ToolKit

Pinpoint™

Inside Pinpoint

The Technical Reference Manual
for the Pinpoint Toolkit

by

Brian Skiba • Carlton Sue • Mark Wieder

Pinpoint Publishing
Emeryville, California

Limitation on Warranties and Liability

Even though Pinpoint has tested the software described in this manual and reviewed its contents, neither Pinpoint Publishing, Virtual Combinatics, Inc., nor its software suppliers make any warranty or representation, either express or implied, with respect to this manual or to the software described in it, its quality, performance, merchantability, or fitness for any particular purpose. As a result, this software and manual are sold "as is." The purchaser assumes the entire risk as to their quality and performance. In no event will Pinpoint, Virtual Combinatics, Inc., or its software suppliers be liable for direct, indirect, special, incidental, or consequential damages resulting from any defect in the software or manual, even if they have been advised of the possibility of such damages. In particular, they shall have no liability for any programs or data stored in or used with Pinpoint products, including the costs of recovering these programs or data. If you think you bought a defective copy of Pinpoint software, send the disk back to us and we'll send you a new disk, free. Apple Computer, Inc. makes no warranties, either express or implied, regarding the enclosed computer software package, its merchantability or its fitness for any particular purpose. The exclusion of implied warranties is not permitted by some states. The above exclusion may not apply to you. This warranty provides you with specific legal rights. There may be other rights that you may have which vary from state to state.

Copyright Notice

Published by Virtual Combinatics, Inc.
dba Pinpoint Publishing
5901 Christie Avenue • Emeryville, CA 94608
PO Box 13323 • Oakland, CA 94661 • (415) 654-3050
Telex 245-8579 • CompuServe 76244,123 • Delphi and MCI: PINPOINT

Pinpoint and Virtual Combinatics reserve the right to make improvements to this manual and software described in it at any time without notice. Pinpoint and Virtual Combinatics cannot guarantee that you will receive notice of a revision to the software described in this manual, even if you have received the product and returned a registration card received with the product. You should periodically check with your authorized Pinpoint dealer.

First Edition

Program Authors: Brian Skiba, Carlton Sue, and Mark Wieder
Documentation Authors: Brian Skiba and Gerry Villareal
Technical Editor: Mate Gross
Copy Editor and Desktop Publisher: Mary Sanichas

Trademark Acknowledgements

Table of Contents

Preface

Chapter 1: The Pinpoint Toolkit

Chapter 2: The Pinpoint Internal Architecture

Chapter 3: RunRun

Chapter 4: General ProDOS Tools

Chapter 5: Tools of the Trade

Chapter 6: Designer Tools

Chapter 7: Writing Desktop Accessories

Chapter 8: Advanced Installation

Appendix: Other Pinpoint Products

Preface

Isn't it nice to receive software on non-copy protected disks? We distribute all our software, including the Pinpoint Toolkit, on non-copy protected disks because we want our software to be convenient to use, and because we trust you. Please remember that giving a copy of our software to a friend is a violation of copyright laws. Illegal copying of our software also makes it harder for us to make a living selling the software you'll be using and depending on every time you use your computer.

We value your assessment of our software, and we want to include your ideas in our new product plans, so let's hear from you. Write to us at:

Pinpoint Publishing • PO Box 13323 • Oakland, CA 94661
Attention: Product Development

Or call our general offices: (415) 654-3050

24 hr. Toll Free Order Desk

If you would like to order another Pinpoint product, we have toll free sales lines. Just check the appendix in this manual for a list of Pinpoint products and call:

Sales: United States **(800) 633-2252 Ext. 582**

About Inside Pinpoint

Inside Pinpoint is designed for use mainly by Apple II programmers and analysts.

Chapter 1 provides an overview of the Pinpoint Toolkit, its components, and how it applies to each user.

Chapter 2 explains the internal architecture of the Pinpoint Desktop Accessory Manager, its various components and capabilities. The latter portion of this chapter details how you can access the Pinpoint dispatcher via Assembly language calls. Each dispatcher function is discussed in detail.

Chapter 3 introduces you to RunRun. This is an interactive shell to ProDOS that allows you to write desktop accessories or ProDOS applications.

Chapter 4 discusses several general purpose desktop accessories useful to all Toolkit users. These include the ProDOS Filer accessory, a screen printing accessory, a disk-resident version of the popular phone dialer. The source code to the phone dialer and the screen printing accessories are included with the Toolkit. The PrintScreen accessory is an excellent example of how you can use the printer and font library to perform graphic or text printing.

Chapter 5 discusses several desktop accessories designed especially for the programmer. These include the Accessory Mover, the Hex Calculator, and the Memory Window. The Accessory Mover allows you to define new desktop accessories to Pinpoint. It also allows you to move desktop accessories about. The Hex Calculator performs numeric and Boolean operations in decimal, hexadecimal, and binary modes.

Chapter 6 discusses two design tools for constructing Pinpoint accessories: the Video Resource Editor and the Resource Converter. The Video Resource Editor allows you to design your interface visually for a desktop accessory or primary application. You can build desktop accessories very quickly using the Resource Editor. The Resource Converter transposes the binary resource file to something more compatible in source code. It generates source code for a variety of compilers/assemblers.

Chapter 7 focuses on the process of building desktop accessories. This chapter instructs you on how to piece together the visual resources, dispatcher calls, and accessory-specific code. Chapter 7 also discusses the use of KYAN PASCAL,™ MICOL BASIC,™ and Assembly language in building desktop accessories.

Chapter 8 discusses the Advanced Installation Facility. This is a modified version of the program found with the Pinpoint Desktop Manager. It allows you to install the Pinpoint dispatcher on startup system files of your own.

Programming Conventions

Throughout this book programming examples are given to clarify the presentation of material. The examples were written in 65C02 assembler code and were developed using the ProDOS Tools™ from Apple Computer, Inc.. The syntax of the Apple assembler, EDASM,™ will be different than other assemblers used. A brief description of the syntax is included below.

The main mnemonic difference between EDASM and other assemblers is in the expression of the high-byte and low-byte operators. EDASM always maintains a 16-bit value when it evaluates expressions such as a label. The high-byte operator is expressed with the "<" symbol, and the low-byte operator with the ">" symbol.

For instance: >$EE22 is equivalent to $22
<$EE22 is equivalent to $EE

The location counter is expressed with an "*" (asterick symbol).

For further clarification, you may wish to refer to the book *ProDOS Assembler Tools*, Apple Computer, Inc. 1983.

Assumptions

This manual assumes that you have used the Pinpoint desktop accessories and that you have read the *Pinpoint AppleWorks Desktop Accessories User Guide* (ISBN:0-917413-00-8 Pinpoint Publishing, copyright 1985).

Product Support

The Pinpoint Toolkit is primarily for those who wish to build their own desktop accessories. Technical support for this product is not included with the purchase price of the product, but rather at an additional charge. Please refer to the Software Support Agreement included in this package for support details.

Chapter 1: The Pinpoint Toolkit

An Overview

Pinpoint was designed, from the beginning, to be a technical *foundation* to be built upon. Pinpoint is a technology, not a single, static product. The original desktop accessories that are included with the Pinpoint Desktop Manager serve as a few good, simple examples of what desktop accessories can be and do. They are not, by any stretch of the imagination, the "end-all" desktop accessories. Better ones will be built. The *important* aspect is the foundation — Pinpoint.

The Pinpoint Toolkit contains the hammers, saws, and nails — the tools needed to make the building. It contains sample blueprints, in the form of desktop accessories with source code. It does not contain the final blueprint, however. That is up to you.

The Pinpoint Toolkit will serve a variety of purposes for a heterogeneous users' base. On one end, the computer hacker can design and implement desktop accessories to complement the original Pinpoint desktop accessories. A third-party software vendor could make their products "Pinpoint Compatible." At the other extreme, users could use RunRun as a working shell to operate their applications and accessories. End users would find the ProDOS Filer accessory or the PrintScreen accessory very handy in their daily activities.

The Toolkit contains six basic components:

(a) Detailed technical documentation that provides tremendous insight into the inner workings and capabilities of Pinpoint technology.

(b) RunRun — A ProDOS shell environment that allows you to run desktop accessories within desktop accessories.

(c) Several key tools for constructing desktop accessories including the Video Resource Editor, Resource Converter, and Accessory Mover.

(d) Several sample accessories that highlight the use of the Pinpoint dispatcher, printer library, graphic library, and font library.

(e) Several general purpose desktop accessories that are handy for developers as well as end users, including a version of the ProDOS Filer running as a desktop accessory, screen printing facilities, etc.

(f) An advanced installation program that allows Pinpoint desktop accessories to be installed on personal programs as well as third party programs.

Building Desktop Accessories

Building desktop accessories is not a trivial process. This is largely due to several inherent problems with the architecture of the Apple II. First, the 65C02 microprocessor has a simple, efficient instruction set. The limited availability of hardware registers, coupled with addressing limitations, makes most compiled applications difficult to operate without large runtime libraries. A runtime library can be anywhere from 4 to 20 K in size. This is not bad, simply time consuming. Second, the ProDOS operating system was intended to handle a single task operating at a time. It provides very little memory management capability, and does not operate beyond the lower 64K of memory in the Apple II.

In many ways, the first problem is remedied by companies such as Applied Engineering in Carrolton, Texas. They produce memory expansion boards for the Apple II to boost the memory within the Apple. The concern over runtime libraries 20K in size is diffused. Large size desktop accessories can reside on a RAM drive and be loaded almost instantly.

The Pinpoint Desktop Manager helps with the second problem. Pinpoint manages the Apple II/ProDOS operations so as to provide a capability known as *limited multi-tasking*. This means that a primary task can be interrupted for an indefinite period of time by a secondary application, a desktop accessory. The state of the machine is preserved, so the primary task has little idea of what has occurred. The Pinpoint Desktop Manager can also provide a "messaging" capability between two co-resident applications such as AppleWorks™ and the Pinpoint Spelling Checker.™ In this case, both applications are active and communicating with one another via inter-application messaging.

Given the direction of the Apple II audience, towards larger memory and the use of RAM drives, writing desktop accessories has become more of a manageable task. The original Pinpoint desktop accessories required more than 56,000 lines of hand-optimized assembler code. If this had been written in a higher level language, such as PASCAL, this could have been reduced to about a third of that size.

Desktop accessories can be written is Assembly, PASCAL (using KYAN PASCAL), or compiled BASIC (using the MICOL BASIC compiler).

How this Manual Can Help

This manual, coupled with the programming examples included on the source disk, will help guide you towards creating desktop accessories. It explains the architecture of Pinpoint and how it can be used to create desktop accessories. It also explains how you can make ProDOS applications you have developed compatible with Pinpoint.

Getting the Toolkit Ready

The Pinpoint Toolkit is distributed on two *flippy* disks. Each disk contains two sides of information on it, the equivalent of 4 disks. The disks we provide you will not work because they are write-protected. You must use non-write-protected copies of the Pinpoint Toolkit. We are providing you write-protected disks to help you protect your investment.

So, before you do anything with the Pinpoint Toolkit, please use Central Point Software's Copy II Plus,™ the Apple IIc System Utilities,™ the Apple IIe ProDOS Filer,™ or any ProDOS copy routine to make copies of the disks we provided to you. Use the copies as your work disks and do not put write-protect tabs on them (covering up the write-protect notches on the side of the disk jacket).

If you are using a large capacity ProDOS volume, such as a hard disk or Unidisk 3.5,™ you may wish to copy the contents of the Pinpoint Toolkit over to a subdirectory within the ProDOS volume. You will have to modify the application list used with RunRun to reflect the new ProDOS paths. You will want to copy the contents of disk 2, side 1 (the desktop accessories) over to the standard location that holds your other Pinpoint desktop accessories.

The Disks

The Pinpoint Toolkit is distributed on 2 *flippy* disks, each containing two sides of information.

Disk 1 contains RunRun on side 1. It is labeled as "/PP.RUNRUN." RunRun is discussed in Chapter 3 of this manual. The flip side of disk 1 is side 2, and it is labeled "/PP.TOOLKIT." It contains the Advanced Installation Facility, a wide-calendar print program, and the Dialer configuration program.

Disk 2 contains the desktop accessories that are distributed with the Pinpoint Toolkit — the Accessory Mover, the Pop-Up Filer, the Hex Calculator, the Video Resource Editor, the Resource Converter, Printer Control Program, and the Memory Window. The actual desktop accessories can be found on side 1 of the disk. If you have stored your other Pinpoint desktop accessories under a ProDOS subdirectory, you should copy the contents of this disk over to that subdirectory. The flip side of this disk, side 2, contains the source code for many of the desktop accessories. Various libraries are also included on this disk.

Chapter 2: The Pinpoint Internal Architecture

Figures

About this Chapter

This chapter focuses on the internal architecture of the Pinpoint Desktop Manager. It provides insight for curious Pinpoint users as well as programmers.

A Technical Overview

Computer hackers are very individualistic. If we had to stereotype this group of people, it would be safe to say that they are opinionated, clever, work strange hours, appear obsessed at all times, and fail to pass the Fortune 100 dress code. Given all that, Pinpoint has a challenge ahead. The Pinpoint Desktop architecture is fairly simple. It provides some basic extensions to ProDOS so that desktop accessories can become a reality in the existing Apple IIe or IIc. It is not the end-all, but is truly a work horse — written by hackers for hackers.

Pinpoint is a handy "add-on" to ProDOS applications such as AppleWorks,™ Point-to-Point,™ InfoMerge,™ BusinessWorks,™ RunRun,™ etc... It enables you to suspend the current program and carry out an accessory function, such as a notepad, spell-checker, phone dialer, or lawn sprinkler. In some instances, such as the Pinpoint Spelling Checker,™ it manages two ongoing tasks by managing "virtual machines" for each process in memory.

Pinpoint essentially mimics the Macintosh™ desktop metaphor without actually using a mouse. The design was intentional, since (a) the actual use of mice on the Apple IIe/C is currently minimal, and (b) the memory overhead for processing mouse events is large.

