have identical entries in a single category, the records are duplicates. You must first sort the records based on the category you want to use to check for duplicate records. Put the cursor in the category you want to use to check for duplicate records and issue a <sa-A> command.
2. Leave the cursor in the category you want to check and press <ba-D> to start the operation.
3. The macros ask if you are in the correct category and if you want to delete blank records. Answer each query as indicated.

The macro goes to the top of the file, reads in the first category, and jumps to the <sa-L> macro. That macro loops through the file and compares the current record with the previous record using the <sa-C> macro. If <sa-C> finds they are identical, it deletes the entry as long as it is not blank. If it is blank,

<sa-C> checks if the user authorized the deletion of blank records. If authorized, it deletes the blank record.

Conclusion

This chapter discussed IF-THEN-ELSE logic using the <if> statement. It also described advanced applications of <if>, including <ifnot>, <else>, and nested <if> statements and introduced the <ctrl-x> and <zoom> commands and presented two applications of complex macro sets that use IF-THEN-ELSE logic to perform useful data base file maintenance tasks.

Now we will examine more advanced programming commands and concepts.

14: More Power Programming

This chapter describes the keyboard equivalents of UltraMacros tokens, forcing AppleWorks into a certain condition, managing "sleeping macros" and other advanced programming commands such as <store>, <recall>, <disk>, <path>, <clear>, <print>, <chr$>, <pr#>, <peek>, and <poke>.

Each chapter in this book introduces new UltraMacros commands you can use to prepare compiled macros. This chapter will describe commands that let you (a) force AppleWorks into a desired condition, (b) control when you execute a macro, (c) store UltraMacros data in an AppleWorks file, (d) clear all the variables you established with UltraMacros, (e) output directly to a printer, and (f) examine and change what is in memory. First, I will describe how to generate some UltraMacros commands from the keyboard.

Keyboard Equivalents

As you know, UltraMacros adds many commands to AppleWorks. Some of these commands work from both the keyboard while using an UltraMacros-enhanced copy of AppleWorks, and within macros by using tokens. *Figure 14.1* lists the keyboard equivalents of the different tokens. While not every token has a keystroke equivalent, the keystrokes are useful when you want to use these features without writing a compiled macro.

Forcing a Condition

While writing a macro you must sometimes force AppleWorks into a certain state or condition. For example, some macros only work correctly when a data base file is in multiple record layout mode.

The previous chapter described the <zoom> command which forces AppleWorks to "zoom out". <zoom> tells AppleWorks to hide the display of printer commands in the word processor, show values instead of formulas in the spreadsheet, and display multiple record layout mode in the data base module. It also described how the combination <zoom> <oa-z> forces AppleWorks to "zoom in".

UltraMacros offers other commands that let you force AppleWorks into different states. The <insert> command tells Apple-

Figure 14.1: Keyboard Equivalents of UltraMacros Tokens

Token	Keyboard Equivalent	Notes
<ahead>	sa-.	Finds first blank space to right of cursor.
<back>	sa-,	Finds first blank space to left of cursor.
<cell>	oa--	Reads current spreadsheet cell or data base entry.
<clear>	oa-ctrl-X	Clears all numeric and string variables.
<date>	sa-'	Displays date in format September 1, 1991.
<date2>	sa-"	Displays date in format Ø9/Ø1/91.
<dec>	oa-ctrl-A	Decrements character at current cursor position.
<disk>	oa-&	Reads current volume name or subdirectory pathname.
<find>	sa-Return	Moves cursor to next carriage return marker.
<findpo>	sa-^	Moves cursor to next caret mark.
<getstr>	oa-Ø	Store user entry in $Ø.
<inc>	oa-ctrl-W	Increments character at current cursor position.
<insert>	oa-!	Turns on the insert cursor.
<lc>	oa-;	Changes character to lower case.
<nosleep>	oa-ctrl-N	Cancels the currently defined sleeping macro.
<oa-ctrl-@>	oa-ctrl-@	Sends a Control-@ from within a macro.
<path>	oa-*	Reads current volume name or subdirectory name and the currently highlighted file name.
<read>	oa-^	Reads character at current cursor position.
<recall>	oa->	Recalls data stored with <store>.
<store>	oa-<	Stores first 15 characters of macro Ø into the current word processor or spreadsheet file.
<time>	sa-=	Displays time in format 3:1Ø pm.
<time24>	sa-+	Displays time in format 15:1Ø.
<uc>	oa-:	Changes character to upper case.
<zoom>	oa-@	Forces zoom out.

Works to put the cursor in insert mode; i.e., to move existing text to the right as you enter new text. Similarly, <insert : oa-e> forces AppleWorks into overstrike mode.

<ahead> forces the cursor to the first blank space after the current cursor position. As you might expect, <back> moves the cursor to the first blank space to the left of the cursor.

In the word processor, the <find> command forces the cursor to the next carriage return marker. Similarly, <findpo> forces the cursor to the next caret mark.

Figure 14.2 depicts a macro that rearranges the presentation of data in a data base file. This macro uses the <zoom> command to make certain AppleWorks is in multiple record layout mode before it issues an <oa-L> command to change the layout of the columns.

Figure 14.2: Macro to Change Multiple Record Layout

```
V:<adb: zoom:          { Force zoom out }
        oa-L:          { Change layout  }
  oa-D: oa-D:          { Remove categories from layout }
        oa-I:          { Now insert a category }
    up: rtn :          { Choose first category on the list }
  oa-I: rtn:           { Insert another category }
        esc>!          { Return to multiple record layout }
```

The macro in *Figure 14.2* reverses the position of the first two categories in multiple record layout. You can use this macro if you sometimes want to display data with last name first and other times with first name first.

The macro first forces AppleWorks to zoom out into multiple record layout mode and then issues an <oa-L> command to change the layout of the display. Next, the macro uses two

Figure 14.3: Alarm Clock Macro

```
A:<all:wake sa-Q at 17:ØØ>!

Q:<all:bell:bell:bell:msg ' Time to go home! '>!
```

<oa-D> commands to delete the first two categories and issues <oa-I> commands to insert the two categories in reverse order.

"Sleeping" Macros

If you have an Apple IIGS or a ProDOS-compatible clock in your Apple IIe or IIc, you can specify the time that UltraMacros should execute a macro. For example, you can tell AppleWorks to wait until 8 pm and then print a long data base report, or you can tell TimeOut TeleComm to wait until midnight and then transmit an AppleWorks spreadsheet to a remote location. The <wake> command controls this process.

The syntax is:

```
<wake MACRO at HH:MM>
```

where MACRO is the name of the macro you want to start, and HH:MM represents a 24 hour time for the macro to start its operation. For example,

`<wake sa-Q at Ø5:ØØ>` says
"Start macro <sa-Q> at 5 am".

Note that there is no "sleep" command; <wake> specifies that you want the macro to "sleep" in the background until the time indicated.

Figures 14.3 and *14.4* include examples that use the <wake> command.

The macros in *Figure 14.3* show how to use UltraMacros as a alarm clock. You execute macro <sa-a> which issues a

Figure 14.4: Auto-Save Macros

```
W:<all:M=Ø:H=8:              { Set H to hour -- 8, and M to minute }
   wake sa-S at Ø8:ØØ>!      { Wake <sa-s> macro at 8:ØØam }

S:<all:oa-s                  { Save file }
   M=M+1Ø :                  { Add 1Ø minutes for next time to wake up }
     if M=6Ø then
       M=Ø :H=H+1:elseoff    { If M=6Ø, then you are at the next hour, }
                             { so set M to Ø and update H }
   wake sa-S at H:M>!        { Put macro back to sleep and }
                             { Wake it up 1Ø minutes from now }
```

<wake> command. <wake> tells UltraMacros to execute macro <sa-Q> at 5 pm (17:00 in 24-hour time). At 5 pm, <sa-q> sounds the bell and displays a message to remind you that it is quitting time.

The macros in *Figure 14.4* use <wake> to issue an <oa-S> command to save your work every ten minutes starting at 8 am. Macro <sa-w> sets the values for variables H and M to 8:00 am and puts the <sa-s> macro "to sleep" until that time. At 8:00 am, <sa-s> starts automatically. The <sa-s> macro does an <oa-s> to save the current file, and then adds 10 (minutes) to the M (minutes) variable. If M is 60, then you are beginning a new hour, and the macro sets the H (hour) variable to the next hour and sets the M (minutes) variable back to zero. This sets the values of the H and M variables to 10 minutes from now. Then the <wake sa-s at H:M> command puts the current macro back to "sleep" for another 10 minutes.

You can cancel a "sleeping" macro either by using the <nosleep> command inside a macro, or by entering an <oa-ctrl-n> directly from the keyboard. If there is no sleeping macro active, the command does nothing.

Figure 14.5: Macros that Use <store> and <recall> to Link Files

```
N:<all:msg ' Enter Spreadsheet Name ': $Ø = getstr 15 :
      rtn: store>!

A:<all:oa-Q: esc: rtn: rtn:          { Goto Add Files Menu }
      input: rtn:                    { Allow user to choose file }
      recall:                        { Get linked file }
      oa-Q: esc: rtn: rtn:           { Back to Add Files Menu }
      find: rtn>!                    { Search for file and load }
```

<store> and <recall>

The <store> command captures the first 15 characters stored in variable $Ø and appends those characters to the current word processor, data base, or spreadsheet file. UltraMacros stores these characters in an area not usually used by AppleWorks, so the characters do not appear on the screen, nor do they affect the operation of the program.

The <recall> command finds those characters in the file and places them in variable $Ø. You can then manipulate the contents of variable $Ø as you would any other string variable. When you issue a <store> or <recall> command, UltraMacros displays the characters stored or recalled in the bottom right hand corner of the AppleWorks screen.

The fifteen character limit of <store> is not an accident; fifteen characters is enough space to hold a filename. That explains why the most common use of <store> and <recall> is to link two files together.

How to Link Files

Figure 14.5 presents two macros that use <store> and <recall> to link files; let me explain these macros:

Imagine that you have a data base file and a spreadsheet file you use together as part of an accounting package. You want the spreadsheet file available whenever you load the data base onto the AppleWorks desktop. The macros in *Figure 14.5* load the spreadsheet onto the desktop automatically whenever you bring in the linked data base file.

To use the macros, enter a <sa-n> with the data base on the screen. The <sa-n> macro asks for the name of the spreadsheet file you want to link to the data base, and attaches that file name to the data base file.

Now you enter a <sa-a> whenever you want to bring both the data base and spreadsheet files onto the desktop. The <sa-a> macro takes you to the Add Files Menu, lets you choose the data base file, loads the data base onto the desktop, checks if there is a linked file, and loads the linked file onto the desktop. You end up with both the data base and spreadsheet available for use.

<disk> and <path>

The <disk> and <path> commands store the name of the current disk volume or the current pathname in variable $Ø. With AppleWorks 2.x, these commands only work with the AppleWorks Files Menu on the screen. Starting with AppleWorks 3.0, the <disk> command works anywhere in AppleWorks.

The <disk> command puts the current data disk name in variable $Ø. If you specified the data disk location by slot and drive, <disk> puts the volume name in $Ø. If you specified the current data disk location by pathname, <disk> puts the subdirectory pathname in $Ø.

<path> stores the current volume name, subdirectory name, and the highlighted file name in variable $Ø.

Note that while <disk> and <path> can capture pathnames, <store> only uses the first 15 characters in that pathname.