There are several other important distinctions, however, from the Macintosh. First, although there is a functional library available to the desktop accessory, you do not have use it. A desktop accessory is simply a ProDOS system file with a few constraints that are stated below. As a programmer, you are not mandated to use a specific toolbox in ROM to do everything. The Apple II™ hackers are far too unique and individual to fall into the Macintosh doctrine. They prefer to "program on the metal." The success of the Apple II is clearly rooted in its software base. The software base is broad, and numerous metaphors have been implemented successfully. Second, a desk accessory is *disk resident* (or resides on a RAM drive), so it is not limited by the amount of available memory. The dispatcher provides for segmented accessories up to 16 megabytes in size. As ProDOS reaches beyond that limitation, so will Pinpoint.

Several important hardware trends have been underway in the Apple II market for some time now. Apple II users are willing to upgrade their machines to accommodate more memory, a more powerful CPU, larger mass storage capacity, and telecommunications. The future computing power for the Apple II will only become more powerful as RAM and CPU price/performance curves work in favor of the consumer.

Open Architecture

One of the key features of Pinpoint is that it is an expandable accessory environment. The Pinpoint Desktop Manager initially comes with eight valuable accessories, but allows for installation for up to 16. The hardware analogy to this is the slot in the Apple IIe. You can add new desktop accessories such as the Pinpoint Spelling Checker, a pop-up thesaurus, pop-up graphics, and much more. An *accessory* is a ProDOS interpreter (SYSTEM file) that has a maximum internal memory requirement of 36K and adheres to some reasonable constraints. A segmented accessory can be up to 16 megabytes in size.

How Pinpoint Works

The accessory manager (referred to as the *dispatcher*) works along with the *poller*. The poller is a small piece of code located in the $300 page of main memory (subject to change between applications). It views each key pressed looking for an ⌘-P combination of keys. Once it finds this combination, it transfers control to the dispatcher. The dispatcher is located in the higher 12K of alternate RAM memory, as depicted in Figure 2.1. It is "swapped" into lower memory ($0C00-$1FFF) after it has been invoked. The area of alternate high memory ($D000-$E3FF) is exchanged with main memory. The pop-up kernal resides there. Figure 2.1 shows the sleep-state and runtime memory map for Pinpoint.

Next a pop-up window appears, and you select a particular desktop accessory to run. At this point the Pinpoint dispatcher must suspend the primary application. It essentially puts the primary application into a sleep state. Hence, if the time from a clock is displayed on the video, it will no longer be updated by the primary application. The requested accessory requires a memory region to run in. The dispatcher knows the memory requirements and saves this portion of main memory in a temporary file "W.TMP" on a ProDOS device. Other critical information, such as the current zero page and stack contents are saved away also. The $200 page is not saved, however.

The desktop accessory you requested is then loaded into memory beginning at $2000. It can occupy the area between $2000 and $BEFF. If an accessory plans on using the printer library, the font library, or the double hi-res graphic library, it should not occupy the space above $AFFF. The entire accessory does not have to be loaded in at once, it can be segmented and brought in by way of the memory management calls available in the Pinpoint dispatcher. A defined "boot length" for each accessory is set up. This determines how much of the accessory should initially be brought into memory.

Once loaded, the desktop accessory is rendered control via a JSR instruction. An RTS instruction or a dispatcher EXIT call (preferred) would relinquish control back to the dispatcher. The dispatcher has constructed in the temporary file "wake-up" instructions. This is used to reinstate the primary application and give back control.

Figure 2.1: Pinpoint Sleep-State and Runtime Map

Resides at:		*Maps to:*
$D000-$E3FF	Pop-up dispatcher and functional library	$0C00-$1FFF
$E400-$E7FF	Error handler (for internal use)	$0800-$0BFF
$E800-$EBFF	Font library	$BB00-$BEFF
$EC00-$F0FF	Double hi-res graphic library	$B500-$B9FF
$F500-$F9FF	Printer library	$B000-$B4FF
$FA00-$FDFF	System I/O buffer	$0800-$0BFF
$FE00-$FFFF	Video save area, zero page, stack, etc.	$2000-$21FF

The Dispatcher Function Library

When you are writing desktop accessories, there are many functions that need to be carried out that are common. For instance, when an accessory appears on the screen, it typically resides in a window that occupies a section of the screen. It is the responsibility of the accessory to ensure that any portion of the video display that is distorted will be recovered when you exit the accessory. The CAPTURE function, described below, handles this for the accessory. In addition, the PROJECT function recovers the capture contents upon exiting. ProDOS I/O functions, such as OPEN, CLOSE, READ, WRITE, and BLOAD are examples of very common functions that must be carried out by an accessory, and are all supported by the dispatcher function library.

Figure 2.2: Accessory Runtime Memory Map

$0800-$0BFF	Available for system I/O buffer (via dispatcher call)
$0C00-$1FFF	Dispatcher library
$2000-$AFFF	Accessory space
$B000-$B4FF	Printer library
$B500-$B9FF	Double hi-res library
$BA00-$BAFF	Reserved
$BB00-$BEFF	Font set

Figure 2.3: Dispatcher Global Variables

Name	*Size*	*Address*	*Meaning*
DISK.RC	byte	$1143	ProDOS return code for disk I/O action.
REFNUM	byte	$1144	ProDOS reference number assigned to an open file.
RECNUM	word	$1145	Word, low byte first, denoting a logical record number. Currently unused.
LRECL	word	$1147	Logical record length. Used when issuing a READ or WRITE call for transfer count.
ACCESS	byte	$1149	ProDOS Access code for CREATE.
FILETYPE	byte	$114A	ProDOS filetype for file being opened.
PATH	word	$114B	Pointer to pathname. When the accessory first gains control, the path will point to itself.
OFFSET	3 bytes	$114D	File Offset (3 bytes, low byte first). Positions for READ or WRITE calls.

Figure 2.3: Dispatcher Global Variables *(Continued)*

Name	*Size*	*Address*	*Meaning*
IOBUFFER	word	$1150	Pointer to 1K ProDOS system I/O buffer.
IOB	16 bytes	$1152	I/O Block area for ProDOS to make I/O requests.
FILELEN	3 bytes	$1164	Length of file most recently loaded via BLOAD call.
DBUFFER	word	$1166	Pointer to data source or destination for READ, WRITE or BLOAD.
CREATE.SB	byte	$1169	Status byte. If high bit is on ($80), OPEN should not CREATE a file if file is not found.
DEVICE.ID	byte	$116A	Device used for ONLINE request
ROW	byte	$116B	Video row (0-23) for display.
COL	byte	$116C	Video column (0-79) for display output.
WIDTH	byte	$116D	
LENGTH	byte	$116E	Length of field, used in INVERT call.
MINROW	byte	$116F	Video parameters...
MAXROW	byte	$1170	
MINCOL	byte	$1171	
MAXCOL	byte	$1172	
WND.ROW	byte	$1173	
WND.COL	byte	$1174	
WND.WIDTH	byte	$1175	
WND.LEN	byte	$1176	
KEY	byte	$1177	Most recent keystroke hit. High-bit set on if hit in conjunction with ⌘-key.
DRAWCHAR	8 bytes	$1178	Used in DRAW and XDRAW calls.
EXTERNCMD	byte	$1180	ProDOS external call value.

How the Dispatcher Works

The dispatcher transfers control to the accessory and remains resident in main memory at address $0C00 thru $1FFF. In addition, various dispatcher components (font library, printer library, graphic library) occupy the area between $B000 and $BEFF.

A toolkit of handy functions are available for the accessory to use. The accessory does not have to use any of these functions. They are simply there as a means to reduce the amount of code required for an accessory. In order to use these functions, the accessory should set up the dispatcher parameters (located at $1143 through $1180), set the X-Register with the requested function number, and make a call to DISP.SVC ($1100). In addition, most Pinpoint functions require arguments sent down. These arguments must use global variables, or zero page locations ($F9-$FF).

The global variables listed in Figure 2.3 are included in a source code library called DISPATCHER.LIB.

Figure 2.4: Dispatcher Function Summary Guide

Function	*Name*	*Description*
0	OPEN	Opens a ProDOS file
1	CLOSE	Closes a ProDOS file
2	BLOAD	Loads a file into memory
3	READ	Reads data from a ProDOS file
4	WRITE	Writes data to a ProDOS file
5	GETPREFIX	Gets prefix/volume
6	GETVOL	Returns volume pathname
7	DESTROY	Deletes a ProDOS file
8	FREEPAGE	Frees a memory page for use
9	MLICALL	Standard MLI call entry
10	BASCALC	Compute video refresh address
11	PUTSCRN	Place text on video display
12	GETSCRN	Return text string from video
13	INVERT	Inverts area in video display
14	PROJECT	Project a video resource
15	CAPTURE	Copy a region of video display
16	DRAW	Create a ghost region on video
17	XDRAW	Undo ghost region on video
18	PICK	Retrieve a character from video
19	STOR	Place a character on vide
20	GETKEY	Wait for keypress, return value
21	IOBUFF	Allocate/deallocate I/O buffer
22	MMU	Memory manager services
23	PRLIB	Allocate/deallocate printer library
24	GRLIB	Allocate/deallocate graphic library
25	VERSIONUM	Return version number of Pinpoint
26	ERRBELL	Generate a gentle tone
27	CLOCKOK	Enable/disable RunRun clock
28-30	RESERVED	Reserved for future use
31	EXIT	Accessory exit

The source code library PINPOINT.H contains the equates for the function calls found in Figure 2.4.

Dispatcher Calls

A dispatcher call is made by setting the required dispatcher global variables and transient variables with the appropriate values, setting the X-register to the function number, and calling DISP.SVC ($1100).

For instance:

```
LDX #ERRBELL                ;Ring the error tone
JSR DISP.SVC
```

Function: OPEN (0)

Description: The OPEN function opens a ProDOS file.

Parameters:

PATH — A word pointer that points to the pathname, beginning with the length byte, followed by the actual pathname.

IOBUFFER — Should point to an available 1K system I/O buffer that is available to use. It must be available within the context of the ProDOS bitmap scheme for memory management. It must not extend beyond the address range of the accessory, since this will destroy the primary application. For more information, refer to function 21, IOBUFF.

DISK.RC — Return code from ProDOS, 00 if OK.

REFNUM — Reference number returned by ProDOS.

Example:

```
MYPATH    STR 'TESTFILE'
          ....
          LDA #>MYPATH          ;Point to MYPATH
          STA PATH
          LDA #<MYPATH
          STA PATH+1
          LDX #IOBUFF           ;Allocate I/O Buffer
          JSR DISP.SVC          ;at $0800-$0BFF via Pinpoint
          LDX #OPEN             ;Open "MYPATH"
          JSR DISP.SVC
          LDA DISK.RC           ;Check return code
          BEQ PROCEED
PROCEED   LDA REFNUM            ;Save away ProDOS refnum
          STA HOLDREF
          ....                  ;And continue onward
```

Function: CLOSE (1)

Description: The CLOSE function closes a ProDOS file opened via an open call.

Parameters:

REFNUM Reference number assigned by ProDOS.

IOBUFFER Should point to the 1K system I/O buffer allocated via the OPEN call.

Example:

```
LDA HOLDREF          ;Restore reference number
STA REFNUM           ;(if using more than 1 file)
LDX #CLOSE           ;And close file
JSR DISP.SVC
```

Function: BLOAD (2)

Description: BLOAD performs a simple binary load of any type of file at a particular location in memory. The memory should be within the assigned area of the accessory.

Parameters:

PATH — Word pointer to the filename.

DBUFFER — Word pointer to a location for the BLOAD.

IOBUFFER — Should point to an available 1K I/O area that is available to use. It must not extend beyond the range of the accessory, and cannot conflict with current ProDOS memory usage. For more information, refer to function 21, IOBUFF.

DISK.RC — Return code from ProDOS, 00=OK.

FILELEN — Returned actual length of BLOAD.

Example:

```
MYPATH    STR 'TESTFILE'
LOADADDR  EQU $4000
          ...
          LDA #>LOADADDR          ;Set Bload address
          STA DBUFFER
          LDA #<LOADADDR+1
          STA DBUFFER+1
          LDA #>MYPATH            ;Set pathname
          STA PATH                ;to "TESTFILE"
          LDA #<MYPATH
          STA PATH+1
          LDX #BLOAD              ;Perform BLOAD
          JSR DISP.SVC
          LDA DISK.RC             ;Check I/O return code
          BEQ PROCEED
          .....
PROCEED   EQU *                   ;FILELEN contains length of
                                  ;BLOAD.
```

Function: READ (3)

Description: The READ function will read some data from a ProDOS file. This a random access read. Data will be read from the file position specified in the OFFSET parameter.

Parameters:

LRECL — Logical record length to be read in, expressed in bytes. Up to $8000.

OFFSET — The byte offset into the file to be read. This is a 3-byte value, low order first.

REFNUM — The ProDOS reference number, returned via the OPEN call.

DBUFFER — Word pointer to the memory location to be read into. Value must be between $2000 and the top of memory for the accessory.

IOBUFFER — Pointer to 1K system I/O buffer.

DISK.RC — Return code from ProDOS, 00=Ok.

Example:

```
HOLDREF   DFB 0                ;Stored value from OPEN call
READADDR  EQU $4000            ;Location to read record
READLEN   EQU $0400            ;Record length = 1024 bytes
          ...
          LDA HOLDREF          ;Set reference number
          STA REFNUM           ;for READ.
          LDA #>READADDR       ;Location to read into
          STA DBUFFER
          LDA #<READBUFF
          STA DBUFFER+1
          LDA #>READLEN        ;Set read length to $400 bytes
          STA LRECL
          LDA #<READLEN
          STA LRECL+1
          LDX #READ            ;Perform READ call
          JSR DISP.SVC
          LDA DISK.RC          ;Check return code...
```

Function: WRITE (4)

Description: The WRITE function will write some data to a ProDOS file. This is a random write similar to the READ function described above.

Parameters:

LRECL	Logical record length to be written. Word value, expressed in bytes.
OFFSET	The byte offset into the file to write to. The is a 3-byte quantity with the first byte being the low-order byte.
REFNUM	The ProDOS reference number.
DBUFFER	Word pointer to the memory location to be written from.
IOBUFFER	Pointer to 1K System I/O area.
DISK.RC	Return code from ProDOS, 00 if OK.

Example:

```
HOLDREF   DFB 0              ;Stored value from OPEN call
WRITADDR  EQU $4000          ;Location to write from
WRITLEN   EQU $0400          ;Record length = 1024 bytes
          ...
          LDA HOLDREF        ;Set reference number
          STA REFNUM         ;for READ.
          LDA #>WRITADDR     ;Location to write from
          STA DBUFFER
          LDA #<WRITBUFF
          STA DBUFFER+1
          LDA #>WRITLEN      ;Set write length to $400 bytes
          STA LRECL
          LDA #<WRITLEN
          STA LRECL+1
          LDX #WRITE         ;Perform WRITE call
          JSR DISP.SVC
          LDA DISK.RC        ;Check return code...
```

Function: GETPREFIX (5)

Description: GETPREFIX can accomplish two different tasks:

1. Get the current default prefix.
2. Get the online volume name.

Parameters:

To get the default prefix:

$F9	Function switch. Set to $00 to return the default prefix.
$FA/$FB	Word pointer to buffer area to hold the prefix returned. A 64-byte buffer should be allocated.

To get an online volume:

$F9	Function switch. Set to $FF to signal that the volume (found in DEVICE.ID) should be returned.
$FA/$FB	Pointer to buffer area to hold the prefix returned.
DEVICE.ID	Contains the ProDOS device number of the device to have its volume returned.

Note: Function 6, GETVOL, described on the next page is a more convenient way to accomplish the Online Volume task above.

Warning: If an online volume request is made and the value of DEVICE.ID is set to 0, then all online volume names will be returned. This requires a buffer of 256 bytes to accommodate up to 16 potential ProDOS paths.

Example:

```
HOLDPATH  DS 64,0                ;Allocate 64-byte buffer
          ...
          STZ $F9                ;Set Function switch to 0.
          LDA #>HOLDPATH         ;Set pointer to path
          STA $FA
          LDA #<HOLDPATH
          STA $FB
          LDX #GETPREFIX         ;Call dispatcher
          JSR DISP.SVC           ;to get default path
          ....
```

Function: GETVOL (6)

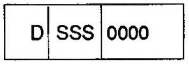
Description:

GETVOL retrieves information particular to a volume. It returns both the volume name (formatted as a ProDOS pathname) and the volume information. The volume name is returned as a ProDOS path beginning with the length byte, followed by a "/ " and a pathname, and terminated with a "/ ".

A ProDOS GET_INFO call is made prior to returning control to the desktop accessory. The results of that call are returned in the IOB variable. Refer to the ProDOS technical reference material regarding the results of a GET_INFO call.

Parameters:

$F9 Contains the device ID number used to locate the volume. Format of byte is as follows:



D:Drive 0=Drive 1
1=Drive 2
S:Slot 0..7 allowed

IOBUFFER Pointer to 1K System I/O buffer for ProDOS.

PATH Word pointer to 16 byte buffer area to contain the volume name of device upon return.

IOB Returned results of a GET_INFO call for the device requested.

DISK.RC ProDOS return code, 00=OK

Example:

```
VOLNAME  DS 16,0
         ...
         LDA #$60          ;Slot 6, Drive 1 Device
         STA $F9
         LDX #GETVOL       ;Make GETVOL Call
         JSR DISP.SVC      ;Assume I/O Buffer allocated
         LDA DISK.RC       ;Everything OK?
         BEQ GOT.DEV
         ...
```

Function: DESTROY (7)

Description: The DESTROY function destroys *(erases)* a particular path. *It should obviously be used with caution.*

Parameters:

PATH	Pointer to name of path to be destroyed.
DISK.RC	Return code from ProDOS, 00=OK.

Example:

```
WORKPATH  STR 'WORKFILE'
          ...
          LDA #>WORKPATH          ;Delete "Workfile"
          STA PATH
          LDA #<WORKPATH
          STA PATH+1
          LDX #DESTROY
          JSR DISP.SVC
          ...
```

Function: FREEPAGE (8)

Description:

FREEPAGE frees up a memory page for I/O use with ProDOS. ProDOS maintains a bitmap in the global page ($BF00: main memory) that determines which memory pages within the lower 48K of main memory can be used for I/O. The FREEPAGE function clears pages for use.

When an accessory is given control, its entire memory space is allocated as free for use by ProDOS. When you exit the accessory, the ProDOS bitmap is set back to its original value.

Arguments:

$F9 Contains the page number. Note that the area still must be preserved by the accessory if it is beyond the address region of the program.

Example:

```
LDA #$30                ;Free page $30
STA $F9
LDX #FREEPAGE           ;Make dispatcher call
JSR DISP.SVC
...                     ;Now $3000..$30FF can be used.
```

Function: MLICALL (9)

Description MLICALL is used to make a standard MLI call that has not been provided for by the Pinpoint dispatcher. For instance, a GET_INFO call on a particular path may be used by this call.

Parameters:

EXTERNCMD Contains a function number for a ProDOS MLI call.

IOB Contains the arguments for a ProDOS MLI call.

DISK.RC ProDOS I/O return code. 00=OK.

Example:

```
WORKPATH  STR 'TESTFILE'
          ...
          LDA #$C4                ;ProDOS command
          STA EXTERNCMD           ;for "GET_INFO"
          LDA #$0A                ;10 arguments
          STA IOB
          LDA #>WORKPATH          ;Set IOB path
          STA IOB+1
          LDA #<WORKPATH
          STA IOB+2
          LDX #MLICALL            ;Make dispatcher call
          JSR DISP.SVC
          LDA DISK.RC             ;Check I/O results
          BEQ GOT.INFO
          ...
```

Function: BASCALC (10)

Description:

BASCALC compute the base video address. In order to place characters or a string onto the video display, a corresponding address must be computed for the target location on the video. The target location is expressed in Cartesian coordinates that correspond to the vertical row and horizontal column on a 24 row by 80 column screen image. The upper left corner is referred to as (0,0). The video refresh memory is not contiguous, therefore the computation is not intuitive.

In addition, if the graphic library is invoked via the GRLIB call, the computation will be altered from the text video address to the double hi-res video image.

Only the vertical row is actually needed for the computation. This call should be made prior to issuing a PICK or STOR dispatcher call.

Arguments:

ROW — Video row (0-23) for display output

$FE/$FF — Word returned, low byte first, containing the base address within the video refresh to be used for the PICK or STOR call.

Example:

```
LDA #0                  ;Grab character from upper left
STA ROW                 ;corner of the screen
STA COL
LDX #BASCALC            ;Perform computation
JSR DISP.SVC
...                     ;Grab character now...
LDX #PICK
JSR DISP.SVC
```

Function: PUTSCRN (11)

Description:

PUTSCRN places a string of characters on the video display. The advantage of using this call over a standard COUT call is speed. The string is placed directly into the video refresh. No conversion of character to normal or inverse (or from icon) is made.

The other advantage of using the PUTSCRN call over COUT is that Pinpoint detects whether the graphic driver has been invoked via the GRLIB call. If so, it outputs the string characters to the double hi-res image automatically. It uses the font library to generate the characters on the graphic display. This allows the programmer to write accessories in the same way for text and graphics.

Parameters:

$F9	Length byte of the string. Valid value range is 1 to 80 decimal.
$FA/$FB	Word pointer to the actual string.
ROW	Vertical video row (0-23) for display output.
COL	Horizontal column (0-79) for display output.

Example:

```
TESTSTRG  ASC 'THIS IS A TEST'
          ...
          LDA #10                ;Place string at row 10
          STA ROW
          LDA #20                ;and column 20
          STA COL
          LDA #>TESTSTRG         ;Point to string
          STA $FA
          LDA #<TESTSTRG
          STA $FB
          LDA #14                ;Length of test string
          STA $F9
          LDX #PUTSCRN           ;Place on video now
          JSR DISP.SVC
          ...
```

Function: GETSCRN (12)

Description: GETSCRN returns a string that contains a portion of the video refresh in it. This function is very handy for "cutting" information from the video area. It is similar to the PUTSCRN function described above, but the "read" equivalent.

This function is not supported when the graphic driver is active, however. Successive calls of the PICK function must be made in order to accomplish this while in a graphic mode.

Parameters:

$F9	Contains the length of the string to be retrieved. Valid value range is 1 to 80.
$FA/$FB	Word pointer to target string.
ROW	Vertical video row (0-23) for retrieval.
COL	Video column (0-79) to begin retrieval.

Example:

```
TESTSTRG  DS 80,0
          ...
          LDA #$00              ;Read first row of video
          STA ROW
          STA COL
          LDA #80               ;for 80 bytes in length
          STA $F9
          LDA #>TESTSTRG        ;Point to string
          STA $FA
          LDA #<TESTSTRG
          STA $FB
          LDX #GETSCRN
          JSR DISP.SVC
          ...
```

Function: INVERT (13)

Description: The INVERT function inverts an area of the video refresh. This function is very handy for pull-down windows in which commands are highlighted and deactivated. The same is true for dialog boxes. Two successive calls to this function will return the screen to its original contents.

This function is supported when the graphic driver is active as well.

Parameters:

ROW	Video row (0-23) where highlighting occurs.
COL	Video column (0-79) where highlighting begins.
LENGTH	Length of field to be highlighted.

Example:

```
LDA #10                 ;Highlight at (10,20)
STA ROW
LDA #20
STA COL
LDA #24                 ;for a length of 24 bytes
STA LENGTH
LDX #INVERT
JSR DISP.SVC
...
LDX #INVERT             ;Now de-highlight area
JSR DISP.SVC
```

Function: PROJECT (14)

Description: The PROJECT function projects a VRE resource onto the video refresh. This technique is used to place windows, dialog boxes, and even strings onto the video refresh.

This function is supported when the graphic driver is active as well. It can be used to restore the video contents saved away by the CAPTURE function call.

Parameters:

$FA/$FB Word pointer to VRE resource.

Example:

```
LEADWND  DFB....                ;VRE Resource
         ...
         LDA #>LEADWND          ;Project the resource
         STA $FA
         LDA #<LEADWND
         STA $FB
         LDX #PROJECT
         JSR DISP.SVC
         ...
```

A *video resource* is a binary image of the particular window. It begins with a 6-byte header followed by a compressed image of the video refresh clip region. The header begins with the upper left row (0-23), then the upper left column (0-79). The third byte contains the width of the window; and the fourth byte contains the length of the clip region. The fifth and sixth bytes contain other information for the Pinpoint dispatcher.

If you wish to position a window to another location other than its default location, simply change the contents of the first two bytes of the resource to another value. The value must be valid, and not position the window off the screen.

Function: CAPTURE (15)

Description:

CAPTURE saves a rectangular portion of the video refresh (as defined by the parameters below) in a buffer area. The purpose of this function is to make it easy for a desk accessory to save and restore the video to its past splendor after you exit the accessory.

This function is typically called before projecting a pull-down window or dialog box, so as to save the area of the video beneath the upcoming window.

To restore the image captured, use the PROJECT call.

Text screen: buffer size=(WND.LEN*WND.WIDTH)+6 bytes

Graphic screen: buffer size=((WND.LEN*WND.WIDTH) * 8)+6 bytes

Parameters:

$FA/$FB	Word pointer to buffer area used to capture contents being displayed.
WND.ROW	The upper left corner row of the area to be captured. Range is 0 through 23.
WND.COL	The upper left corner column of the area to be captured. Range is 0 through 79.
WND.WIDTH	The horizontal width of the capture area. Range is 1 through 80.
WND.LEN	The vertical length of the capture area. Range is 1 through 24.

Example:

```
STORAGE   DS 1926                  ;Save area for whole video
          ...
          LDA #>STORAGE            ;Point to storage
          STA $FA
          LDA #<STORAGE
          STA $FB
          LDA #0                   ;Start at upper left corner
          STA WND.ROW
          STA WND.COL
          LDA #80                  ;80 columns wide
          STA WND.WIDTH
          LDA #24                  ;24 rows long
          STA WND.LEN
          LDX #CAPTURE             ;Capture video
          JSR DISP.SVC
          ...
          LDA #>STORAGE            ;Now recover video
          STA $FA
          LDA #<STORAGE
          STA $FB
          LDX #PROJECT
          JSR DISP.SVC
```

Function: DRAW (16)

Description: The DRAW function creates a ghost-like outline on the video refresh. It can be used to drag or size a window.

This function is not supported when the graphic driver is active.

Parameters:

DRAWCHAR 8-byte area containing values as follows:

Description	*Default*
Upper left corner	$5A
Top center character	$5C
Upper right corner	$5F
Leftmost middle character	$5A
Rightmost middle character	$5F
Bottom left corner	$A0
Bottom center character	$4C
Bottom right corner	$A0

$FA/$FB Word pointer to a buffer area that is used to capture characters that lie underneath the ghost image being generated. The ghost image can be "undone" via the XDRAW call.

The size of the buffer area is:
(WND.WIDTH*2)+WND.LEN

WND.ROW The upper left corner of the ghost image.

WND.COL The upper left corner column of image.

WND.WIDTH The horizontal width (in columns) of the ghost image.

WND.LEN The vertical length (in rows) of the ghost image.

MINROW Should be set to the value of WND.ROW.

MAXROW Should be set to WND.ROW+WND.LENGTH-1.

MINCOL Should be set to WND.COL.

MAXCOL Should be set to WND.COL+WND.WIDTH-1.

Warning: The default values should be restored if changed by the accessory so that all desktop accessories can make the assumption that the DRAWCHAR values are the default values.