Figure 14.6: Macros that Use <disk> and <path>

```
D:<all:disk: msg $Ø>!  { Display disk name }

P:<all:path: msg $Ø>!  { Display full pathname of highlighted file }

F:<all:disk: x=len $Ø: { Set X = to the length of the disk name }
      path: y=len $Ø: { Set Y = to the length of the full path }
      x=y-x-1:        { Calculate the filename length }
      $Ø= right $Ø,x: { Set $Ø to filename }
      msg $Ø>!        { and display it }
```

Thus, you must be cautious when using <store> in combination with the <disk> or <path> commands.

Figure 14.6 presents three macros that use the <disk> and <path> commands. <sa-D> displays the disk or subdirectory location of the file you highlighted on the disk catalog. <sa-P> displays the full pathname, including the filename. <sa-F> displays only the currently highlighted filename.

Note that a bug in UltraMacros versions 1.x and 2.x sometimes causes <disk> and <path> to give incorrect results. This problem is fixed in UltraMacros-enhanced copies of AppleWorks 3.x.

<id#>

Every TimeOut module has a different identification number; the table in *Figure 14.7* lists the identification number of each module and the name of the disk on which that application appears. The statement <A=id#> in a macro sets variable A equal to the identification number of the active TimeOut module. If you are not inside a TimeOut module, <A=id#> stores a Ø in variable A. You can then use an <if> statement to determine if you are in a TimeOut module.

Figure 14.7: ID# for TimeOut Modules

ID#	TimeOut Application	TimeOut Disk
1 =	Utility	All disks
2 =	Data Converter	*
3 =	QuickSpell	QuickSpell
4 =	Graph	Graph
5 =	SuperFonts	SuperFonts
6 =	SideSpread	SideSpread
7 =	FileMaster	FileMaster
8 =	Macro Compiler	UltraMacros
9 =	Macro Options	UltraMacros
10 =	Puzzle	DeskTools
11 =	Calculator	DeskTools
12 =	Calendar	DeskTools
13 =	Page Preview	DeskTools
14 =	Word Count	DeskTools
15 =	Envelope Addresser	DeskTools
16 =	File Encrypter	DeskTools
17 =	Case Converter	DeskTools
18 =	Clock	DeskTools
19 =	Notepad	DeskTools
20 =	Dialer	DeskTools
21 =	Debug	MacroTools
22 =	Menu Maker	MacroTools
23 =	Task Launcher	MacroTools
24 =	UM Tokens	MacroTools
25 =	File Status	MacroTools
26 =	Paint	SuperFonts,Graph
27 =	Thesaurus	Thesaurus
28 =	AWP to TXT	PowerPack
29 =	File Librarian	PowerPack
30 =	Help Screens	PowerPack
31 =	Desktop Sorter	PowerPack
32 =	Triple Desktop	PowerPack
33 =	Triple Clipboard	PowerPack
34 =	Category Search	PowerPack
35 =	Line Sorter	PowerPack
36 =	Program Selector	PowerPack
37 =	ASCII Values	PowerPack
38 =	Area Codes	DeskTools II
39 =	Calculator+	DeskTools II
41 =	Measurements	DeskTools II
42 =	Printer Manager	DeskTools II
43 =	Screen Out	DeskTools II
44 =	Stop Watches	DeskTools II
45 =	DirecTree	DeskTools II
46 =	Clipboard Viewer	DeskTools II
47 =	Screen Printer	DeskTools II
48 =	Disk Test	DeskTools II
49 =	File Search	DeskTools II
50 =	Analyzer	SpreadTools
51 =	CelLink	SpreadTools
52 =	Block Copy	SpreadTools
53 =	QuickColumns	SpreadTools
54 =	Rows <--> Cols	SpreadTools
55 =	FormulaToValue	SpreadTools
56 =	Directory Manager	Late Nite Patches
57 =	Vital Stats	Late Nite Patches
58 =	Bell Changer	Late Nite Patches
59 =	PathMaster	PathFinder
60 =	FileLister	MacroTools II
61 =	Publisher Menu	MacroTools II
62 =	UltraLock	MacroTools II
63 =	ReportWriter	ReportWriter

* Graph, DeskTools, UltraMacros, SpreadTools

Figure 14.8: Macro that Reduces Keystrokes in FileMaster

```
C:<all:   Z = id# :
          ifnot Z = 7 then
               msg ' This macro only works inside FileMaster ' :
               stop :      { End all macro activity now }
          endif:
          >1<rtn:          { Select file activities }
          >2<rtn:          { Copy files }
          rtn:             { Use current disk location }
          input:           { Allow user to input destination }
          oa-right:        { Select all the files }
          rtn:             { Begin copying }
          >Y<              { Answer "Automatically replace existing files?" }
          >Y!              { Answer "Keep original dates?" }
```

The <sa-C> macro in *Figure 14.8* is as an example of how to use the <id#> token; this macro lets you enter fewer keystrokes when you copy files with TimeOut FileMaster.

When you issue a <sa-C>, the macro checks which TimeOut application is active by setting variable Z equal to the identification number. If Z contains a seven, FileMaster is active. Otherwise, the macro displays an error message and stops. (Any macro that called <sa-C> would also stop.) The macro then selects the first option ("File Activities") on the FileMaster Menu and then "Copy Files". Next, the macro enters a <rtn> to select the default location as the source for the copies. The <input> command pauses operation while you choose the destination drive.

The macro continues by entering an <oa-right>, which is the FileMaster command that selects all files on the disk. The <rtn> tells FileMaster you specified the files you want to copy. Finally, the two "Y" keystrokes respond to the "Automatically replace existing files?" and "Keep original file dates?" prompts.

<clear>

The <clear> command resets all the UltraMacros variables to their default values. That is, it sets numeric variables A-Z to zero, and string variables $Ø-$9 to " " (empty strings). <clear> lets you use a single token to reset all the variables instead of writing a series of individual commands to reset each variable.

Use the <clear> command when you write many compiled macros and want to be certain you reset all the variables to their default settings before beginning a new series of operations. However, you must be cautious about using <clear>; the command resets all variables you used, including those used by a sleeping macro.

You can also use the keystroke <oa-ctrl-x> to clear the variables.

<print> and <chr$>

Earlier chapters described how to use the <print> command to print words and numbers into an AppleWorks document. You know that <print $Ø> prints the contents of $Ø at the current cursor position in AppleWorks. Similarly, <print A> converts the value stored at A into text characters and sends it to AppleWorks.

You can also use the <chr$> command in combination with <print> to enhance your output. For example, a Control-B (the AppleWorks command for Boldface Begin) has an ASCII value of two. So the commands

```
<print chr$2 + "AppleWorks" + chr$2>
```

insert the string "AppleWorks" in a word processor document with the commands for Boldface Begin and Boldface End. *[Ed: A complete ASCII conversion chart appears in Appendix A.]* A macro that prints the word AppleWorks in boldface appears in *Figure 14.9*.

Figure 14.9: Macro that Prints "AppleWorks" in Boldface

```
B:<awp: print chr$ 2 + "AppleWorks" + chr$ 2>!
```

Figure 14.10: Macro that Routes <print> to the Printer

```
P:<all: pr# 1 : print " This will print on your printer " : rtn :
        pr# Ø>!
```

<pr#>

The <pr#> command lets you control where UltraMacros sends its printed output. The default setting is <pr# Ø>, which sends the characters to the AppleWorks screen. Commands of <pr# 1>, <pr# 2>, and <pr# 3> send the output to the first, second, or third printer on your AppleWorks printer list. The macro in *Figure 14.10* sends the output to the first printer on your printer list instead of the AppleWorks screen.

However, you must be cautious about using <pr#>. First, remember that you need a space between the "pr#" and the number when you type the macro; the correct syntax is <pr# 1>, not <pr#1>.

Second, remember to issue a <pr# Ø> command when you are done sending information to the printer. UltraMacros continues to send text and commands to the specified location until you direct UltraMacros to send the output back to AppleWorks.

Although I now use TimeOut ReportWriter, I have written many macros that use the <pr#> command to generate sophisticated reports with headers and footers from the data base and spreadsheet modules.

Figure 14.11: Macros that Use <peek>

```
N:<all: x = peek $C54 :                    { Current open file }
        msg ' This is file # ' + str$ x + ' on the desktop '>!

G:<all: x = peek $C55 :                    { Number of files on the desktop }
        msg ' There are ' + str$ x + ' files on the desktop '>!

J:<all: x = peek $7B23 * 256 + peek $7B22: msg x >!
```

<peek> and <poke>

<peek> and <poke> are advanced programming commands that let you examine and change a value stored in your computer's memory. The <peek> command captures the value stored at a location you specify. For example, <z = peek $7DFØ> stores the current value from memory location $7DFØ into variable z. The <poke> command replaces a value currently in memory with a new value.

<peek> and <poke> are valuable to macro programmers because AppleWorks stores itself in memory during operation. You can learn about the internal workings of AppleWorks by examining what it stores in memory, and you can control the program by changing the code in memory.

Figure 14.11 presents several macros that use <peek> to get information from AppleWorks. The first macro (<sa-N>) displays the number on the Desktop Index associated with the current file. AppleWorks 2.x and 3.x stores this information in memory location $C54.

Macro <sa-G> displays the total number of files on the desktop. AppleWorks stores that number in location $C55.

<sa-J> displays the number of records in the current AppleWorks 2.1 data base file. This information consists of two

Figure 14.12: Macro that Inserts a Space

```
<spc>:<awp: x = peek $1ØF1   { Get current cursor type }
      insert :               { Go to insert mode }
      spc : left             { Do a space and move back }
      if x=1 then oa-e:      { If cursor was overstrike, issue an <oa-e> }
      :elseoff>!
```

Figure 14.13: Macro that Marks File "Unchanged"

```
C:<all: poke $C6C,Ø: msg ' Current file is now marked "Unchanged" '>!
```

Figure 14.14: Macro that Highlights First File on Remove Files List

```
T:<all: oa-Q: esc:              { Goto Main Menu }
        poke $C54,Ø :           { Clear current file number location }
        >4<rtn>!                { Choose "Remove Files" from Main Menu }
```

bytes stored in memory locations $7B22 and $7B23. <sa-J> shows you how to examine two-byte memory locations.

The macro in *Figure 14.12* demonstrates a practical application of the <peek> command. Macro <sa-spc> inserts a space in a document no matter which type of cursor is active. The macro uses <peek> to save the contents of memory location $1ØF1 in variable x ($1ØF1 holds the current cursor type; zero is the insert cursor, one is the overstrike cursor). The macro then forces the <insert> cursor mode and enters a space. If the cursor was in overstrike mode before you called the macro, it restores the cursor back to that mode.