Function: XDRAW (17)

Description: XDRAW restores the image behind a window outline created by the DRAW call described above. It is used to "undo" a recent ghost image.

Parameters:

$FA/$FB	Word pointer to the buffer area used by the DRAW call. The ghost image will be undone with the contents of this buffer.
WND.ROW	The upper left corner row of the ghost image being undone.
WND.COL	The upper left corner column of the ghost image being undone.
WND.WIDTH	The horizontal width (in columns) of the ghost image being undone.
WND.LEN	The vertical length (in rows) of the ghost image being undone.
MINROW	Should be set to the value of WND.ROW.
MAXROW	Should be set to WND.ROW+WND.LENGTH-1.
MINCOL	Should be set to WND.COL.
MAXCOL	Should be set to WND.COL+WND.WIDTH-1.

Function: PICK (18)

Description: PICK retrieves a single character from the video refresh. The BASCALC function should be called prior to this function to set up the offset into the video refresh. See function BASCALC for details.

This function is supported when the graphic driver is invoked, although a slightly different result is returned.

Parameters:

$FE/$FF	Should be set by BASCALC call to contain the offset into the video refresh.
Y-reg	Sent to function containing the column number (0-79) to be PICKed.
A-reg	Returned with the contents of the video.
$FA/$FB	Only used when graphic driver is active. This should be set as a word pointer to an 8-byte buffer area where the contents of the video are returned.

Example:

```
LDA #0                  ;Grab character from (0,0)
STA ROW
LDX #BASCALC            ;Compute video address
JSR DISP.SVC
LDY #0                  ;Column 0
LDX #PICK
JSR DISP.SVC            ;Grab the character
STA ...                 ;A-Reg holds character.
```

Function: STOR (19)

Description: STOR places a character in the video refresh. This technique is quicker than the COUT function, and is supported in both graphic and text modes.

Parameters:

$FE/$FF	Word containing the video offset. This value is set by BASCALC.
Y-Reg	Sent down containing the column number.
KEY	Sent down containing the character to be placed on the video.

Example:

```
STZ ROW                 ;Upper left corner
LDX #BASCALC            ;Compute video offset
JSR DISP.SVC
LDA #$C1                ;Place an "A" in upper left
STA KEY
LDY #0                  ;Column 0
LDX #STOR
JSR DISP.SVC
```

Function: GETKEY (20)

Description: The GETKEY function retrieves a keystroke from the keyboard. A wait loop is processed until a keystroke is hit. A flashing cursor is not implemented.

If the ♁ or keys are hit in conjunction with a keystroke, the value returned has its high bit set on, otherwise it is turned off.

Parameters:

KEY — Returned with value of key pressed. This value is also contained in the A-register.

Example:

```
LDX #GETKEY          ;Wait for a keypress
JSR DISP.SVC
CMP #$B1             ;♁-1?
...
```

Function: IOBUFF (21)

Description:

This function allocates/deallocates a 1K system I/O buffer at $0800 in main memory. The IOBUFFER global variable is initialized with a word pointer to the buffer location. Refer to the global value, rather than the absolute location of $800.

This call is useful for accessories that must open a file and require a system I/O buffer to do so, or for those applications that need working space of $400 bytes.

By making a second call to this function, you deallocate the buffer.

Parameters:

IOBUFFER Returned word pointer that indicates the location of the system I/O buffer.

Example:

```
LDX #IOBUFF                ;allocate I/O buffer
JSR DISP.SVC
...
LDX #IOBUFF                ;restore contents of $800-$BFF
JSR DISP.SVC
```

Function: MMU (22)

Description:

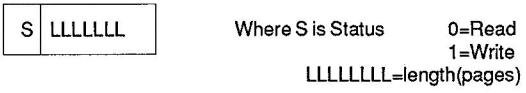
MMU is the memory management function that allows the desktop accessory to *page* memory in or out. Paging means exchanging the contents of a disk file with memory or vice versa. This function is key to constructing desktop accessories that are greater than 36K in size.

A desktop accessory can be broken down into segments that can be loaded into main memory upon demand. This is analogous to *chaining* in a BASIC program. A program can implement a demand-paging algorithm to provide a virtual machine size up to 16 megabytes in size, if necessary.

The memory management function is also used when portions of auxiliary memory (such as Page HGR2X) must be saved to the disk because that memory is needed by the graphic display.

Parameters:

$F9 Read/Write byte set as follows:



$FA/$FB Source of Read/Write. If reading (high bit set off on Read/Write byte at $F9), then this should hold the file offset on the disk file. If writing, this word should contain the memory address (on a page boundary) from which to begin writing.

$FC/$FD Destination of Read/Write. If writing memory to disk, this should refer to the file offset on disk. If reading, this should refer to the target destination in memory for the disk contents.

DISK.RC Return code from ProDOS, 00 if OK.

Note: The file offset is expressed as a page number, not an absolute address. This is expressed in 256-byte increments from the start of the accessory. This allows an accessory to address up to 16 megabytes of memory from disk with a two-byte identifier.

Example:

```
LDA #$94          ;Write $14 pages
STA $F9
STZ $FA
LDA #$0C          ;Source address=$0C00
STA $FB
LDA #$05          ;File offset=5*256=1280 bytes
STA $FC           ;from start of file
STZ $FD
LDX #MMU
JSR DISP.SVC
LDA DISK.RC
BNE ERROR
...
```

Function: PRLIB (23)

Description: PRLIB invokes the Pinpoint printer library. The printer library consists of a specially constructed printer and interface card drivers that deal with text and graphic printing. Each printer and interface card are different, and have specific command sequences, hardware strobe locations, etc.

Printing simple ASCII text is rather trivial. Standard BASIC or PASCAL 1.1 hooks in the firmware can be used in a very standard way to output text. Printing graphics, however, where 8-bit output is required, is much more difficult. In this case, the Pinpoint device drivers communicate directly with the hardware to transfer characters through the interface cards.

The printer library is divided into three distinct sections. The first component is a universal driver that maintains standard entry points for all printers and interface cards. The second component is an interface-specific driver. This component is generated by the install facility. The third component is a printer-specific driver. It contains the peculiar escape codes and associated logic to generate text and graphic printing.

The printer library is "swapped" into main memory from auxiliary memory at location $B000 thru $B4FF. A desktop accessory cannot have code in this area. It must run concurrently when printing.

There are five primary entry points to the printer library. They are as follows:

$B000	PRTINIT	Initialization of the printer
$B003	PRTCLOS	Resets the printer
$B006	PRTWRIT	Write a character/string to printer
$B009	PRTCMD	Command printer to take an action
$B00C	PRTSTAT	Poll state of the printer driver

Allocating the Printer Library

The printer library is allocated by making a PRLIB function call to the Pinpoint dispatcher. It is de-allocated by making another dispatcher call. This is similar to the IOBUFF call.

Example:

```
LDX #PRLIB          ;Allocate printer driver
JSR DISP.SVC
JSR $B000           ;Now initialize printer
....
```

PRTINIT You would typically call this entry point after you have invoked the PRLIB function. The printer is sent a carriage return <$0D> and a test is made to determine if the printer is online. If it is, then the leading printer protocol (interface codes) are sent to the printer.

Note: If using Version 1.3 or greater of Pinpoint, the firmware is bypassed on the interface card. Characters are directly output to the hardware strobe.

Parameters:

A-reg Return code from printer initialization. If the printer does not appear to be online, and the user presses an escape code, then a value other than $00 is returned, meaning there is a problem.

PRTCLOS This function closes the printer down. The printer is reset, and the default printer characteristics are reset. The printer library is **not de-installed** with this call, this requires a PRLIB function call.

There are no parameters involved in this call.

PRTWRIT This function writes out a character (or a string of characters) to the printer. There are 3 flavors of the PRWRIT command as follows:

X-reg=$00 Print a single character contained in the A-reg. The Y-reg contains a repetition factor.

X-reg=$01 Print a character string to the printer. Y-Reg holds the length of the string to be printed, and $FA/$FB contain a word that points to the string itself.

X-reg=$02 Print a character string similar to above, but terminate the string with an end-of-line string (usually CR and LF).

PRTCMD Command the printer to carry out an action. The X-reg holds the command, the Y-reg holds the settings.

X-reg=$00 Set the printer mode to text. This is the default mode.

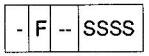
X-reg=$01 Set the printer mode to graphics. This allows all 8-bits/byte of data to flow through the interface card to the printer. The characters are directly output to the card, bypassing the firmware.

X-reg=$02 Set printer pitch. Y-reg holds values as follows:

Y-reg=$80	10 characters/inch
Y-reg=$81	12 characters/inch
Y-reg=$82	17 characters/inch
Y-reg<$80	sets printer to a specific setting.

The specific printer settings can be obtained by a PRTSTAT call.

X-reg=$03 Set printer style. Y-reg holds values as follows:



F:Flag 0=On 1=Off
S:Style 0=Normal
1=Bold
2=Underline
3=Headline
4=Superscript
5=Subscript

X-reg=$04 Set printer spacing. Y-reg holds the value as follows:

Y-reg=$80	8 lines/inch
Y-reg=$81	6 lines/inch
Y-reg=$82	8/72" spacing
Y-reg=$83	1/72" spacing
Y-reg<$80	Printer specific spacing

X-reg=$05 Advances printer to top-of-form (TOF).

X-reg=$06 Set current line to TOF.

Additionally, Version 1.3 or above of Pinpoint supports the following commands:

X-reg=$07 Set the left margin of printer. Y-reg holds the value expressed in characters.

X-reg=$08 Set the right margin of printer. Y-reg holds the value expressed in characters.

X-reg=$09 Set graphic resolution. The Y-reg holds the resolution request. The PRSTAT command returns various printer resolutions.

X-reg=$0a Alert printer the *n* characters following are a graphic pattern to be displayed. A word value at $FA/$FB expresses the *n* value.

PRTSTAT This function is used to query the status of the printer driver, and to ascertain the characteristics and capabilities of the printer driver that has been installed. The X-reg holds the request number as follows:

X-reg=$00 Return the status of various attributes The A-reg is returned with values as follows:

A-reg=$00	Text only printing
A-reg=$01	Text and graphic printing

X-reg=$01 Return a table of available printer pitches.

X-reg (low order) Y-reg (high) returned pointing to table. A-reg contains number of entries in the table.

X-reg=$02 Return a table (same format as above) of available styles.

X-reg=$03 Returns a table of available spacing.

X-reg=$04 Return the current settings. Y-reg should holds query type as follows:

Y-reg=$00	pitch
Y-reg=$01	style
Y-reg=$02	spacing

A-reg is returned with the current setting value.

Additionally, Version 1.3 or greater of Pinpoint has the following functions:

X-reg=$05 Return a table of characters/inch.

X-reg=$06 Return a table of lines/inch.

X-reg=$07 Unused.

X-reg=$08 Find closest available characters per inch. A-reg is returned with value.

X-reg=$09	Return #dots/inch at current resolution.
X-reg=$0A	Return table of available resolutions.

Function: GRLIB (24)

Description:

GRLIB invokes the installation of the graphic library. A graphic driver is installed to communicate to the video refresh. The library consists of a series of double hi-res graphic routines that are located at $B500 through $BEFF. The standard dispatcher I/O hooks (PICK, STOR, PROJECT, PUTSCRN, CAPTURE, INVERT, BASCALC) apply to the double hi-res screen. Several video functions - DRAW, XDRAW, GETSCRN do not work while the graphic driver is in place.

The double hi-res routines allow the desktop accessory to open graphic windows on top of the text screen. This is not actually what has happened, but rather the illusion. The text screen is first converted to an equivalent in double hi-res. Video resources are projected onto the video display as if they were a text display. A character generator, as well as a font library, provide the means to generate text characters on the graphic display.

This function call rolls the graphic library in or out, similar to the PRLIB or IOBUFF call. The first call to the function rolls the graphic library in. The next time it is called, it will roll the graphic library out.

When the graphic library is invoked the area of memory from $B500-$BEFF is consumed by the graphic driver and font library.

$0000-$1FFF	Unchanged to accessory
$2000-$3FFF	Double hi-res main/aux image
$4000-$5FFF	8K image of aux $2000-$3FFF
$6000-$AFFF	Accessory space
$B000-$B4FF	Optional printer library
$B500-$B9FF	Double hi-res graphic library
$BA00-$BAFF	Reserved for future use
$BB00-$BEFF	Font library (for print/graphic)

Note that double hi-res graphic images require a large amount of memory. The GRLIB call should be made from $6000 thru $AFFF in main memory. An 8K image of the auxiliary memory corresponding to HGR2X ($2000-$3FFF) is stored at $4000 in main memory. Typically an accessory would issue an MMU call to save this 8K region to disk, and avail the workspace to the accessory. If the memory is not required, avoiding the MMU call would increase the speed.

Parameters:

$F9 Set the value to $01. This causes the text screen to be converted to a double hi-res equivalent.

Notes:

The PROJECT call will generate a video image just as if this were a text call with a textual resource created by VRE. PROJECT also supports a graphical resource (such as the ruler or Wait-Clock in the GraphMerge). PROJECT detects whether the resource is text or graphic, and acts accordingly.

Function: VERSIONUM (25)

Description: VERSIONUM returns the Pinpoint version number in the A-register. The high order nibble contains the version number, the low order nibble contains the revision number. If an $FF is returned, then the RunRun is the operating environment.

Example:

VVVVRRRR	Version/Revision ($10=Version 1.0)

Function: ERRBELL (26)

Description: The ERRBELL function generates a gentle tone. There are no arguments.

Function: CLOCKOK (27)

Description: CLOCKOK is only available from RunRun. Access to this function while the Pinpoint dispatcher is resident simply returns back to the caller. This function allows you to update the clock display in the upper right portion of RunRun while an accessory is active.

Parameters:

Y-reg	$00=Clock is disabled (default) $01=Clock enabled

Functions 28-30 are reserved for future use.