The <poke> command inserts a value you specify into a given location. For example, <poke $C6C,Ø> pokes the value of zero into memory location $C6C. A zero in location $C6C indicates that the current file is "unchanged", so AppleWorks does not

Figure 14.15: Selected Memory Locations in AppleWorks 2.1

Memory Location	Contents of Location	Which Modules?
$ØC52	Data type on clipboard	all
$ØC54	Number of current desktop file	all
$ØC55	Number of files on desktop	all
$ØC6B	File type of current desktop file	all
$ØC6C	File is changed or unchanged	all
$ØE95	Table of numbered menu items (x,y,length)	all
$ØF19	Number of printers defined	all
$ØFD3-$ØFD4	K free on desktop	all
$1ØF1	Active cursor type	all
$1ØF2	Cursor visible flag	all
$1ØF5	Character under the cursor	all
$7B1B	active report number	adb
$7B2Ø	Data base zoom status	adb
$7B21	Number of categories in this record	adb
$7B22-$7B23	Number of records in this file	adb
$7B24	Number of reports in this file	adb
$7C61	Word processor zoom status	awp
$7D88	Window status	asp
$7DFØ	Spreadsheet zoom status	asp

warn you the file is changed when you quit the program. Macro <sa-C> in *Figure 14.13* shows how to use the <poke> command to insert the value zero in location $C6C. This macro changes the status of the current file to "Unchanged".

The <sa-T> macro in *Figure 14.14* demonstrates another possible use of the <poke> command. Macro <sa-T> takes you to the Remove Files Menu, and highlights the first file on the list of desktop files, not the current file.

Here is how <sa-T> works:

AppleWorks assigns a desktop index number to every file on the desktop and stores the number of the active file in memory

Figure 14.16: Selected Memory Locations in AppleWorks 3.0

Memory Location	Contents of Location	Which Modules?
$ØC52	Data type on clipboard	all
$ØC54	Number of current desktop file	all
$ØC55	Number of files on desktop	all
$ØC6B	File type of current desktop file	all
$ØC6C	File is changed or unchanged	all
$ØE95	Table of numbered menu items (x,y,length)	all
$ØF19	Number of printers defined	all
$ØFD3-$ØFD4	K free on desktop	all
$1ØF1	Active cursor type	all
$1ØF2	Cursor visible flag	all
$1ØF5	Character under the cursor	all
$851B	Active report number	adb
$852Ø	Data base zoom status	adb
$8521	Number of categories in this record	adb
$8522-$8523	Number of records in this file	adb
$8524	Number of reports in this file	adb
$7C61	Word processor zoom status	awp
$7F88	Window status	asp
$7FFØ	Spreadsheet zoom status	asp

location $C54. If you go to the Main Menu and say you want to "Delete files from the Desktop", AppleWorks highlights the file that was on your screen last. It remembers that choice by looking at the desktop index number stored in location $C54.

Macro <sa-T> issues an <oa-Q : esc> to go to the Main Menu and replaces the current desktop index number stored in location $C54 with a zero. Next, the macro selects choice #4 ("Remove files from the Desktop") from the Main Menu. A value of zero in location $C54 tells AppleWorks that there is no active file, so AppleWorks highlights the first file in the list.

Memory Locations

To use <peek> and <poke>, you need to know the memory locations used by AppleWorks. *Figure 14.15* presents a list of some of the locations for AppleWorks 2.1, *Figure 14.16* presents corresponding locations for AppleWorks 3.0. Adventurous readers can use <peek> to explore the default entries in these locations and <poke> to determine the impact of changing these values.

Remember that <poke> changes the values in memory, not the AppleWorks program on your disk. When things go awry, reboot your computer to load a fresh copy of AppleWorks back into memory.

Conclusion

UltraMacros offers a powerful programming language we can use to enhance AppleWorks. Like all programming languages, there is always more to learn about both the commands and the logic necessary to write useful programs. Given time, practice, and a good deal of patience, you should now be able to write almost any type of macro with UltraMacros.

15: New Features of 3.0 and Later

The latest version of UltraMacros includes more than twenty features not available earlier. This chapter describes those features. The commands described in this chapter only work with UltraMacros-enhanced copies of AppleWorks 3.x.

The most significant change in UltraMacros 3.x is the addition of 21 commands not available in earlier editions of the program. In this chapter, I describe how to use the most important of these new commands. Note that while UltraMacros 3.x is compatible with AppleWorks 2.0 and later, these new commands only work with AppleWorks 3.x.

<asr>

Application tokens such as <all>, <awp>, and <asp> specify where a macro works; <asr> ("A SubRoutine") indicates that the macro will only function as a subroutine. Therefore, macro <sa-R> in *Figure 15.1* will only run as a subroutine; nothing happens if you issue a <sa-R> from the keyboard.

Figure 15.1: Application of the <asr> Token

```
F:<all:sa-R>!
R:<asr:msg ' You cannot call this macro from the keyboard '>!
```

The <asr> token lets you write macros that users cannot access directly from the keyboard. This eliminates a concern of macro writers who often try to protect users from keystroke errors.

In addition, <asr> makes it easier to understand the logic of large macro sets. When you see "<ba-R: <asr:", you know the macro is a subroutine and that you should look for macros that call the routine.

Finally, <asr> gives you more flexibility when assigning keystroke labels to tokens. You no longer have to reserve a name when it appears as the name of a subroutine.

<cls>

<cls> clears the AppleWorks screen between the line of hyphens or tab ruler at the top of the display, and the line of hyphens at the bottom of the screen.

You should use <cls> before you use the new <msgxy> command that lets you print text in the AppleWorks work area.

Note that <cls> does not clear the character at the current cursor position; that character appears each time the cursor flashes. The trick is to issue an <ahead> command immediately before a <cls>. That puts the cursor on a blank space so nothing appears on the screen. The <ahead> command is not required in the spreadsheet module because the AppleWorks cursor is at the bottom of the screen.

<display>

<display> lets you blank the entire screen while running a macro. It eliminates the flashing that occurs when UltraMacros runs AppleWorks through a procedure.

You enter <display Ø> in a macro to turn off the screen, and <display 1> to resume the normal screen display. Remember to issue a <display 1> command to restore the screen or you will have to reboot your computer to see the display.

The <find> command works only when the screen is active; don't issue a <find> when <display Ø> is active.

<msgxy>

<msgxy> lets you display messages anywhere on the AppleWorks screen. The syntax is:

```
<msgxy x,y>
```

where x represents the column number on the screen and y represents the row where the message will start. For example,


```
<msgxy Ø,2 : msg "Hello there">
```

prints "Hello there" at the upper left-hand corner of the AppleWorks screen.

<msgxy> lets you address 80 columns across the screen (labelled positions Ø-79) and 24 rows (labelled Ø-23). <msgxy Ø,Ø> puts the cursor in the upper left-hand corner of the screen and <msgxy 79,23> puts the cursor in the lower right-hand corner.

Note the following if you use <msgxy>:

1. The command <msgxy Ø,128> resets the cursor position so UltraMacros prints future messages on the standard message line. Remember to issue a <msgxy Ø,128> after you use <msgxy> to relocate the cursor.
2. If you specify an x value of 255, UltraMacros will automatically center text horizontally on the AppleWorks screen. For example, <msgxy 255,1Ø : msg 'Hello'> will print "Hello" in inverse letters in the middle of the screen.

UltraMacros 3.x's ability to clear the AppleWorks screen and display messages using regular, inverse text, or Mousetext greatly enhances the display capabilities of the program. Macro <sa-L> in *Figure 15.2* demonstrates the sequence of commands you can use to control message displays with UltraMacros 3.x. This macro clears the AppleWorks screen, displays the message "Welcome" in an inverse box centered on the display and resets the cursor so future messages appear in their usual positions.

<endif>: <endif> is equivalent to <elseoff>; you can use either statement in your macros.

<first> and **<last>**: These cursor movement commands are equivalent to the <oa-,> and <oa-.> commands in AppleWorks 3.x.

Figure 15.2: Macro Showing the Messaging Capability of UltraMacros 3.x

```
L:<all:
     ahead : cls :          { Put cursor on a space and clear the screen }
     msgxy 255,9 :          { Position cursor  }
     msg '             ' : { Draw first line of blank inverse display }
     msgxy 255,1Ø :         { Move cursor }
     msg '  Welcome   ' :  { Print message }
     msgxy 255,11 :         { Move cursor }
     msg '             ' : { Draw another line of blank inverse display }
     msgxy 255,13 :         { Move cursor }
     msg "Press any key to continue" { Print message }
     x=key :                { Wait for keypress }
     msgxy Ø,128 :          { Reset cursor to standard message position }
     oa-Q : esc>!           { Return to Main Menu }
```

<first> moves the cursor to the beginning of the current line in a word processor document, to the first category of the current record in the data base module, and to column A in a spreadsheet. <last> moves the cursor to the end of the line in the word processor, to the end of the record in the data base, and to the last column in a spreadsheet.

<launch>: <launch> starts a task file from within a macro. This is equivalent to using the Macro Options to launch a new task file. <launch> lets you link task files, thus eliminating any problems caused by UltraMacros' somewhat smaller limit on the number of characters in a macro set.

The syntax of <launch> is as follows:

```
<launch 'string of text'> or <launch $1>
```

You can replace the reference to variable $1 with a reference to any string variable.

Figure 15.3: Examples Using Launch

```
1:<all: oa-Q : esc : rtn : rtn :  { Go to the Add Files Menu }
        $Ø = "Checkbook" :        { Set up for the Find Command }
        find : rtn :              { Find Checkbook file and load it }
        launch "check.macros">!   { Launch the task file for Checkbook }

2:<all: oa-Q : esc : rtn : rtn :  { Go to the Add Files Menu }
        $Ø = "Gradebook" :        { Set up for the find command }
        find : rtn :              { Find Gradebook file and load it }
        launch "grade.macros">!   { Launch the task file for Gradebook }

3:<all: launch "ultra.system">!   { Restore default macro set }
```

The macros in *Figure 15.3* demonstrate how to use <launch> to link AppleWorks spreadsheet files to their associated UltraMacros task files.

For these examples, imagine you are a teacher with two spreadsheet files; one maintains your home checkbook, the other is a gradebook that keeps track of student grades. You have separate task files for each application and you want to load the appropriate set of macros with each spreadsheet.

The first two macros in *Figure 15.3* will load the AppleWorks file and launch the associated task file with a simple keystroke.

The third macro lets you return to your original macro set with one keystroke. Note that macro <sa-3> launches "ultra.system", not "default.macros". Version 3.x of UltraMacros restores the default macro set when you launch "ultra.system"; you no longer need a separate task file to maintain a set of default macros. When you convert macros prepared for earlier versions of UltraMacros, you can now restore the default macro set with the command <launch "ultra.system">.

<mid>: The <mid> token lets you extract any part of a string; it is similar to the MID$ function in Applesoft BASIC. I described the syntax of <mid> in Chapter 10.

The <sa-K> macro in *Figure 15.4* demonstrates a practical application of the <mid> command. To understand this macro, you should imagine that you entered first and last names in a single category in a data base file. Now you want to divide the two portions of the name into separate first name and last name categories. You can use the <sa-K> macro in *Figure 15.4* as a subroutine to locate the space between the two names. <sa-K> stores the first name in variable $1 and the last name in variable $2. *[Ed: This parallels the function of subroutine macro <sa-J> in Chapter 13, Figure 13.4.]*

Figure 15.4: Sample Use of <mid> to Separate a Name

```
K:<asr :                  { This macro only works as a subroutine }
  X=1 :                   { X is current character position in the string }
  Y=len $Ø  :             { Y is total length of the string }
  begin:                  { Start repeating the loop from here }
     $1= mid $Ø,X,1 :     { Store the current character in $1 }
     if $1 = " " then     { If the character is a space... }
        Z=X-1 :           { Store the number of characters to the left of }
                          { the space in variable Z }
        $1=left $Ø,Z :    { Capture the string to left of the space }
        Z=X+1 :           { Z now contains the number of characters to the }
                          { beginning of the last name }
        $2=mid $Ø,Z,Y :   { Capture the string to the right of the space }
     else                 { If the character is not a space... }
        X=X+1:            { Go to the next character }
        if X<Y then :     { If the current position is less than total length }
rpt>!                     { loop back to the <begin> statement }
```

<exit> and <endmacro>

These commands help you control UltraMacros loops. <exit> tells UltraMacros to exit the current loop by skipping past the next <rpt> command. Use <exit> when you want to terminate a loop but stay within the current macro.