Function: EXIT (31)

Description: EXIT should be called by an accessory to exit. You can also use an RTS. The advantage of using the EXIT call is that the actual stack pointer does not have to be accurate.

When you exit, the dispatcher regains control, and recovers the state of the world. The temporary file "W.TMP" is read back into memory. The zero page and stack are recovered to their original state as well.

There are no parameters for this call.

Chapter 3: RunRun

Introducing RunRun

RunRun is a versatile ProDOS program selector. It also features the unique capability of being able to run Pinpoint desktop accessories right from the selector. It has mouse-like features, such as pull-down windows and dialog boxes, but does not need an actual mouse. In addition, RunRun makes it possible for you to run a desktop accessory within another desktop accessory. An example of this would be dialing a phone number from a calendar entry.

RunRun was originally designed and used as a "simulation tank" for writing and testing desktop accessories. Program selection became a desired feature, so it was added. Viewing of ProDOS catalogs was needed, so it was implemented. Naturally sorted catalogs were desired, so sorting by filename, file type, modification date, and file size were all added. RunRun is program that is still evolving. It was written using the Pinpoint Toolkit.

Starting Up RunRun

To start up RunRun, turn off the power to your Apple, insert the working copy of your RunRun disk in your startup drive, and then turn the power back on. If your Apple is already turned on, you can "warm boot" by pressing Control-Ó-Reset.

Alternatively, if Apple's ProDOS disk operating system has already been loaded, and you're in the Applesoft command mode (that's where you see the] prompt symbol), you can begin RunRun by inserting the RunRun working disk in any drive and entering a command of the form:

```
-RUNRUN.SYSTEM,S6,D1 [RETURN]
```

The ",S6" indicates the disk drive slot number (usually 6) and the ",D1" indicates the drive number (usually 1).

If you have previously transferred the RunRun files, RUNRUN.SYSTEM and RUNRUN.APPLST, to a ProDOS subdirectory that is not the active directory, you must specify the complete pathname to invoke RunRun. For example, if RUNRUN.SYSTEM is stored in a subdirectory called PINPOINT that is contained in a disk directory called /HARD1, enter:

```
-/HARD1/PINPOINT/RUNRUN.SYSTEM [RETURN]
```

from Applesoft command mode to start up this program.

If you are using the Pinpoint RAM Enhancement Kit,™ you should copy the two files, RUNRUN.SYSTEM and RUNRUN.APPLST, to the RAM device. Your startup program should be RUNRUN.SYSTEM.

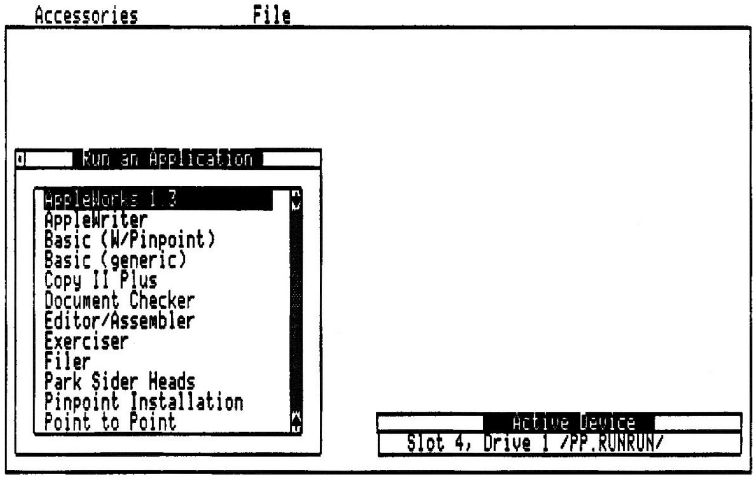
The RunRun Desktop

Once RunRun has been invoked, the RunRun desktop will appear.

The RunRun desktop consists of three areas:

- The command bar appearing on the top of the screen. This allows you to select a command from a pull-down window.
- The main desktop area. This area is used to house the Application List, the ProDOS device table, and the ProDOS directories.
- The message line at the bottom of the screen. This area is used to display informational and error messages.

Figure 3.1: The RunRun Desktop



Initially, RunRun displays the Application list window in the lower left corner of the desktop area, and the ProDOS device window in the lower right corner.

Moving About the Application List Window

By pressing the UP or DOWN cursor arrows, you can position to an application within the list. By pressing the RETURN key, you can invoke the application.

By pressing the ESCAPE key, you will be positioned at the command bar within either the "Accessories" or "File" pull-down menus. Pressing ESCAPE again will return you back to the Application list window.

The Command Bar

The command bar enables you to move to either the "File" or "Accessories" pull-down window by pressing the RIGHT or LEFT cursor arrows. By moving the UP or DOWN arrows within the a pull-down window, you can position to a desired accessory or activity.

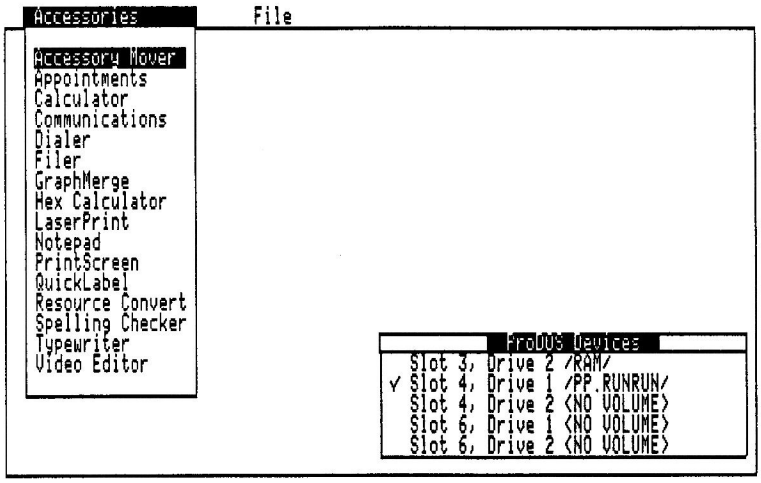
By pressing the ESCAPE key, you will return to the main desktop area window.

The Accessory Pull-Down Menu

Once at the command bar, you can position to the Accessory pull-down menu by pressing the LEFT or RIGHT cursor arrows until it appears on the screen. A list of desktop accessories appears in the window. The accessories were installed on RunRun using the Pinpoint Desktop Manager (Version 1.3 or greater).

By pressing the UP or DOWN cursor arrows, you can position to a desired desktop accessory. Alternatively, you can press a character that corresponds to the first character of the accessory name (i.e. "P" for "PrintScreen"). You can select the desktop accessory by pressing RETURN.

Figure 3.2: The Accessory Pull-Down Menu

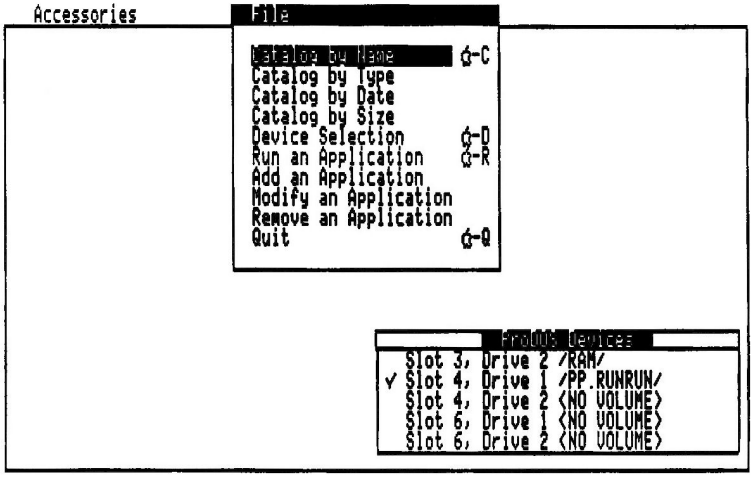


The File Pull-Down Menu

The File pull-down menu, also referred to as the Action window, allows you to obtain sorted catalogs of ProDOS volumes and directories, run ProDOS applications, add, modify, or remove ProDOS applications from the Application list window, and exit RunRun.

By pressing the UP or DOWN cursor arrows, you can position to a desired function. There are several "Quick Keys" defined. They appear on the right side of the File pull-down menu. By pressing ♁-C, you can invoke the catalog function. ♁-D allows you to select a new ProDOS device to focus on, etc.

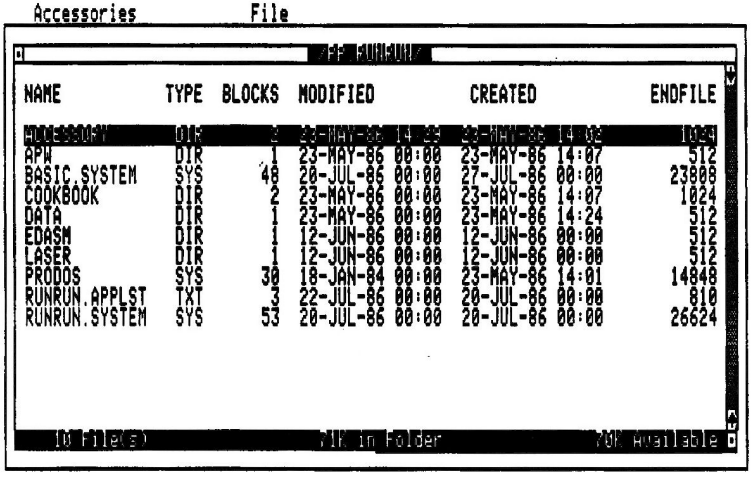
Figure 3.3: File Pull-Down Menu



Catalog of Files

From the File pull-down menu, you can invoke the catalog function. This causes a new window to open up on the main desktop area containing a list of ProDOS files within a directory. The files can be presented in a variety of ways — sorted by the filename, type, modification date, or size.

Figure 3.4: A Catalog Window



By pressing the UP or DOWN arrows, you can move the cursor from one file to another. If all the filenames cannot fit within the catalog window, you can scroll the window with the cursor arrows. If you wish to scroll the window quickly, and an extensive list of files exists, you can use the ✿-UP and ✿-DOWN keys to move a screen at a time.

By pressing the RETURN key while pointing to a system file (type SYS), you can run that application immediately. This is an alternate method of running a ProDOS application.

If you select a directory file (type DIR) using the RETURN key, it will cause that directory catalog to be displayed.

Sizing the Catalog Window

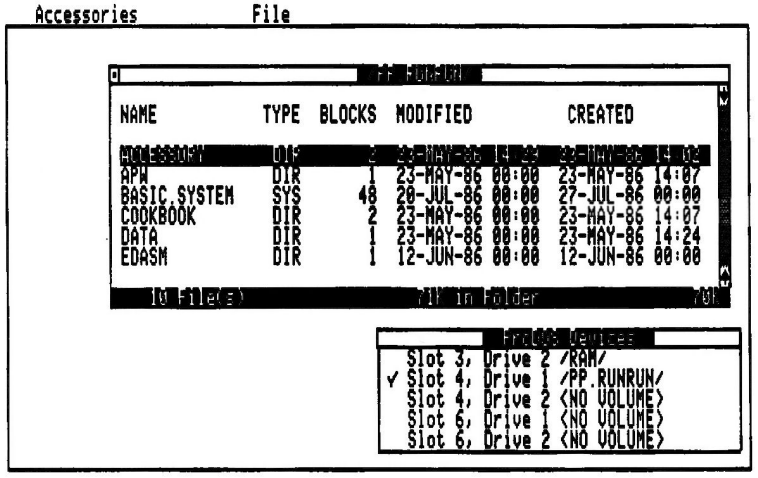
You can size the catalog window to fit your particular needs. Let's say that you really don't care about the creation date or the end of file. Here' how you can shrink the window.

Press the GROW (Ó-G) key to de-activate the window. The border will change slightly. Press the LEFT arrow and UP arrow to size the window accordingly. Press the RETURN key to lock the catalog window.

Dragging the Catalog Window

Once the catalog window has been reduced in size, you can drag it about. Simply press the DRAG (Ó-D) key to enter the drag mode. Press the cursor arrow keys to position the window. Press the RETURN key to lock it in place.

Figure 3.5: Catalog Window Sized and Relocated



Device Selection

From the File pull-down menu, you can change the active ProDOS device by selecting the "Device Selection" menu choice. The installed ProDOS devices will be read, and a list of devices with their current volume name will be displayed. Position to the desired device using the UP or DOWN arrows and select it using the RETURN key.

Selecting a device with a "<NO VOLUME>" name will prevent you from obtaining a catalog of this device, as it does not contain a valid ProDOS volume.

The Application List — Adding an Application

You can customize your RunRun desktop to include a list of frequently run ProDOS applications such as AppleWorks, Pinpoint InfoMerge, Pinpoint Point-to-Point, AppleWriter, etc.

Adding ProDOS applications to the Application list is simple. Select "Add an Application" from the File pull-down menu. You can position to this option quickly by pressing "A" for "Add."

Press RETURN to select this option. The "Add an Application" entry box will appear containing 3 fields to be filled in, the title, system filename, and the program path (if any).

The flashing cursor will appear in the *title* field. The title is a descriptive name assigned by you. For instance, if you wish to add AppleWorks to the Application list, enter "AppleWorks" as the title. Press RETURN or UP or DOWN arrow to move to another field.

The *system file* is the name of the ProDOS "SYS" file that is used to start up the application. This should also contain the volume name of the system file. For instance, if your AppleWorks is stored on a hard disk volume "/HARD1" under a directory "APPLEWORKS", you would enter the field:

```
/HARD1/APPLEWORKS/APLWORKS.SYSTEM
```

The *program path* is an optional field, and is typically used with interpreters such as "BASIC.SYSTEM". It can serve a variety of purposes.

First, you can specify a particular program that should be run. Let's say for instance, you wish to run a BASIC program called "COMPRESSEDPRINT". It resides on your hard disk, volume "/HARD1" under directory "PROGRAMS". You would enter for the program path:

```
/HARD1/PROGRAMS/COMPRESSEDPRINT
```

Rather than running the file "STARTUP" from BASIC at startup, it would run "COMPRESSEDPRINT". The default prefix is set to the path containing the program path.