UltraMacros Primer: Errata

Please replace the two paragraphs that describe <keyto> starting at the bottom of page 187. Corrected text:

<keyto>: <keyto> is an alternative to the <input> command, but <keyto> lets you designate any key that tells the macro to continue. The syntax is <keyto n>, where "n" is a number that represents the ASCII value of a keystroke. <keyto> passes all user input directly to AppleWorks until the user presses the Escape Key or enters the keystroke with the ASCII value of "n". When it senses the specified keystroke, <keyto> tells the macro to continue. <keyto> sets variable Z to zero if the user presses the Escape Key or to the ASCII value of the keystroke that terminated the macro.

For example, entering a Return in response to an <input> command tells UltraMacros to stop accepting input and continue the macro. The <sa-L> macro in *Figure 15.6* uses the <keyto n> command to accept a Return in the string of text. This macro passes text onto AppleWorks until the user enters an <oa-Return>. (The value 141 is the high ASCII value of the Return Key, thus signifying an oa-Return.) *[Ed: A table of the ASCII value of all keystrokes appears in Appendix A.]*

the <keyto n> syntax, where "n" is a number that represents the ASCII value of a keystroke. Then <keyto> accepts input until you either press the Escape Key or the key with the ASCII value of "n". <keyto n> also sets variable Z to zero if the user presses the Escape Key or to the ASCII value of the keystroke that terminated the macro.

Normally, entering a Return in response to an <input> command tells UltraMacros to stop accepting input and continue the macro. The <sa-L> macro in *Figure 15.6* demonstrates how you can use the <keyto n> command to accept a Return in the string of text. This macro captures text until the user either presses the Escape Key or enters an <oa-Return>. The macro works because the value 141 is the high ASCII value of the Return Key, thus signifying an oa-Return (rather than a regular Return, which has an ASCII value of 13). *[Ed: A table of the ASCII value of all keystrokes appears in Appendix A.]*

Figure 15.6: <keyto n> Macro that Permits Entry of Returns

```
L:<awp:
    msg ' Press Escape to cancel ; oa-Rtn to continue ' :
    keyto 141 :                { Permit anything except Escape and oa-Return }
      if Z=Ø then              { If user pressed Escape... }
          oa-D:oa-left:rtn:    { Delete the current word }
      endif>!
```

<and> and <or>

<and> and <or> add and/or logic to the if-then-else-endif capability of UltraMacros.

The syntax for <or> commands is:

```
<if test1 or test2 or testn then true : else : all.false : endif>
```

where <test1> is the first test in the <if> statement, <test2> is the second test, and <testn> is the last test. <true> is what

occurs if any one or more of the tests is true. <all.false> is what occurs if all tests are false.

When you use <or>, UltraMacros executes the commands after <then> if any test is true. If all the tests are false, UltraMacros skips the commands after <then> and executes the commands after <else>.

<and> statements use a similar syntax. (See *Figure 15.7* for a sample <if> statement with <and> logic.)

You can use <and> or <or> by themselves or you can combine these statements in a single if-then-else statement. However, be careful when you get into complex and/or logic combinations. UltraMacros analyzes your statements from left to right and does not allow parentheses to control the flow of operations. For example, the macro

```
a:<all: if X=1 and Y>3 or $Ø="test" then sa-x : else : stop>!
```

calls macro <sa-x> if $Ø contains the characters "test" or if both X=1 and Y>3. Consider this logic and you will see the difficulty one incurs when combining <and> and <or> in a single statement.

One useful application of the <and> and <or> commands is to check whether a response is within a specified range. For example, imagine an accounting spreadsheet where you want users to enter a tax code between 10 and 99 in a cell.

The <sa-m> macro in *Figure 15.7* demonstrates how to limit the entries to valid codes. This macro accepts a tax code of up to three characters and converts those characters into a value. The <if> statement checks if the entry is between 10 and 99 and, if so, prints the tax code in the spreadsheet. If the entry is less than 10 or more than 99, <sa-m> clears the screen and displays an error message. When you press a key, the macro repositions the message cursor at the bottom of the screen and issues an <oa-q> followed by a <Return> to restore the spreadsheet display.

Figure 15.7: Using <and>/<or> to Validate Entries

```
M:<asp:
  msg ' Enter Tax Code Number ' :
  $Ø = getstr 3 :              { Get up to three characters }
  X  = val $Ø :                { Store the numeric value of $Ø }
   if X>9 and X<1ØØ then       { If entry is between 1Ø and 99... }
      print X : rtn :          { Put the entry in the cell }
         else :                { If entry is not between 1Ø and 99... }
            cls :              { Clear screen for a message }
            msgxy 255,7 :      { Move display cursor to middle of screen }
            msg ' Error .. tax codes must be between 1Ø-99; Press any key ' :
                               { Display message in inverse }
            Z=key :            { Pause for a keypress }
            msgxy Ø,128 :      { Reset cursor to standard message position }
            oa-Q : rtn :       { Restore view of the spreadsheet screen }
   rpt:                        { Allow a new entry }
   endif>!
```

<putvar> and <getvar>

UltraMacros 3.x uses <putvar> and <getvar> to overcome UltraMacros' limit of 26 numeric and 10 string variables in a single macro set. These commands let you store and recall up to eight different sets of variables. For example, <putvar 1> saves all the current variable settings into the space for variable set #1. <getvar 1> replaces the current variable settings with those in variable set #1.

The macros in *Figure 15.8* demonstrate an application of these commands.

Example of <putvar> and <getvar>

Imagine you use an accounting spreadsheet to track 18 different clients. You don't want to type each client's name when you record a transaction. The macros in *Figure 15.8* will automatically type the correct client's name when you enter two keystrokes.

Figure 15.8: Examples of <putvar> and <getvar>

```
I:<all:$1 = "Acme Fish Vendor" :
       $2 = "Betty's Bakeshop" :
       $3 = "Cori's Canners"   :
       $4 = "Dave's Diamonds"  :
       $5 = "Edward's Easter Eggs" :
       $6 = "Fritz's French Fries" :
       $7 = "Gary's Garage" :
       $8 = "Harry's Hardware House" :
       $9 = "Irwin's Ice Cream Shop" :
       putvar 1 :                        { Save as set 1 }
       $1 = "Jay's Jets" :
       $2 = "Karin's Card Shop" :
       $3 = "Larry's Letterhead" :
       $4 = "Mark's Macro Shop " :
       $5 = "Nina's Network" :
       $6 = "Oliver's Olive House" :
       $7 = "Pet Shop Boys" :
       $8 = "QuickPrint" :
       $9 = "Randy's Bug Exterminators" :
       putvar 2  >!                      { Save as set 2 }
<ba-1>:<all : getvar 1>!                 { Switch to first set of variables }
<ba-2>:<all : getvar 2>!                 { Switch to second set of variables }
1:<all:print $1>!                        { Print contents of variable $1 }
2:<all:print $2>!                        { Print contents of variable $2 }
3:<all:print $3>!
4:<all:print $4>!
5:<all:print $5>!
6:<all:print $6>!
7:<all:print $7>!
8:<all:print $8>!
9:<all:print $9>!
```

UltraMacros normally can manage a maximum of ten string variables in a macro set. However, the macros in *Figure 15.8* use the <putvar> command to establish two sets of string variables. Macros <ba-1> and <ba-2> use <getvar> to replace the current entries in each variable with the alternate set.

The <sa-I> macro stores "Acme Fish Vendor" in variable $1, "Betty's Bake Shop" in variable $2, and so on. Then <sa-I> issues a <putvar 1> command to store all nine entries in variable set #1. The macro then replaces the contents of variables $1 through $9 with new entries such as "Jay's Jets" and "Karin's Card Shop". The command <putvar 2> stores those nine entries in variable set #2. Then you can use the <ba-1> and <ba-2> macros to toggle between sets of variables. Once you choose the correct variable set, the command <sa-1> prints the contents of the first variable, <sa-2> prints the contents of the second variable, and so forth.

<peekword> and <pokeword>

<peekword> and <pokeword> are similar to the <peek> and <poke> commands I described in Chapter 14, with one exception: <peekword> and <pokeword> display or manipulate the contents of two adjacent memory locations. <peek> and <poke> only let you address a single memory location.

For example, AppleWorks always keeps a count of the total number of data base records in a file. Since that number can be greater than 255 (255 is the maximum number of different indicators you can store in a single byte), the program allocates two bytes of memory to store that number. You can check this value by using two <peek> commands as I did in macro <sa-J> in *Figure 14.11* in Chapter 14, or you can use <peekword> to check both locations in one operation.

AppleWorks 3.x stores this count in locations $8522 and $8523, so the correct command is <peekword $8522>. You specify the first of the locations, and the <peekword> command assumes you want the values in the two memory locations starting at the address you specify.

Figure 15.9: Example of <peekword>

```
b:<all x = peekword $FD3 : msg str$ x + 'K Avail'>!
```

<pokeword> works in the same fashion but inserts a value in two adjacent locations. For example, <pokeword $300,3000> puts the decimal value 3,000 in locations $300 and $301.

Macro <sa-b> in *Figure 15.9* uses <peekword> to read the desktop space available from locations $FD3 and $FD4 and displays that value on the screen.

Conclusion

UltraMacros 3.x represents the latest evolution of a product first announced in 1985 as MacroWorks. Version 3.x includes 21 new commands and modification to eight existing commands that add power and flexibility to UltraMacros' programming language

16: Introduction to Debugging

This chapter suggests techniques to help you find and correct errors in your macros.

As you proceeded through the exercises in this book, you probably made occasional errors when you typed a macro. Sometimes you found these errors as you proofread the macros. At other times you discovered your errors while compiling the macro.

Testing and debugging is an important step in the creation of any program, and compiled macros are no exception. The testing and debugging process grows in complexity and difficulty as you add variables and sophistication to your macros.

In this chapter, I will describe some techniques to help you find and correct errors in your macros. While I can suggest the procedures to follow, much of debugging is an art. You will learn the art form as you gain experience with the process.

Syntax and Logic Errors

In general, there are two steps to debugging any program. First, you check the syntax of the commands. Syntax errors in one or more commands halts operation of the UltraMacros Compiler and keeps a macro from compiling. Once the macro runs, you can start the more demanding task of identifying the errors in the logic and structure of the macro.

The primary tool to help you debug a macro is the UltraMacros Compiler. As you undoubtedly discovered, the compiler checks the syntax of every statement and tries to identify the nature and approximate location of any errors.

Syntax errors are generally of two types: typographical errors that are easy to fix, and incorrect syntax for a command.

Here are three of the most frequently occurring syntax problems:

1. Forgetting a colon after the text of a message. UltraMacros requires a colon after the closing single or double quotation at the end of a message. The correct syntax for message

commands is: <msg 'This is the message':stop> not <msg 'This is the message' stop>.