Within a BASIC program, you can issue the "BYE" command to return to RunRun.

Second, you can specify a default path, rather than a startup program. This causes the default path to be set to this path rather than that of the system path.

A Special Note to RAM Kit Owners:

If you add, modify, or remove applications from the Application list, and the file "RUNRUN.APPLST" is modified, and you have copied this file to the RAM drive for faster operation, it should be copied back to a permanent mass-storage device before you turn off your machine. If you have "Sized" your catalog window, the file "RUNRUN.SYSTEM" is modified, and must be saved to disk as well.

Running an Application

When you choose the "Run an Application" option from the File pull-down menu, a list of applications will appear in the lower left corner of the desktop. You choose an application using the cursor arrows or pressing a character, i.e. "C" for "Copy II Plus". Press RETURN to invoke the application you have chosen.

If the application uses the traditional ProDOS exit procedure, then RunRun will regain control after you have quit your application.

Modifying an Application

When you select "Modify an Application" from the File pull-down menu, you can select an application definition that can be modified. You can change the title, system filename, or the program pathname.

Removing an Application

The "Remove an Application" option on the File pull-down menu allows you to remove unwanted ProDOS application entries from the Application list. Simply position your cursor to the undesired application, and press RETURN. It will be removed from the list.

Quitting RunRun

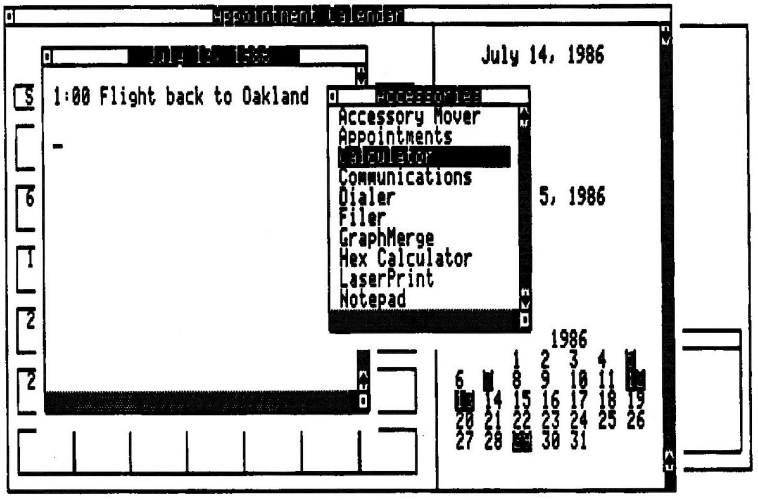
In order to quit RunRun, select the "Quit" option from File menu. Press Control-Ó-Reset to "Warm Boot" your Apple with another startup disk. To return to the RunRun desktop, press the ESCAPE key after you have chosen "Quit."

Running Accessories within Accessories

With the RunRun environment, you can run a desktop accessory directly from the RunRun desktop. You can also invoke a desktop accessory while running another.

For instance, let's say that you invoke the Pinpoint Appointment Calendar from the RunRun desktop accessory window. You suddenly decide that you wish to compute a number with the Calculator. Press ♦-P, and the Pinpoint desktop accessory window will appear.

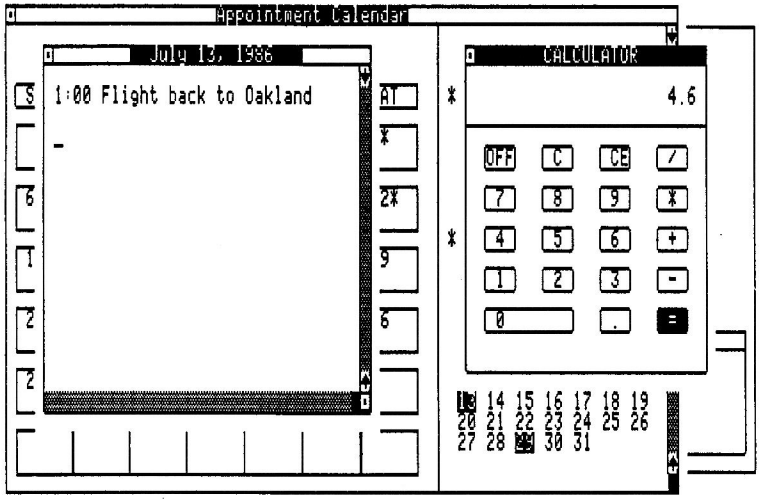
Figure 3.6: Invoking A Desktop Accessory within an Accessory



Selecting the Calculator accessory, and it will appear on the right side of the Calendar. Figure 3.7 below illustrates this point.

Certain accessories cannot be interrupted. GraphMerge, for instance, cannot be interrupted because it is a graphic desktop accessory. The Appointment Calendar cannot be run within the Appointment Calendar because ProDOS will not tolerate two separate applications maintaining the same open ProDOS file (the Appointment Calendar's APPOINTMENTS.PP).

Figure 3.7: Calculator within the Appointment Calendar



Installing RunRun

You can install RunRun by selecting "RunRun Installation" from the Application list that appears on the RunRun desktop. This invokes the system file INSTALLRR. You will be asked to select a target device. This location is where you wish to have the files RUNRUN.SYSTEM and RUNRUN.APPLST copied to. The file RUNRUN.SYSTEM will be placed as the first system file on the device, unless the file "CLOCK.SYSTEM" is the first ProDOS system file on the device. In that case, RUNRUN.SYSTEM will be placed as the second file on the disk.

If you are reinstalling RunRun on a disk, only the system file RUNRUN.SYSTEM will be updated. Your personal application list, RUNRUN.APPLST will not be overwritten.

It is important that these two files remain together. If you plan to use the Pinpoint RAM Enhancement Kit, you will want to copy both of these files to your RAM device.

Partitioning AppleWorks

If you are using the RAM Enhancement Kit, you can allocate a RAM drive, and copy the files "RUNRUN.SYSTEM" and "RUNRUN.APPLST" up to the RAM drive for fast switching between applications. You may wish, for instance, to move between BASIC, Point-to-Point, an assembler, and AppleWorks. If Pinpoint is installed on the application, it is automatically signaled by RunRun that a RAM drive has been allocated, and that various desktop accessories are residing there.

Running AppleWorks expanded with the Applied Engineering Super AppleWorks Desktop Expander™ (S.A.D.E.) can run into some problems, however. Once AppleWorks has been expanded, it becomes a memory monger and desires all memory it can get its code on. This rather greedy approach to resource sharing can present problems. AppleWorks must be taught to behave, and be cognizant that there are others in the world, and that sharing a RAM drive can be rewarding. The partition program modifies AppleWorks so that it shares memory with the RAM drive as specified. You must provide the size of the RAM drive, as well as the path that contains the AppleWorks startup file APLWORKS.SYSTEM.

Chapter 4: General ProDOS Tools

About this Chapter

This chapter covers the three desktop accessories that are general purpose in nature — the ProDOS Filer accessory, the PrintScreen accessory, and MegaDialer, the disk-based phone dialer. The source code to the PrintScreen and Dialer accessories have been included with the Toolkit so they can be modified to meet your individual needs.

The ProDOS Filer Accessory

The ProDOS Filer program is a handy, general utility for ProDOS. It is normally run as a stand-alone application. Mark Wieder, a very crafty programmer at Pinpoint Publishing, modified the ProDOS Filer so as to run as a desktop accessory. It has been changed slightly to provide a greater degree of flexibility. The tutorial that is normally available with the ProDOS Filer has been removed from the accessory version, however.

The ProDOS Filer accessory serves several purposes. First, it shows that almost anything can become a desktop accessory. Pinpoint is a solid technology that facilitates limited multi-tasking for the Apple II and ProDOS. Second, it serves as a great adjunct to various ProDOS applications, such as AppleWorks, BASIC, Point-to-Point, RunRun, etc. The Filer accessory can be very valuable for formatting disks, copying and renaming files, and creating subdirectories.

Invoking the ProDOS Filer Accessory

The ProDOS Filer is a utility program written by Apple Computer, Inc. For in-depth documentation on how to use the ProDOS Filer, refer to Apple Computer's *ProDOS Technical Reference Manual.*

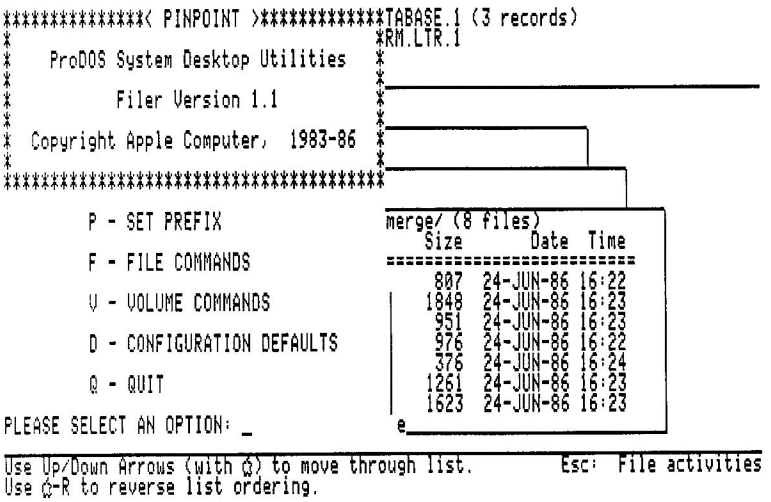
To open the Filer accessory:

Open the Pinpoint menu.	Press ⌘-P for Pinpoint. If you are using RunRun, select the accessory window.
Choose the Filer.	Highlight it and press RETURN from the Pinpoint menu or RunRun.

The left side of the 80-column video display will contain the Filer accessory as shown in Figure 4.1.

The Filer accessory is *menu driven.* This means that menu choices are presented, and you choose a menu item by selecting the corresponding letter. The Filer accessory allows you to carry out file and volume level commands. A volume is the equivalent of a disk.

Figure 4.1: The Filer Accessory



The Main Filer Menu

The Main menu of the Filer accessory offers five choices. The choices are listed in the Filer menu. A key corresponds to the choice as well. For instance, if you press the "F" key, you will select the File commands. The choices from the Main menu are shown in Figure 4.1 above. Volume commands allow you to format, copy, compare or rename volumes (disks). File commands allow you to copy, delete, compare or rename individual files.

To return to the Main Filer menu, press the ESCAPE key at the File Command menu.

The File Command Menu

When you press "F" at the Main Filer menu, the File Command menu will appear on the screen. The File Command menu allows you to set the default prefix, list a directory, copy, delete, rename, compare, lock and unlock files, and even create subdirectories. You can select the desired command by pressing the character that corresponds to the command. Press the ESCAPE key to return to the Main Filer menu.

The Volume Command Menu

Press "V" at the Main Filer menu to bring up the Volume Command menu on your screen. The Volume Command menu allows you to set the default prefix, format, copy, compare, list and rename volumes. In addition, block allocation and bad-block detection capability is provided.

Note that setting the default prefix is a temporary action. When you exit the Filer accessory, the default prefix will be reset to its previous setting.

Press the ESCAPE key at the Volume Command menu to return to the Main Filer menu.

Configuration Defaults

When you press "D" from the Main Filer menu, the Configuration Default menu will appear. From this menu you can set or restore default values. Press the ESCAPE key to return to the Main Filer menu.

Quitting the Filer Accessory

To exit the Filer accessory, and return to your primary application, desktop accessory, or RunRun, press "Q" from the Main menu.

Some Possible Uses

The Filer accessory is handy to use for copying a group of files from one location to another. For instance, you may have numerous files on your RAM drive that you wish to have copied back to disk before you exit AppleWorks. Invoke the Filer accessory, and copy files using the "=" wildcard copy capability.

Another possible use for the Filer accessory is to format disks if you need more storage space to save files. AppleWorks does not feature a disk formatting option. The Filer accessory does, and it can be used from within AppleWorks.

The PrintScreen Accessory

The PrintScreen accessory is very simple, yet handy desktop accessory. It prints the contents of the 80-column video display on the printer. This can be very handy in the RunRun environment, since desktop accessories do not offer a screen dump command (the equivalent of ♁-H from AppleWorks).

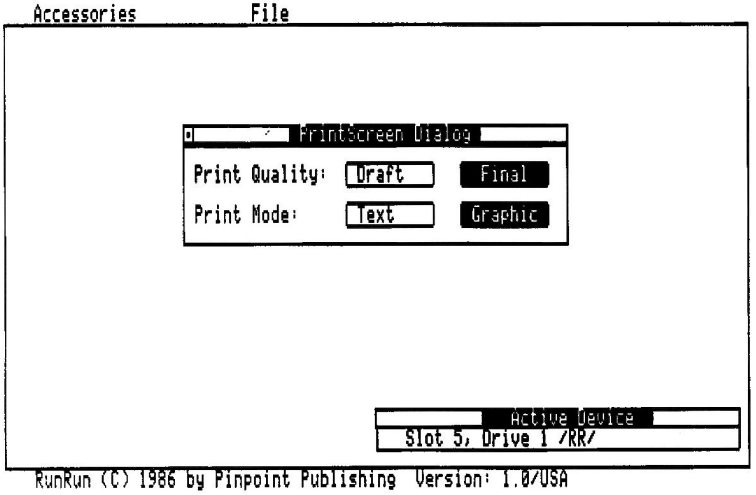
Running the PrintScreen Accessory

To open the PrintScreen accessory:

Open the Pinpoint menu.	Press ♁-P for Pinpoint. If you are using RunRun select the accessory window.
Choose PrintScreen from from the Pinpoint menu or RunRun.	Highlight it and press RETURN.

A dialog box will appear in the center of the display as shown in Figure 4.2 below. You have two choices to make — the print quality and the print mode. You can select your choices by moving the LEFT and RIGHT arrow keys to position to the desired selection, and pressing RETURN.

Figure 4.2: The PrintScreen Accessory



Print Quality

You can set the print quality to draft or final. The print quality is only pertinent if you have chosen the graphic printing mode. A print quality of *final* forces a two-pass print of the graphic representation of the screen.

Print Mode

The print mode can be set on most printers to either text or graphics. Certain printers, such as Common and Okidata™ are not supported in graphic mode.

Text printing is fairly fast. The existing 80-column video display is printed out less any mousetext icons that appear.

Graphic printing is somewhat slower, because each character is plotted to the printer. This requires about 18 times the amount of effort than a straight text print requires. Mousetext video

characters can be printed, however. A true *WYSWYG* print is made. WYSWYG is an abbreviation for "What you see is what you get."

Source Code

The source code to this accessory is distributed on the Toolkit source code disk. You can modify the source code as needed to meet you particular needs.

The Disk-Based Dialer

The Phone Dialer is a very simple, yet powerful memory resident desktop accessory. However, the Dialer is limited in its capability due to the amount of free memory available in the Pinpoint dispatcher.

The disk-based dialer referred to as "MegaDialer" is a disk-based version of the memory resident dialer. The source code to this dialer is included with the Pinpoint Toolkit. This allows you to customize your dialer to your specific needs.

MegaDialer will act just like the memory-resident dialer, except that it is capable of using a 32-byte lead dial string rather than only 15 characters. You can configure it by choosing the "Configure MegaDialer" option from the application list in the RunRun window.

For details regarding the use of the Dialer, refer to the *Pinpoint AppleWorks Desktop Accessories User Guide.*

Chapter 5: Tools of the Trade

About this Chapter

This chapter discusses the Accessory Mover, the Hexadecimal Calculator, and the Memory Window. Each desktop accessory can be very valuable when working within the Pinpoint/RunRun environment.

The Accessory Mover

The Accessory Mover is a tool for the programmer to define a new desktop accessory. This is primarily a tool for the programmer rather than the end-user because the various parameters must be entered in hexadecimal values. Consumer desktop accessories will be automatically added to the Pinpoint Installation disk via custom procedures, such as those found in the Pinpoint Spelling Checker.

The Pinpoint Desktop Manager is driven by a file found on the Installation disk (or a directory) called "PINPOINTPROFILE." This file contains a list of desktop accessories, their corresponding pathnames, the printer and interface parameters, the dialer attributes, and flags to determine which desktop accessories should be automatically loaded into RAM if it is available. This list is copied into a ProDOS system file containing Pinpoint during the installation process. The file is 768 bytes in length, and can be located $0D00 bytes into the start of the system file containing Pinpoint. The actual layout of this file is discussed in detail at the end of this section.

The Accessory Mover manipulates the file "PINPOINTPROFILE." It does not actually move any accessories about. New desktop accessories can be made available to applications by changing the contents of this file, and re-installing Pinpoint onto the system file.

It is important to note that the Installation program must be run after any modifications to PINPOINTPROFILE have been made in order to take advantage of those changes.

Opening the Accessory Mover

Open the Pinpoint menu.	Press ⌘-P for Pinpoint. If you are using RunRun select the Accessory window.
Choose Accessory Mover from the Pinpoint menu or RunRun.	Highlight it and press RETURN.

The Accessory Mover will be activated. The Accessory Mover must locate the file "PINPOINTPROFILE" in order to operate. The first thing the Accessory Mover will do is to prompt for this file to be available on a mounted ProDOS volume.

The file "PINPOINTPROFILE" was originally distributed on the installation side of the Pinpoint Desktop Manager disk, "/PP.INSTALL." If you have stored this file in a subdirectory somewhere on a hard disk, then press ⌘-P to specify the subdirectory pathname.

A small menu window will appear in the upper left corner. Five choices are available — add, delete, edit, copy or quit.

Adding a New Desktop Accessory

In order to add a desktop accessory, highlight the option "Add an Accessory" and press RETURN. If 16 desktop accessories are already defined, the Accessory Mover will not allow any more accessories to be added. A larger window will appear, and a flashing cursor will be visible in the lower left corner. There are four fields you must enter information into. The four fields are the accessory name, filename, load size, and memory requirements. Figure 5.1 illustrates this.

The *accessory name* is simply a descriptive name assigned to the accessory. When the Pinpoint Desktop Manager is invoked via ⌘-P (or from RunRun), this name appears in the window. It is limited to 16 characters in size, and the first character should be an uppercase letter, but may be a symbol such as an asterisk or dollar sign.

The accessory *filename* should contain the name of the system file that contains the desktop accessory. This file must be a "SYS" file. The filename should be specified here, not the entire

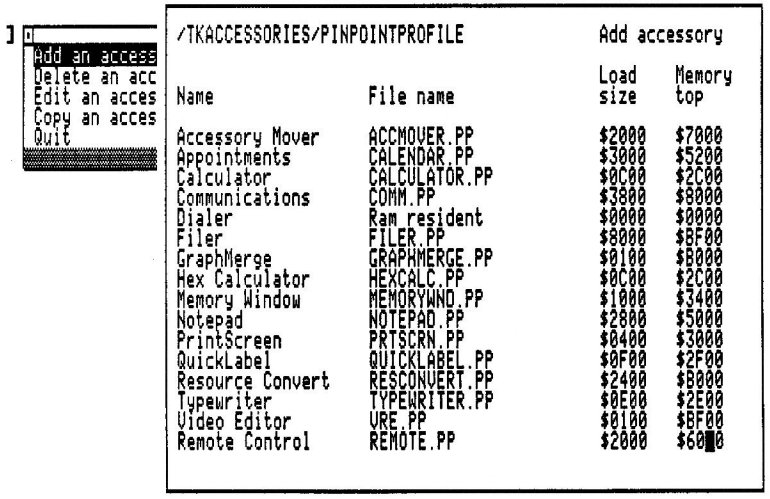
path. It must reside (a) in the root directory of a volume online, or (b) in the standard location defined for desktop accessories.

The *load size* should contain the number of bytes (in hexadecimal) that the dispatcher should load to startup the accessory. If the entire accessory should be loaded, place a high value on the load size, such as $9000. This value is flexible because dynamic "booting" of accessories may be preferred. A $100 byte "boot" portion of the desktop accessory may be loaded. It would then determine which hardware is running, what hardware specific code should be loaded, and what to do next.

The *memory top* defines the highest portion of memory that a desktop accessory will use. It should be slightly high in value if there are any doubts on the memory requirements. A portion of memory from $2000 to top of memory is saved to a RAM disk or drive in file "W.TMP". This file contains the memory contents of the primary application in "sleep state." This number cannot be less than the load size, and cannot be greater than $BF00, since ProDOS resides there.

Once you have entered these four fields, the Accessory Mover menu will appear again. Quitting causes the file "PINPOINTPROFILE" to be updated with the new accessory definition. The Pinpoint Install program (or the Toolkit Advanced Install program) must be run to re-install Pinpoint on a primary application.

Figure 5.1: Adding a Desktop Accessory



Deleting a Desktop Accessory

To remove a desktop accessory from the list, select the "Delete an Accessory" option. Then highlight the accessory to be removed, and press RETURN. If you choose to delete memory resident accessories, you will not be able to reinstall them. The Accessory Mover only allows the installation of disk-resident accessories.

Editing an Accessory

To edit an accessory, select the "Edit an Accessory" option from the Accessory Mover menu. A list of desktop accessories will appear. Highlight the desired accessory using the UP or DOWN arrows. Press RETURN to edit the accessory definition. The accessory name must be unique.

Copying an Accessory

You can move accessory definitions about from one copy of the file "PINPOINTPROFILE" to another using the copy option. You will have to supply two versions of the PINPOINTPROFILE file to do this.

Select the "Copy an Accessory" option from the Accessory Mover window to begin a search for the other version of PINPOINTPROFILE. A dialog box may prompt you for direction to this file. Once located, two windows will appear on the screen labeled Source and Destination. The Source is where you will be copying accessory definitions from. The Destination is the receiving party where new accessory definitions will be stored. By pressing the LEFT or RIGHT cursor arrows, you can change the direction of the source/destination relationship.

To move an accessory definition from the source to the destination press the RETURN key. The destination file must contain less than 16 desktop accessories in order for the copy to occur correctly.

Quitting the Accessory Mover

Select the "Quit" option from the Accessory Mover to cause the file "PINPOINTPROFILE" to be updated with the desired changes. You will have to run the Pinpoint Installation program to re-install the accessories onto the application.

The Layout of the File "PINPOINTPROFILE"

The file "PINPOINTPROFILE" is 768 bytes long. It contains the descriptive names of the desktop accessories and their corresponding ProDOS pathnames. It also contains information about the particular printer and interface type, the load size and memory top for each accessory, the standard location for the desktop accessories, and whether they should be autocopied to a RAM device at startup.

A copy of the file is stored in each system file containing the Pinpoint desktop accessories. This copy is stored at $0D00 into the system file. For instance, if you "BLOADED" the AppleWorks startup file "APLWORKS.SYSTEM" at $2000 in memory, you would see the names of the desktop accessories installed at $2D00 in memory.

When the Pinpoint dispatcher gains control of the machine, and renders it to a desktop accessory, the PINPOINTPROFILE contents are memory resident at $0E00 thru $10FF in memory. They are maintained there so that each accessory can access the common information.

The actual memory layout for the file is as follows:

$0000-$00FF	Descriptive name of each desktop accessory high bit is turned on. Each accessory name is 16 bytes in length. If the initial byte is blank (value $A0), it is not considered an accessory.
$0100-$0280	16 24-byte paths — Each path is preceded by a length byte, followed by the actual path name. The high bit of each byte is off. The maximum size for any accessory name is 15 bytes. The remaining bytes are used if a RAM device is present to redirect the accessory to the RAM drive.
$0280-$029F	16 2-byte pairs — Each pair consists of a load size byte followed by a memory-top byte. These correspond to the entries made in the Accessory Mover.

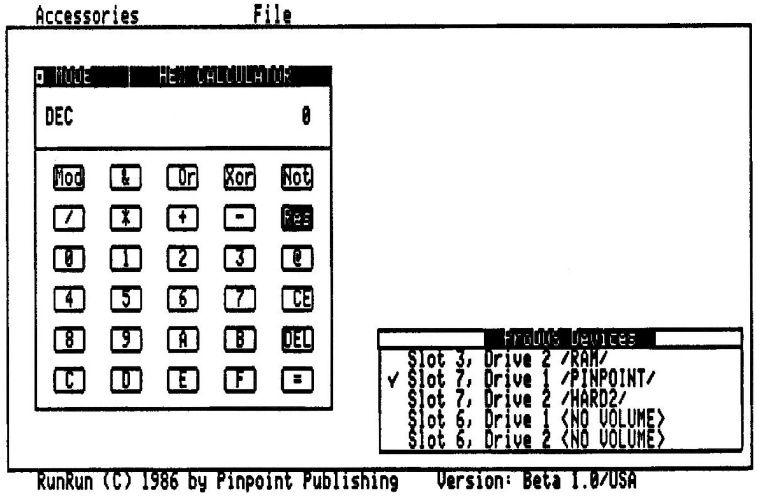
$02A0-$02BF	Standard location of the desktop accessories. The inital byte at $2A0 is an indicator byte. If its value is non-zero, then it contains the slot/drive of the desktop accessories. This byte is stored in the normal ProDOS device format. If this byte does have a value of zero, however, the remaining 31 bytes in the field contain a ProDOS directory that contains the Pinpoint accessories. If the length byte of this path (at $2A1) is null, then no standard location has been set.
$02C0	DESTROYOPT — Flag set to 0 or 1. This flag determines if the temporary file "W.TMP" should be destroyed after exiting Pinpoint. In general, it is set to 0 if a RAM drive is present, and 1 otherwise.
$02C1	PTR.TOF — Multi-purpose flag. The low order bit is set to 0 or 1 depending on whether the printer recognizes a top-of-form character or not. The higher-order portion of this byte stores the number of the interface card selected in the installation.
$02C2	PTR.SLOT — The printer slot selected during the installation. The printer slot number is stored in the lower nibble of this byte. The upper nibble contains a value of "C" If the printer slot is 1, the value of this field would be "C1".
$02C3	PTR.TYPE — Selected printer type. Value corresponds to installation choice.
$02C4-$02CC	PTR.LEAD — Lead string sent to printer upon initialization. The string is terminated by a $FF.
$02CD-$02CE	PTR.TRAIL — Two byte line terminating string. It contains a <$0D> and, optionally, a line feed <$0A>. If the second byte has a value of 0, then no line feed is required.
$02CF	LOCAL.1 — Set to 1 or 0. This is a flag to determine if a "1" should be dialed in advance of a phone number within area code.
$02D0	DIAL.SLOT — Slot number of modem for dialing and communications. The slot number is stored in the lower order nibble of the byte; and higher order nibble contains a "C".
$02D1-$02D3	AREACODE — Local area code. 3-byte value with high order bit turned on.
$02D4-$02E2	DIAL.LEAD — Lead string sent to modem prior to phone dialing. This string has the high order bits turned on, and usually contains an access code for MCI™ or SPRINT™ services.
$02E3	DIAL.LEN — Minimum phone number length. This value is for the Dialer. No groups of numbers with less than this value will be considered a phone number for dialing.
$02E4-$02E5	DIAL.TRL — Optional accounting codes that are appended onto the phone number just dialed. These are accounting codes, stored with the high order bit on.

$02E6	DIALS16 — Dialer slot multiplied by 16.
$02E7-02EF	DIAL.CMD — Dial command used for Hayes AT compatible modems. High order bit is on. The typical command is "ATS7=30DT". A phone number would then follow that.
$02F0-$02FF	16 1-byte flags — These determine if the corresponding accessory should be autocopied to the RAM device at startup. A value of 0 means that they should not.