2. The incorrect number of parameters for a command. Most often this occurs with a command that requires three parameters such as the <screen> command.

3. The wrong type of parameter. Most frequently this involves specifying a numeric variable as a parameter when the command expects a string variable, or specifying a numeric value when the command expects a variable. For example, the <posn> command requires that you specify two numeric variables as parameters. The command <posn 1,1> yields a syntax error; the correct statement is <posn A,B>.

Appendix B of this book includes a summary of the correct syntax of each UltraMacros command and an example of the command. You can often save time by referring to either Appendix B or to the reference section of the UltraMacros manual when you cannot identify the cause of an UltraMacros Compiler syntax error.

Logic Errors

Identifying the source of errors in the logic of a macro is the biggest challenge in debugging a program. Logic errors can be classified into four types: Using the wrong operation, specifying an incorrect sequence of operations, incorrect branching, and storing or using the wrong variable.

All these errors can result in incorrect values in one or more UltraMacros variables, and the key to diagnosing the errors is to determine the value stored in each variable. You can then use those values to learn the source of the error.

There are at least two ways to determine the contents of each variable when a macro fails. The first is to use the TimeOut Debug module. The Debug module for UltraMacros 1.x and

Figure 16.1: AppleWorks 3.x Macro that Displays Contents of all Variables

```
<ba-%>:<all:oa-Q:cls:
            msgxy 1Ø,2:  msg "A = " + str$ A :
            msgxy 1Ø,3:  msg "B = " + str$ B :
            msgxy 1Ø,4:  msg "C = " + str$ C :
            msgxy 1Ø,5:  msg "D = " + str$ D :
            msgxy 1Ø,6:  msg "E = " + str$ E :
            msgxy 1Ø,7:  msg "F = " + str$ F :
            msgxy 1Ø,8:  msg "G = " + str$ G :
            msgxy 1Ø,9:  msg "H = " + str$ H :
            msgxy 1Ø,1Ø: msg "I = " + str$ I :
            msgxy 25,2 : msg "J = " + str$ J :
            msgxy 25,3 : msg "K = " + str$ K :
            msgxy 25,4 : msg "L = " + str$ L :
            msgxy 25,5 : msg "M = " + str$ M :
            msgxy 25,6 : msg "N = " + str$ N :
            msgxy 25,7 : msg "O = " + str$ O :
            msgxy 25,8 : msg "P = " + str$ P :
            msgxy 25,9 : msg "Q = " + str$ Q :
            msgxy 25,1Ø: msg "R = " + str$ R :
            msgxy 4Ø,2 : msg "S = " + str$ S :
            msgxy 4Ø,3 : msg "T = " + str$ T :
            msgxy 4Ø,4 : msg "U = " + str$ U :
            msgxy 4Ø,5 : msg "V = " + str$ V :
            msgxy 4Ø,6 : msg "W = " + str$ W :
            msgxy 4Ø,7 : msg "X = " + str$ X :
            msgxy 4Ø,8 : msg "Y = " + str$ Y :
            msgxy 4Ø,9 : msg "Z = " + str$ Z :
            msgxy Ø,12: msg "$Ø=" + $Ø :
            msgxy Ø,13: msg "$1=" + $1 :
            msgxy Ø,14: msg "$2=" + $2 :
            msgxy Ø,15: msg "$3=" + $3 :
            msgxy Ø,16: msg "$4=" + $4 :
            msgxy Ø,17: msg "$5=" + $5 :
            msgxy Ø,18: msg "$6=" + $6 :
            msgxy Ø,19: msg "$7=" + $7 :
            msgxy Ø,2Ø: msg "$8=" + $8 :
            msgxy Ø,21: msg "$9=" + $9 :
            stop>!            { note: press Escape to restore the screen }
```

Figure 16.2: Macro that Displays Contents of Variables for AppleWorks 2.x

```
<ba-&>:<all:oa-Q:esc:rtn>3<rtn:rtn>Variables<rtn:right:rtn:oa-9:
            print "A = " + str$ A : rtn :
            print "B = " + str$ B : rtn :
            print "C = " + str$ C : rtn :
            print "D = " + str$ D : rtn :
            print "E = " + str$ E : rtn :
            print "F = " + str$ F : rtn :
            print "G = " + str$ G : rtn :
            print "H = " + str$ H : rtn :
            print "I = " + str$ I : rtn :
            print "J = " + str$ J : rtn :
            print "K = " + str$ K : rtn :
            print "L = " + str$ L : rtn :
            print "M = " + str$ M : rtn :
            print "N = " + str$ N : rtn :
            print "O = " + str$ O : rtn :
            print "P = " + str$ P : rtn :
            print "Q = " + str$ Q : rtn :
            print "R = " + str$ R : rtn :
            print "S = " + str$ S : rtn :
            print "T = " + str$ T : rtn :
            print "U = " + str$ U : rtn :
            print "V = " + str$ V : rtn :
            print "W = " + str$ W : rtn :
            print "X = " + str$ X : rtn :
            print "Y = " + str$ Y : rtn :
            print "Z = " + str$ Z : rtn :
            print "$Ø=" + $Ø : rtn :
            print "$1=" + $1 : rtn :
            print "$2=" + $2 : rtn :
            print "$3=" + $3 : rtn :
            print "$4=" + $4 : rtn :
            print "$5=" + $5 : rtn :
            print "$6=" + $6 : rtn :
            print "$7=" + $7 : rtn :
            print "$8=" + $8 : rtn :
            print "$9=" + $9 : rtn :
                                    stop>!
```


2.x is on the TimeOut MacroTools disk. The module for UltraMacros 3.x is on the UltraMacros enhancement disk available from Beagle Bros.

Debug lets you display and change the contents of every variable. In addition, Debug displays additional information including whether there are any sleeping macros and which printer is active.

Another technique is to use the <stop> command to halt the operation of a macro and use <msg> commands to print out the contents of the variables when the macro stops. You can do this by inserting <stop> and <msg> commands in the macro. An alternative is to use the macros in *Figure 16.1* or *Figure 16.2* to stop your macro and display the contents of each variable. You can use either macro with AppleWorks 3.x, but the <ba-%> macro in *Figure 16.1* uses the <cls> and <msgxy> commands and only works with UltraMacros-enhanced copies of AppleWorks 3.x. Note that the <ba-%> macro in *Figure 16.2* creates a word processor file that contains a table of current values for the available macros.

You should copy either of these macros into the word processor file containing your macro source file and compile the macro with your regular macro set. Then you can insert a <ba-%> or <ba-&> command in your macro to call the appropriate debugging macro when you want to see the contents of the different variables. Either macro will display the contents of every variable and then stop.

Conclusion

Any programming activity is a mysterious blend of art and science. In this chapter, I described a series of steps and tools that can help you use "scientific" procedures to locate the source of your errors; but the art of debugging your macros remains up to you.

17: Advanced Programming

This chapter describes indirect string referencing, labels, and menus. These advanced UltraMacros programming concepts are important to developers of commercial-quality macro enhancements to AppleWorks.

One of the lessons learned by accomplished macro writers is that there are few limits to what you can do with UltraMacros-enhanced copies of AppleWorks. Many macro writers naturally progress from writing their own macros to writing macros for use by others.

Macros written for others must meet a higher standard of performance than macros you write for your personal use. They must be easy to understand and use, should present alternatives in an AppleWorks-like menu format, and must anticipate incorrect actions by the user. UltraMacros makes it practical for macro authors to present menus to users and to manipulate complex variables. This chapter describes how to use those advanced features.

Indirect String Referencing

Up to this point you used direct references to identify a string variable. For example, if you want to print the contents of $1, you issued the command <print $1>. UltraMacros also supports indirect string referencing; that is, you can define or call a string variable by using the syntax "$(N)" where N is a numeric variable containing a value of zero through nine. For example, the following two statements perform identical functions in UltraMacros:

Direct string reference: `<print $1>`

Indirect string reference: `<z = 1 : print $(z)>`

Indirect string referencing is a useful tool for macro programmers. They let you write shorter, more flexible, easier to understand macros.

Examples of indirect string referencing appear in *Figures 17.1* and *17.2*.

The macro in *Figure 17.1* uses indirect string referencing to print the contents of variables $1 through $7. The macro sets

Figure 17.1: Macro Using Indirect String Referencing

```
W:<all:
   x = 1 :
   begin :
   print $(x) : rtn :
   x = x + 1 :
   if x < 8 then rpt : endif>!
```

variable x equal to one, prints $1, increments variable x by one, tests to insure that x is less than 8, prints $2, and repeats the process until it prints all seven variables. Without indirect string referencing, this macro would require seven print statements.

The macros in *Figures 17.2A* and *17.2B* demonstrate another application of indirect string referencing. The <sa-a> macro in *Figure 17.2A* defines the contents of five string variables, asks the user to enter a number between one and five, and prints the corresponding string on the AppleWorks screen. Macro <sa-b> in *Figure 17.2B* uses indirect string referencing to simplify that task. While indirect string referencing is available in all versions of UltraMacros, these two macros use "or" statements in the line <if x < 1 or x > 5 then msg...>; thus, they will only run under AppleWorks 3.x.

Macro <sa-b> in *Figure 17.2B* stores the names of different TimeOut modules in strings $1-$5, uses the <msg> command to display some instructions, then uses the <getstr> command to capture one character, which should be a number. The <val> command then converts the character to a number (UltraMacros sets X equal to zero if the entry is anything besides the numbers one through nine). The <if> statement displays an error message and displays new instructions if the entry is outside the acceptable range of entries. If the entry is a number between one and five, the <print $(X)> command gets the value stored in variable X, prints the corresponding string, and issues a <rtn>.

Figure 17.2A: Macro that Prints One of Five Strings

```
A:<awp:
    $1 = " SuperFonts"                    { Initialize the string variables }
    $2 = " TeleComm  " :
    $3 = " Desktools " :
    $4 = " QuickWord " :
    $5 = " Thesaurus " :
    msg ' Enter product number ' :
    begin:                                { Return here if user enters wrong key }
      $Ø = getstr 1 :                     { Store the character entered by user }
      msg "" :                            { Blank the message area }
      x = val $Ø :                        { Convert the character to a number }
      if x < 1 or x > 5 then msg ' Enter a number between 1-5 ': rpt :
                                          { Error message if entry not 1-5 }
    else
    if x = 1 then print $1 : rtn : else { Print the chosen string }
    if x = 2 then print $2 : rtn : else
    if x = 3 then print $3 : rtn : else
    if x = 4 then print $4 : rtn : else
    if x = 5 then print $5 : rtn : endif>!
```

Figure 17.2B: Macro Using Indirect String References

```
B:<awp:
    $1 = " SuperFonts" :                  { Initialize the string variables }
    $2 = " TeleComm  " :
    $3 = " Desktools " :
    $4 = " QuickWord " :
    $5 = " Thesaurus " :
    msg ' Enter Product Number  ' :
    begin :                               { Return here if user enters wrong key }
      $Ø = getstr 1 :                     { Store the character entered by user }
      msg "" :                            { Blank the message area }
      x = val $Ø :                        { Convert the character to a value }
      if x < 1 or x > 5 then msg ' Enter a number between 1-5 ' : rpt :
                                          { Error message if entry not 1-5 }
    else
    print $(X) : rtn : endif>!            { Print the chosen string }
```

Ultramacros' support for indirect string referencing adds useful flexibility for UltraMacros programmers. Astute programmers can use this flexibility to write macros that are more efficient and easier to understand.

Using Labels

Chapters 15 and 16 described how to use <peek>, <poke>, <peekword>, and <pokeword> to examine or change the value in an AppleWorks memory location. These commands add significant power to the UltraMacros programming language.

UltraMacros 3.x makes it easier to use those commands by letting AppleWorks 3.x users specify "labels" instead of memory addresses in their macros. For example, UltraMacros 3.x lets you enter the command <peekword #dbrecs> to determine the number of records in a data base file; <peekword #dbrecs> is functionally equivalent to issuing the command <peekword $8522>.

The availability of labels makes it unnecessary for macro authors to remember specific memory locations; once you know the labels, you can write macros that are easier to understand and edit.

More importantly, using labels should increase the future "portability" of your macros. Future UltraMacros compilers should be able to convert these labels to the correct addresses for any new versions of AppleWorks.

Syntax and Examples Using Labels

The syntax used with labels is relatively simple. You write the command, a space, then a pound sign (#) followed immediately by the label, and another space or a ">" symbol. For example:

<x = peekword #freemem> sets variable X equal to the amount of available desktop memory and is equivalent to issuing the command <x = $ØFD3> in UltraMacros 3.x.

The command <poke #msgh, 255> centers future messages horizontally on the line until the user commands otherwise.

UltraMacros 3.x's ability to accept labels comes from an enhancement in its Macro Compiler. The new Compiler converts the label name into its appropriate value and stores the representation of that value in memory. However, once compiled, the Macro Compiler can reconstruct only the memory addresses, not the original labels you specified in the source file. Thus, you should save the AppleWorks word processor documents containing the original source files. Once you compile the macro, the Macro Compiler cannot reconstruct the original labels.

Figure 17.3 lists the memory labels for UltraMacros 3.x. *Figure 17.4* lists additional labels that let you control or determine the values used by UltraMacros, not by AppleWorks. For example, <poke #msgh, 10> tells UltraMacros to display all future messages at the eleventh position from the left hand edge of the screen (<poke #msgh, Ø> is the default setting). <poke #msgv, Ø> tells UltraMacros to display all future messages on the top line of the screen.

IIGS Keypad Feature

The keypad label in *Figure 17.4* requires additional explanation.

With UltraMacros 3.x, IIGS owners can run some Solid-Apple macros by using the numeric keypad without the Apple Keys. A table in UltraMacros stores a value for each numeric keypad key. You can change the entry in this table so the key no longer causes a normal keystroke but, instead, will call a macro. *Figure 17.5* presents the UltraMacros keypad table.

Figure 17.3: UltraMacros 3.x Labels

Label	What It Means
curhor	Horizontal cursor position.
curschar	Character under the cursor.
curver	Vertical cursor position.
dbfields	Number of categories in a data base.
dbrecs	Number of records in a data base.
dbrpts	Number of reports in a data base.
dbrules	Are selection rules active (Ø = no, 1 = yes)?
dbselrecs	Number of active records (Find or Selection Rules changes the count).
dbzoom	Data base zoom status (47 = multiple record layout; 82 = single record layout).
dirsort	Sorted Add Files directory ($2Ø = sorted directories; $2C = unsorted directories).
filecount	Number of files on the desktop.
filestatus	Is current file saved, changed, or unchanged? (See the Macro Special file on the UltraMacros disk to determine file status value.)
freemem	Available desktop memory in K.
kbtype	Type of data on clipboard (Ø = clipboard empty; 84 = word processor; 76 = data base; 83 = spreadsheet).
openfile	Current file's number in Desktop Index.
socursor	1 = strikeover cursor, Ø = insert cursor.
worktype	Contents of current spreadsheet cell (Ø = blank, 1-127 = label, >127 = value).
workval	Number of characters in a label in the current spreadsheet cell. (Only works if current cell is a label.)
wpwa	Length of current word processor line. (<127 = No return at end of this line. Value is number of characters in the line. >127 = Return at the end of this line. Number of characters is (wpwa - 127)).
wpzoom	Word processor zoom status (Ø = show commands; 1 = delete commands).

Here is how to use the table: Imagine that you want to enable the "1" key on the keypad to function as if it were a Solid-Apple-1. The "1" key is associated with the "offset" of 8, so you would issue the command <poke #keypad +8,Ø> in your macro. That says, "Replace the value of $B1 that is usually associated with the "1" key on the keypad with the value of zero." The "1" key on the keypad will then duplicate the function of the Solid-Apple-1 key combination. You must later issue the command <poke #keypad +8,$B1> to restore that key to its normal function.

Figure 17.4: Special UltraMacros Labels

Label	What It Means
curhor	Horizontal cursor position.
keypad	Exclusion table for auto keypad macros.
varset	Active variable set. (Lets you determine which variable set is active. Use <getvar> to change the active set.)
msgh	msg horizontal value.
msgv	msg vertical line. (default = 128 which puts message on the dashed line at the bottom of the screen.)

Figure 17.5: IIGS Keypad Values

Offset:	Ø	1	2	3	4	5	6	7	8	9	1Ø	11	12	13	14	15	16	17
Key:	Enter*	Clear**	*	+	-	.	/	Ø	1	2	3	4	5	6	7	8	9	=
Hex:	$8D	$98	$AA	$AB	$AD	$AE	$AF	$BØ	$B1	$B2	$B3	$B4	$B5	$B6	$B7	$B8	$B9	$BD

* Substitutes for <sa-Rtn>

**Substitutes for <sa-ctrl-X

Labels Instead of Characters

UltraMacros 3.x also lets you substitute labels for the ASCII value of a character. Specifically, UltraMacros 3.x lets you use labels in these formats: #'A', #"A", or #^A^ where the letter A represents any character.

A character in single quotes (') indicates the value of that true character, thus #'A' represents a value of 65 (the ASCII value of the character A).

A character in double quotes (") indicates the value of an

Open-Apple command with that character. Thus #"A" represent a value of 193 (the high ASCII value of character A).

A character in carets (^) indicates the value of a control character. Thus, #^A^ is the same as the value 1 (the ASCII value of Control-A).

For example, instead of writing:

```
<if x = 193 then goto sa-A : else :
if x = 212 then goto sa-T : endif>
```

UltraMacros 3.x lets you write:

```
<if x = #"A" then goto sa-A : else :
if x = #"T" then goto sa-T : endif>
```

This feature makes it easier to test commands like the Key Command, and it also makes it easier to read and understand a macro.

On/Off and True/False Labels

UltraMacros 3.x also supports four special value labels:

```
#on  = 1      #true = 1
#off = Ø      #false = Ø
```

These labels make it unnecessary to remember the specific value associated with a certain condition and make it easier to understand your macros. You can substitute the label #on or #true whenever the modifier to a command is a one. Alternatively, you can substitute the label #off or #false whenever the modifier is a zero. Here are three examples:

```
<display #on>

<display #off>

<if x=#true then y=#false : endif>
```

As with all labels, the UltraMacros Macro Compiler can recognize these labels, but cannot reconstruct the labels if you use

the Compiler to recreate your original macro source file. Instead, the Compiler inserts the values of zero or one.

To summarize: UltraMacros 3.x supports four types of labels:

1. Labels that let you address specific memory locations with <peek>, <poke>, <peekword>, or <pokeword>.
2. Labels that let you set values used by UltraMacros.
3. Labels that replace an ASCII character value in a macro.
4. Labels that represent values of zero and one as modifiers in UltraMacros statements.

UltraMacros 3.x offers a fifth type of label; labels that work with the <menu> command. I will describe those labels later in this chapter.

Preparing Menus

UltraMacros 3.x also offers a new <menu> command, a feature that makes it easier to build AppleWorks-like menus which prompt users for input. The <menu> command automatically builds a numbered on-screen menu, lets the user move the highlight with the arrow or number keys, and captures the user's selection in variable Z. The command also lets you identify other user responses such as the press of an Apple-Q or Escape instead of one of the expected menu choices.

The syntax for the <menu> command is:

`<menu $n>` where "n"stands for any number between zero and nine, or

`<menu "string1" : menu "string2" : menu "stringn">` where you substitute any string of text for "string1", "string2", and "stringn".

You must either separate each menu command with brackets (e.g., <menu "string1"><menu "string2">), or include a colon after each string (e.g., <menu "string1" : menu "string2">).

Note that you cannot mix other UltraMacros commands within a series of <menu> commands. UltraMacros views any token other than <menu> as a signal to activate the menu.

Here is a portion of a simple macro that generates a typical menu:

```
Z = 5:                              { start display on line 5 }
menu "Add files to the Desktop" :
menu "Work with files on the Desktop" :
menu "Save Desktop files to disk" :
menu "Remove files from the Desktop" :
menu "Other Activities" :
menu "Quit" :
```

Labels that Work with Menus

UltraMacros 3.x also offers six labels you can use to enhance your control over menus. To change the UltraMacros defaults, you <poke> values into these locations before issuing the first <menu> command.

The six special <menu> labels are as follows:

menufirst: Determines which menu item is highlighted. Remember to issue a <poke menufirst, 1> command after the series of <menu> commands to restore the correct default value.

Syntax: <poke #menufirst, 3> to automatically highlight the third item on the menu (default : menufirst = 1).

menuhelp: Lets you build a help screen the user can access with an OA-? keystroke. (You build the actual help screen using the <msgxy> and <msg> commands.) See *Figure 17.6* for an example of a macro that uses the menuhelp label. Also see

the file "Menu Sample" in the FILES.FOR.V3.0 subdirectory on the UltraMacros disk for an example of a macro that builds a help screen for a menu.

Syntax: <poke #menuhelp, 1> or <poke #menuhelp, #true> means menu help is enabled. <poke #menuhelp, Ø> or <menuhelp, #false> means menu help is not enabled.

menuhor: Sets the horizontal position for all following UltraMacros menus on the screen. The new menuhor value stays in effect during the rest of the AppleWorks session, therefore you should include a <poke #menuhor> statement to reset menuhor before each menu. That insures the menu will appear at the appropriate location on the screen. Otherwise, the previous menuhor setting will be in effect.

Syntax: <poke #menuhor, 15> means start the text of the menu items in column 16 (default : menuhor = 26).

menuinc: Sets the spacing between menu items. Remember to issue the command <poke #menuinc, 2> if you want to reset the double spacing default value.

Syntax: <poke #menuinc, n> where n = 1, 2, or 3 to indicate single, double, or triple spacing (default : menuinc = 2).

exitflag: AppleWorks treats oa-q, oa-s, and oa-ctrl-s keypresses differently from all other keystrokes. By referring to exitflag you can check to determine if the user pressed an oa-q, oa-s, or an oa-ctrl-s so your macro can respond to those keystrokes. See *Figure 17.6* for a macro that uses the exitflag label.

Syntax: <n = peek #exitflag> where "n" is any numeric variable. This sets variable n to the ASCII value Ø, 147 (oa-s), 209 (oa-q), or 211 (oa-ctrl-s).

key: Lets you check if the user entered a Return, an Escape, or an oa-? in response to a menu. The macro in *Figure 17.6* demonstrates an application of key.

Syntax: <n = peek #key> where "n" is any numeric variable. The value of n is zero unless the user exits the menu by pressing a Return (which sets n=13), an Escape (sets n=27), or an oa-? (sets n=191).

Putting It All Together with Menus

Building a menu for a macro is a fairly complex process. A good way to understand the different elements necessary to generate menus is to examine the sample macro in *Figure 17.6*. That macro lets users create a new word processor file, add files to the desktop from a disk, or change the active disk drive.

1. The <cls> command clears the screen.
2. <poke #menuhelp, #false> turns off the oa-? prompt; no help is available for this menu.
3. <poke #menuhor, 26>, declares that the menu items will start in column 27 (the first column is column Ø).
4. <Z=11> declares that the menu will start on the eleventh row on the screen.
5. <msgxy 255,4> resets the message cursor so the following message appears centered on the fifth line on the screen.
6. <msg "SUPER MENU"> displays the title of the menu in standard video.
7. <hilight 33,4,48,6> puts a rectangular box on the screen in inverse. The syntax for the highlight command is `<hilight a,b,c,d>` where "a" is the number of the left-hand-most column where you want the highlight to start, "b" is the top row of the highlight, "c" is the right-hand-most column where you want the highlight to end, and "d" is the bottom row you want highlighted. Thus, <hilight 33,4,48,6> says "Highlight a rectangle starting in the thirty-third column and fourth row through the forty-eighth column, sixth row."

Figure 17.6: Macro Demonstrating Menu Functions

```
m:<all: cls :                                   { Clear the screen }
  poke #menuhelp, #false :                      { Turn off the Apple-? "help }
                                                { available" display }
  poke #menuhor, 26 :                           { Define the starting column for these }
                                                { menu items }
  z = 11 :                                      { Line on screen where menu will start }
  msgxy 255,4 :                                 { Center the following message on line 4 }
  msg "SUPER MENU" :                            { Menu title }
  hilight 33,4,48,6 :                           { Splash a little inverse }
  msgxy Ø,128 :                                 { Reset the message cursor to normal }
  menu "Create a new file" :                    { Menu items }
  menu "Add files from disk" :
  menu "Change current disk" :
  q = peek #exitflag :
                                                { Check if the user entered an oa-q, }
                                                { oa-s, or oa-ctrl-s }
  if q = #"Q" then : oa-q : exit : else :       { Exit if user entered oa-q }
  ifnot q = Ø then rpt : endif :                { Re-display menu if user entered }
                                                { Escape, oa-s, or oa-ctrl-s }
  k = peek #key :                               { Check if user entered a Return, }
                                                { Escape, or oa-? }
  if k = 191 then msg ' No help available. Press any key ' :
                                                { Display message if oa-? }
     x = key : msg " " : rpt : endif :          { Continue the macro after response }
  oa-q :                                        { Reset the screen }
  poke #msgh,255 :                              { Center the next message }
  if z = Ø then oa-q : msg ' You pressed Escape ' : stop : else :
  if z = 1 then oa-q : esc : rtn>3<rtn : rtn>Newfile<rtn : else :
  if z = 2 then oa-Q : esc : rtn : rtn : else :
  if z = 3 then oa-Q : esc : up : up : rtn : rtn : endif :
    msg ' You picked item ' + str$ z + ' ' :
    poke #msgh,Ø>!                              { restore normal messages}
```

8. <msgxy Ø,128> resets the message cursor so future messages will appear in the correct location at the bottom of the screen.
9. The three <menu> statements contain the three lines of the menu. UltraMacros will automatically number these statements 1, 2, and 3.

10. <q = peek #exitflag> sets variable q equal to 147 if the user enters an oa-s, to 209 if the user enters an oa-q, or to 211 if the user enters an oa-ctrl-s. If the user enters any other keystroke to leave the menu, <q = peek #exitflag> sets variable q equal to zero.

11. <if q = #"Q" then exit : else> checks if the user enters an oa-q. If the user presses oa-q, the macro executes an oa-q, then stops and returns control to AppleWorks.

12. <ifnot q = Ø then rpt : endif> tells the macro to re-display the menu if the user enters an oa-s (which sets q to147) or oa-ctrl-s (sets q to 211). Thus, if the user enters an oa-s or oa-ctrl-s, nothing appears to happen. If the user enters an Escape, Return, or an oa-?, the macro continues.

13. <k = peek #key> checks to determine if the user exits the macro by typing a Return, an Escape, or an oa-?. If the user presses Return, the statement sets variable k equal to 13, the ASCII value of the return. If the user presses Escape, k equals 27. If the user enters an oa-?, k equals 191.

14. <if k = 191 then msg ' No help available. Press any key ' : x = key : rpt> checks if the user entered an oa-?. If the user did, the statement displays the message "No help available. Press any key" in inverse letters at the bottom of the screen. It awaits any keypress, and re-displays the menu.

15. <oa-q> resets the screen display.

16. <poke #msgh, 255> centers the next message on the screen.

17. UltraMacros stores the user's response in variable z, so z now contains a value of Ø, 1, 2, or 3 representing either a press of the Escape Key or one of the three menu choices on the screen. The statement <if z = Ø then oa-q : msg ' You pressed Escape ' : stop : else> checks if the user exited by pressing the Escape Key (which leaves z equal to zero). If z equals zero, this statement resets the screen with the oa-

q command, displays the message "You pressed Escape" in inverse letters at the bottom of the screen, and returns control to AppleWorks.

18. The statement <if z = 1 then oa-q : esc : rtn>3<rtn : rtn>Newfile<rtn : else> checks if the user selected menu option #1 (z = 1 if the user selected choice #1). This series of commands uses oa-q to reset the AppleWorks screen and then creates a new word processor file called Newfile.

19. The statement <if z = 2 then oa-q : esc : rtn : rtn : else> returns the user to the Add Files Menu and brings the catalog of files from the disk onto the screen if the user selects menu choice #2.

20. The statement <if z = 3 then oa-q : esc : up : up : rtn : rtn : endif> brings the user to the Change Current Disk Menu.

21. <msg ' You picked item ' + str$ z + ' '> confirms the user's choice at the bottom of the screen in inverse letters.

22. <poke #msgh, Ø> resets the message cursor back to its normal position so future messages get placed correctly on the screen.

Conclusion

This chapter describes how to use some of the advanced features available in UltraMacros 3.x. These include how to use indirect string references to make your macros more efficient and labels to make your macros easier to write, more easily understood, and more transportable to future versions of AppleWorks. Finally, you learned how to present menus to users in the middle of a macro.

Appendix A

ASCII Values for Keystroke Combinations

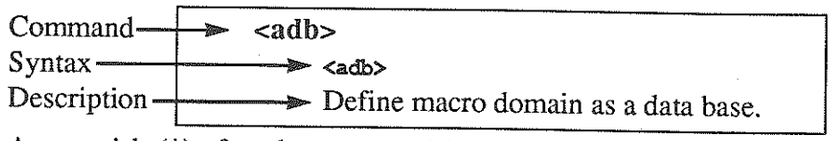
This table presents the ASCII value of all 256 key combinations recognized by UltraMacros. The first number in every row is the ASCII value of the key combination listed. For example, the lowercase letter "a" has a value of 97. The number in parenthesis is the ASCII value if the Open Apple Key is also pressed. For example, Open-Apple-a returns the value 225. The Return Key, Tab Key, and several other keys generate the same value as certain Control characters. For example, the chart indicates that pressing Control-[is equivalent to pressing the Escape Key.

0	(128)	ctrl-@		35	(163)	#	70	(198)	F	105	(233)	i
1	(129)	ctrl-A		36	(164)	$	71	(199)	G	106	(234)	j
2	(130)	ctrl-B		37	(165)	%	72	(200)	H	107	(235)	k
3	(131)	ctrl-C		38	(166)	&	73	(201)	I	108	(236)	l
4	(132)	ctrl-D		39	(167)	'	74	(202)	J	109	(237)	m
5	(133)	ctrl-E		40	(168)	(	75	(203)	K	110	(238)	n
6	(134)	ctrl-F		41	(169)	)	76	(204)	L	111	(239)	o
7	(135)	ctrl-G		42	(170)	*	77	(205)	M	112	(240)	p
8	(136)	ctrl-H	(Left)	43	(171)	+	78	(206)	N	113	(241)	q
9	(137)	ctrl-I	(Tab)	44	(172)	,	79	(207)	O	114	(242)	r
10	(138)	ctrl-J	(Down)	45	(173)	-	80	(208)	P	115	(243)	s
11	(139)	ctrl-K	(Up)	46	(174)	0	81	(209)	Q	116	(244)	t
12	(140)	ctrl-L		47	(175)	/	82	(210)	R	117	(245)	u
13	(141)	ctrl-M	(Rtn)	48	(176)	0	83	(211)	S	118	(246)	v
14	(142)	ctrl-N		49	(177)	1	84	(212)	T	119	(247)	w
15	(143)	ctrl-O		50	(178)	2	85	(213)	U	120	(248)	x
16	(144)	ctrl-P		51	(179)	3	86	(214)	V	121	(249)	y
17	(145)	ctrl-Q		52	(180)	4	87	(215)	W	122	(250)	z
18	(146)	ctrl-R		53	(181)	5	88	(216)	X	123	(251)	{
19	(147)	ctrl-S		54	(182)	6	89	(217)	Y	124	(252)	|
20	(148)	ctrl-T		55	(183)	7	90	(218)	Z	125	(253)	}
21	(149)	ctrl-U	(Right)	56	(184)	8	91	(219)	[	126	(254)	~
22	(150)	ctrl-V		57	(185)	9	92	(220)	\	127	(255)	Delete
23	(151)	ctrl-W		58	(186)	:	93	(221)	]			
24	(152)	ctrl-X	(Clr)	59	(187)	;	94	(222)	^			
25	(153)	ctrl-Y		60	(188)	<	95	(223)	_			
26	(154)	ctrl-Z		61	(189)	=	96	(224)	`			
27	(155)	ctrl-[	(Escape)	62	(190)	>	97	(225)	a			
28	(156)	ctrl-\		63	(191)	?	98	(226)	b			
29	(157)	ctrl-]		64	(192)	@	99	(227)	c			
30	(158)	ctrl-^		65	(193)	A	100	(228)	d			
31	(159)	ctrl-_		66	(194)	B	101	(229)	e			
32	(160)	Space		67	(195)	C	102	(230)	f			
33	(161)	!		68	(196)	D	103	(231)	g			
34	(162)	"		69	(197)	E	104	(232)	h			

Appendix B:

UltraMacros Token List

Below is a list of the UltraMacros commands described in this book. The structure of each entry is as follows:



An asterisk (*) after the command indicates it works only with AppleWorks 3.0 or later. Two atersisks (**) indicate the command only works with AppleWorks 2.x.

<back> 163
```
<back>
```
Find first blank space to left of cursor.

<begin> 119
```
<begin>
```
Start of loop.

<bell> 67
```
<bell:bell>
```
Sound the AppleWorks buzzer twice.

<cell> 93
```
<$2=cell>
```
Read the contents of the current data base entry, spreadsheet cell, or word processor line into variable $2.

<chr$> 171
```
<$5=chr$27>
```
Store a press of the Escape Key into variable $5.

<clear> 171
```
<clear>
```
Clear all numeric and string variables.

<cls>* 182
```
<cls>
```
Clear the screen between the line of hyphens or the tab ruler at the top and the line of hyphens at the bottom.

<ctrl-@> 29
```
<ctrl-@>
```
Stop recording a macro.

<ctrl-x> 29
```
<ctrl-x>
```
Forces UltraMacros to display the cursor during operation of a macro.

<date> 100
```
<date>
```
Send today's date to AppleWorks in the format January 1, 1990.

```
<$2 = date>
```
Store today's date in the format January 1, 1990 in variable $2.

<date2> 114
```
<date2>
```
Send today's date to AppleWorks in the format 01/01/90.

```
<$6 = date2>
```
Store today's date in the format 01/01/90 in variable $6.

<dec> 162
```
<dec>
```
Decrement character at the current cursor position.

<disk> 167
```
<disk: msg $Ø>
```
Store the disk name in $Ø and display the disk name.

<display>* 182
```
<display Ø>
```
Turn off the screen display.

```
<display 1>
```
Resume normal screen display.

<else> 143
```
<if A=B then msg 'yes' : else msg 'no' : elseoff>
```
If A equals B, display "yes". If A does not equal B, display "no".

<elseoff> 79
```
<if $4="Forum" then bell : elseoff>
```
Signals the end of an <if> expression. Functionally equivalent to <endif>.

<endif>* 79
```
<if $1 = "N" then bell : endif>
```
Signals the end of an <if> expression. Functionally equivalent to <elseoff>.

<endmacro>* 186
```
<endmacro>
```
Exit the current macro and proceed as if the macro were completed.

<exit>* 186
```
<exit>
```
Forces macro to skip past the next <rpt> command.

<find> 97
`<oa-q : $Ø="Sales.June" : find>`
Moves highlight cursor to the file "Sales.June" in the Desktop Index. In AppleWorks 3.x, a successful <find> sets Z=1; and unsuccessful <find> sets Z=Ø. Also forces the cursor to the next carriage return marker in the word processor.

<findpo> 163
`<findpo>`
Moves cursor to the next caret mark in the word processor.

<first>* 183
`<first>`
Puts the cursor at the beginning of the current line, on the first category of the current record, or in column A in a spreadsheet.

<getstr> 78
`<$4=getstr 5>`
Stores user entry (up to 5 characters) in $4.

<getvar>* 190
`<getvar 1>`
Replaces the current variable settings with the settings in variable set #1.

<goto> 81
`<goto ba-1>`
Jump immediately to macro Both-Apple-1.

<id#> 168
`<A=id#>`
Sets variable A equal to the ID number of the active TimeOut module. If no module is active, sets A=Ø.

<if> 79
`<if $4="Forum" then bell : endif>`
Sounds AppleWorks buzzer if statement is true.

<ifnot> 145
`<ifnot A=B : bell : elseoff>`
Sounds bell if A does not equal B.

<inc> 162
`<inc>`
Increments the character at the current cursor position.

<input> 70
`<oa-F : rtn : input : rtn>`
Lets the user enter text to find. User presses the Return Key to signify "end of input".

<insert> 161
`<insert>`
Turns on the insert cursor.

<key> 70
`<key>**`
Pauses macro execution until a key is pressed.

`<x=key>`
Stores the ASCII value of the next keypress into numeric variable x.

<keyto>* 187
`<keyto n>`
Accepts input until either the escape key or the key with the ASCII value n is pressed. Sets z=Ø if user pressed Escape; z=n if input terminates with keystroke n.

<last>* 183
`<last>`
Puts the cursor at the end of the current line, at the end of the record, or in the last column.

<launch>* 184
`<launch "check.macros">`
Launch the task file "check.macros".

<lc> 162
`<lc>`
Changes character under the cursor to lower case.

<left> 111
`<$3 = left $1,5>`
Stores the first five characters from variable $1 in variable $3.

<len> 107
```
<Q = len $2>
```
Puts a count of the number of characters stored in variable $2 into numeric variable Q.

<mid>* 111, 186
```
<$Ø = mid $1,5,10>
```
Extracts the first ten characters starting with the fifth character in variable $1 and stores the result in variable $Ø.

<msg> 68
```
<msg 'working'>
```
Displays the word "working" at the bottom of the screen in inverse.

```
<msg "working">
```
Displays "working" at the bottom of the screen in normal text.

<msgxy>* 182
```
<msgxy Ø,2 : msg "Hello there" : msgxy Ø,128>
```
Displays "Hello there" starting at column Ø, row 2 and resets message cursor to bottom of screen.

<nosleep> 165
```
<nosleep>
```
Cancels the currently defined sleeping macro.

<oa-x> 27
```
<oa-x>
```
With AppleWorks 2.x, starts recording a keyboard macro. Macro recording terminates with Control-@. With AppleWorks 3.x, <oa-x> toggles macro recording on and off.

<onerr> 123
```
<onerr goto sa-x>
```
Goes to macro Solid-Apple-x when AppleWorks normally beeps.

```
<onerr off>
```
Causes UltraMacros to return to its normal condition (ignoring all AppleWorks warning beeps).

```
<onerr stop>
```
Stops the current macro when AppleWorks normally beeps.

<or>* 188
```
<if A=B or A<B then msg ' okay ' : else : bell : endif>
```
If A=B or A<B, "okay" is printed, otherwise the bell sounds.

<path> 167
```
<all: path: msg $Ø>
```
Displays the full pathname of the current file.

<peek> 173
```
<Z = peek $7DFØ>
```
Stores the current value of memory location $7DFØ in variable Z.

<peekword>* 192
```
<X=peekword $8522>
```
Stores the current value of memory locations $8522 and $8523 in variable X.

<poke> 173
```
<poke $C6C,Ø>
```
Puts a value of zero into memory location $C6C.

<pokeword>* 192
```
<pokeword $3ØØ, Ø>
```
Puts a value of zero into memory locations $300 and $301.

<posn> 109
```
<posn A,B>
```
Word Processor: Stores the column position of the cursor in numeric variable A and the row position of the cursor in variable B.

Data Base: Stores the current category number in A and the record number in B.

Spreadsheet: Stores the current column number in A and row number in B.

<pr#> 172
```
<pr# 1>
```
Sends future output to the first printer on your printer list.

<print> 79
```
<print $1>
```
Sends the contents of $1 to AppleWorks as keystrokes.

<putvar>* 190
```
<putvar 1>
```
Saves the current variable settings into the space for variable set #1.

<read> 162
```
<read>
```
Stores the character at current cursor position into variable $Ø.

<recall> 166
```
<recall>
```
Puts data stored in the current file with <store> in variable $Ø.

<right> 111
```
<$5 = right $2,7>
```
Stores the last seven characters from variable $2 into variable $5.

<rpt> 119
```
<all: msg ' repeat ' : rpt>
```
Tells UltraMacros to re-execute the instructions in the macro.

<screen> 89
```
<$3=screen 1,20,5>
```
Reads five characters from the screen, beginning at first column of line 20 into location $3.

<store> 166
```
<$Ø=getstr 15: rtn: store>
```
Stores first 15 characters of variable $Ø in a special area within the current word processor, data base, or spreadsheet file. Stored text is displayed at bottom right corner of the screen.

<str$> 113
```
<$1 = str$ A>
```
Converts the number currently stored in variable A into a string of text and stores that string in variable $1.

<then> 140
```
<if A=B then msg ' yes ' : elseoff>
```
If A equals B then executes all steps between then and elseoff.

<time> 101
```
<time>
```
Sends the time to AppleWorks in the format 3:10 pm.

```
<$2 = time>
```
Stores the time in the format 3:10 pm in variable $2.

<time24> 114
```
<time24>
```
Sends the time to AppleWorks in the format 15:10.

```
<$2 = time24>
```
Stores the time in the format 15:10 pm in variable $2.

<uc> 162
```
<uc>
```
Changes character under the cursor to upper case.

<val> 108
```
<V = val $3>
```
If string variable $3 starts with numeric characters, stores the numeric equivalent of those characters in variable V.

<wake> 164
```
<wake sa-Q at Ø5:ØØ>
```
Starts macro Solid-Apple-Q at 5:00 a.m.

<zoom> 153
```
<zoom>
```
Forces zoom out display.

Index

The National AppleWorks Users Group

The National AppleWorks Users Group (**NAUG**) has one mission – to help AppleWorks users.

Formed in 1986, the National AppleWorks Users Group (**NAUG**) is the world's largest association of Apple II users. By early 1990, **NAUG** had grown to more than 14,000 members in the United States and 38 other countries.

Benefits

NAUG members receive the ***AppleWorks Forum***, a 36-page monthly newsletter that describes tips, techniques and hints to help users get more from AppleWorks. The ***AppleWorks Forum*** includes other articles of interest to AppleWorks users, including news of product releases and reviews of AppleWorks enhancements and AppleWorks-compatible software. **NAUG** works closely with Claris Corporation, Apple Computer, Applied Engineering, Beagle Bros, Checkmate Technology, JEM Software, and other AppleWorks developers to insure our members get the latest information about AppleWorks and easy access to help with their AppleWorks problems.

NAUG members have access to hundreds of volunteer-consultants who provide free telephone support for their fellow **NAUG** members. A list of consultants and their areas of expertise appears in each issue of the ***AppleWorks Forum***.

Members get unlimited access to **NAUG**'s AppleWorks electronic bulletin board, the Electronic Forum. The Electronic Forum lets you get answers to questions and download AppleWorks templates and programs 24-hours a day. As of this date, the Electronic Forum has received more than 29,000 calls from **NAUG** members seeking help or sharing information with their **NAUG** colleagues.

Members have access to **NAUG**'s Public Domain Library that contains dozens of disks and hundreds of AppleWorks templates and files. The library also contains AppleWorks enhancement programs submitted by members and commercial vendors.

NAUG members have access to the group's Disk Rescuers Program; professionals who can recover damaged AppleWorks data disks. You will appreciate this program if you ever forget to back up an important file and get the dreaded "Unable to read disk in Drive 2" message.

Finally, **NAUG** members receive significant discounts on AppleWorks products. For example, **NAUG** members can purchase TimeOut enhancements, RepairWorks, SuperPatch, and other AppleWorks-compatible programs at 40% off the suggested retail price. **NAUG** members also get significant discounts for **NAUG**'s AppleWorks seminars.

NAUG membership costs $29 per year and includes 12 issues of the ***AppleWorks Forum***. Send a check or your Visa/MasterCard number and expiration date to:

National AppleWorks Users Group
Box 87453
Canton, Michigan 48187
(313) 454-1115

The UltraMacros Primer:
How to Use TimeOut UltraMacros

About the Book:

TimeOut UltraMacros adds more than 60 easy-to-use commands to AppleWorks and gives AppleWorks the ability to memorize your keystrokes. This makes it possible to save keystrokes and automate repetitive procedures.

The UltraMacros Primer teaches you everything you need to know to use UltraMacros. Author Mark Munz describes how to install and use the program in the easy-to-understand, step-by-step fashion for which he is known. He illustrates his work with useful sample macros, and provides a tutorial for users of all levels.

About the Author:

Mark Munz is a noted author of Apple. Mr. Munz wrote Late Nite Patches, MacroToo 3.0 Companion Disk, TimeOut TextTools, a tutorial series on UltraMacros for the ***AppleWor***. ogram-mer for Beagle Bros, publishers of TimeOut UltraMacros, where he develops and tests software and writes user documentation. Mr. Munz is a consultant to the National AppleWorks Users Group.

Edited by: Warren Williams
William Marriott
Cathleen Merritt
Layout and design: Nanette Luoma

$19.95
ISBN 0-9620807-3-X