The Hex Calculator

The simple calculator found in the Pinpoint desktop accessories is very handy for floating point mathmatics. It does not, however, allow computation in *hexadecimal* (base 16) or *binary* (base 2). These two bases are very useful for programmers.

Figure 5.2: The Hexadecimal Calculator



Opening the Calculator

To open the Hex Calculator:

Open the Pinpoint menu.	Press ⌘-P for Pinpoint.
Choose the Hex Calculator.	Highlight it and press RETURN.

The Calculator will appear with a zero in the display and the abbreviation "DEC" under the word MODE. This means that the Calculator is accepting numbers and providing results in *decimal* (base 10).

Changing Modes

Pressing the mode key (M) to change the display to hexadecimal (base 16), binary (base 2), or decimal (base 10).

Entering Numbers

Type the number using the number keys.

To perform a numeric or Boolean operation, press the appropriate key. The following keys operate the Hex Calculator:

0-9, A-F	Numeric values for input — In binary mode, only 0 or 1 are allowed. In decimal mode, the Calculator only allows decimal numbers 0 through 9. The largest acceptable value is 65535 decimal or $FFFF hex.
/ * + -	Numeric operators — The Calculator is an integer calculator, so results of a division may appear misleading.

& O X N	Boolean operators are as follows: & AND O OR X EXCLUSIVE OR N NOT
M	Mode switch — Press this key to switch between decimal, hexadecimal, and binary input and results. The mode switch provides a handy way to convert numbers between base 2, 10, and 16.
=	Results key — Displays the results of a calculation.
R	Reset key — Clears the contents of the Calculator.
Delete	Delete key — Deletes the last digit in the number.
⌫-C	Clear entry key — This clears the most recent entry to the Calculator.
P	Print key — Toggle switch that enables or disables printed output. If the printer is turned on, then a paper tape of the computations will be generated.
@	Indirection key — This displays the contents of the memory address.
ESCAPE	Exit key — This causes the Calculator to disappear. Control is returned back to the primary application.

Quitting the Hex Calculator:

Press ESCAPE.

The Memory Window

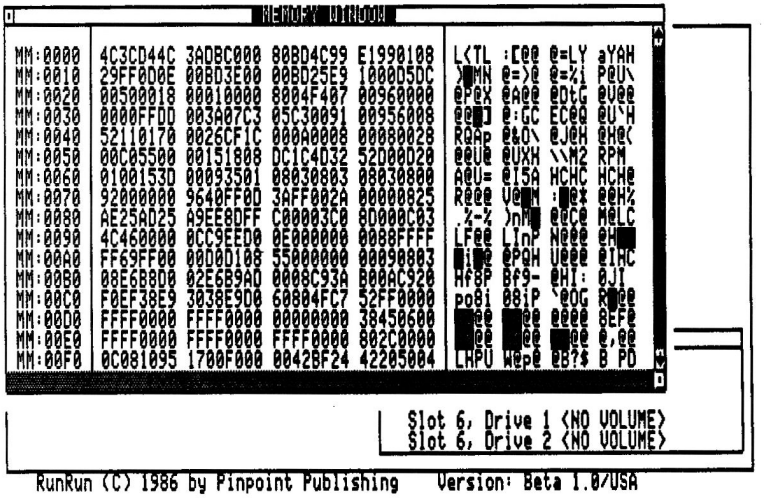
The Memory Window desktop accessory is a handy debugging tool for programming. It allows you to view the current contents of memory.

To invoke the Memory Window accessory:

Open the Pinpoint menu.	Press ⌘-P for Pinpoint. If you are using RunRun, select the accessory window.
Choose Memory Window from the Pinpoint menu or RunRun.	Highlight it and press RETURN.

The Memory Window will appear on the screen, as shown in Figure 5.3.

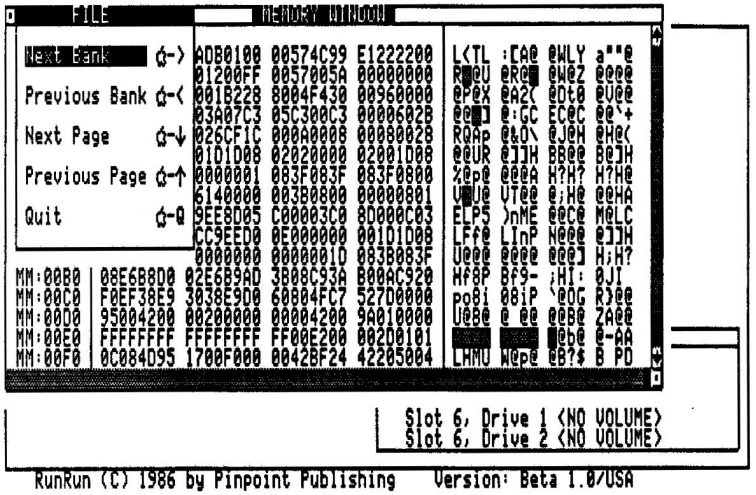
Figure 5.3: The Memory Window



The File Menu

Press the ESCAPE key to bring the File menu onto the upper left corner of the Memory Window. There are five commands available from the File menu. Figure 5.4 illustrates this.

Figure 5.4: The Memory Window File Menu



Switching Between Memory Pages

You can move between memory pages by using the ⌘-UP arrow and ⌘-DOWN arrow keys to position about in memory. A memory page consists of 256 bytes. Viewing memory from $C000 through $CFFF is not allowed. Viewing memory above $D000 looks at the ROM memory, as opposed to bank switched memory.

Switching Between Memory Banks

If your Apple is equipped with bank switched memory (as in the Apple Extended 80 Column Card™ or the Applied Engineering RamWorks™ card), you can move about through memory banks viewing the contents of memory throughout each bank. By pressing the ⌘-> and ⌘-< keys, you can move about through memory.

The bank number appears along the left side of the window. The bank number will appear as "MM" by default, for main memory bank. If you press the ⌘-> key, the bank number will read "00" for the initial auxiliary memory bank.

The Memory Window is Clouded

The Memory Window is a desktop accessory. When it is running, it occupies memory (from $2000 thru $2FFF). In addition, Pinpoint is resident in memory at the time from $0C00 through $1FFF. This means that you cannot view the primary application in that memory region. The source code is included for this accessory, so you may opt to change this accessory to meet your particular needs.

Exiting the Memory Window

Select the "Quit" option from the File menu to exit the Memory Window. Optionally, you can press ⌘-Q to quit.

Chapter 6: Designer Tools

About this Chapter

Items covered in this chapter include the Video Resource Editor and the Resc
instruments used by the designer to create, test, and implement visual displa

The Video Resource Editor

The Video Resource Editor (VRE) transforms the text screen to a sketch pad. Text and icon characters are the drawing medium. An internal script language allows simulating an actual visual interface. The VRE outputs a binary video resource file which you can convert to various source codes with the Resource Converter.

Format of a Video Resource

A *video resource* is a binary image of the particular window. It begins with a 6-byte header followed by a compressed image of the video refresh clip region. The header begins with the upper left row (0-23), then the upper left column (0-79). The third byte contains the width of the window; and the fourth byte contains the length of the clip region. The fifth and sixth bytes contain other information for the Pinpoint dispatcher.

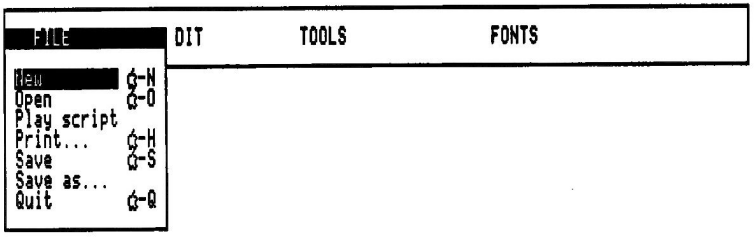
If you wish to position a window to another location other than its default location, simply change the contents of the first two bytes of the resource to another value. The value must be valid, and not position the window off the screen.

Opening the VRE:

Open the Pinpoint Menu.	Press ⌘-P for Pinpoint. If you are using RunRun, select the accessory window.
Choose Video Editor from the Pinpoint menu or RunRun.	Highlight it and press RETURN.

The VRE will be activated, clearing the screen and showing its command bar. Arrow keys will scroll through the four command windows: *file, edit, tools,* and *fonts*. Use the ESCAPE key to toggle displaying the command bar. The entire screen is your visual workspace.

Figure 6.1: The VRE Command Bar



Starting Fresh

When you enter the VRE, your workspace is initialized. The workspace area is expanded to the text screen's boundary and cleared. You can begin "drawing" on the screen.

To reinitialize the VRE, select "New" from the File window. This clears the screen, resets the video workspace, and deselects the command bar.

Picking a Font

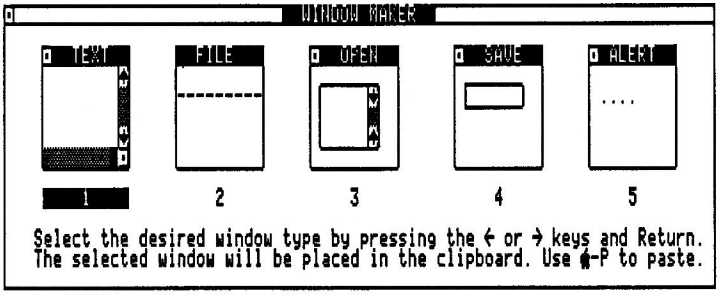
Type directly from the keyboard to "draw" on the workspace. The VRE does not limit itself to normal video typewriter characters. You can use different fonts, listed in the Fonts window, to create images. "Normal" displays white characters on a black background while "Inverse" does just the opposite.

"Icons" allows you to type with the mousetext icon characters. Select "Icon Table" from Tools to access a character reference. It will display mousetext icons and their keyboard equivalents over the workspace. Press any key to close the character reference window. Note that keys not listed in the icon table will produce inverse characters with the icons font selected.

The Window Maker

The "Window Maker" contains a library of window types. Upon selection, it presents you with a choice of windows. Choose the desired window using the LEFT and RIGHT arrow keys. Press RETURN and the selected window will be copied to the clipboard for pasting. All the windows are pre-sized. Select "Show Clipboard" to see the actual window. Or press ESCAPE to abort.

Figure 6.2: Using Window Maker



Copying or Moving to the Clipboard

The clipboard helps make editing simple. "Copy" and "move" to clipboard are similar commands. They transfer specified portions of the workspace to the clipboard for temporary storage. *Copy* makes a duplicate image on the clipboard without altering the workspace, whereas *move* will actually take a piece from your work area and place it on the clipboard.

When you choose to copy or move, crosshairs appear on the workspace. To anchor a corner of the "capture" box, position the crosshairs with the arrow keys and press RETURN. Now the resulting box can be sized with the arrow keys. Press RETURN to complete the copy or move, or hit ESCAPE to return to the crosshairs.

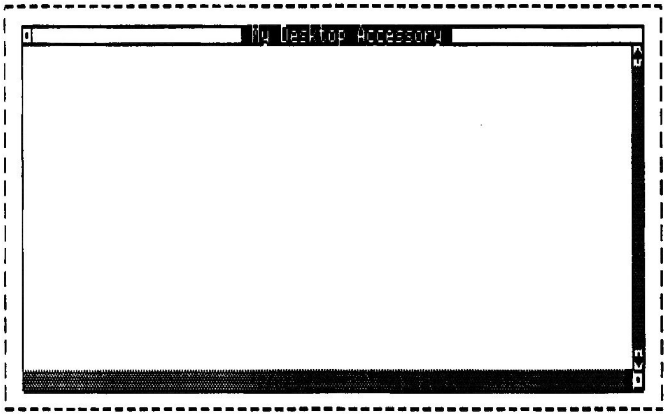
A Look at the Clipboard

To display the clipboard, select "Show Clipboard." The clipboard window will appear over the work area. Type any key to close the window. Only one item may be on the clipboard at a time. Any copy, move, or use of "Window Maker" will destroy its previous contents.

Pasting from the Clipboard

Choose "Paste Clipboard" to copy the image from the clipboard to the video workspace. The paste will occur at the current cursor position as long as the pasted image fits within the screen boundaries. The VRE produces an error bell for invalid pasting locations. Moving the cursor up and/or left should correct the situation. "Undo Paste" will remove the last pasted image from the workspace.

Figure 6.3: Copying to the Clipboard



Locking a Region

You can resize the video work area with the "Lock Region" command. Sizing the region is similar to the copy/move technique. When an area has just been locked, the screen will seem unaffected, but you will discover that the cursor can be positioned only within the specified region. To resize the workspace again, just use the same command.

Opening a Resource File

The File command "Open" reads a video resource file from a ProDOS device and displays it on the workspace. The working area will be sized to the resource's saved dimensions.

When you select "Open," a dialog window opens onscreen. This lists all possible resource and directory files in an online ProDOS volume. To choose a file, highlight the filename and press RETURN. Or, hit ESCAPE to cancel or ☆-N to search through another volume.

Saving a Resource File

To save a resource, pick "Save" or "Save As..." The locked region from the workspace mode is placed in a binary file which can later be converted to compatible code for various compilers and assemblers.

If the you want to save a resource for the first time or under a different name, use "Save As..." Type a name and hit RETURN to save the locked work area to a binary file on the current prefix. Change to a different volume by pressing ⌘-N. ESCAPE will cancel the save.

Selecting "Save" will save the current resource on the workspace to the last pathname used by the "Open" or "Save As..." commands.

Printing the Screen

Select "Print..." to print the work screen to a printer. The ImageWriter™ printers will support the icon characters. If working within the RunRun environment, you can print icons and text on other printers using the PrintScreen accessory.

The Script Processing Language

The VRE contains a valuable prototyping tool, a *script* processor. Scripts are actual programs, using correct script syntax, stored as standard ProDOS textfiles. Scripts may be composed with most ProDOS text editors. The script processing language was designed to be a subset of the 'C' language standard, with certain anomalies imposed by the restricted environment of the accessory. The following is a list of the valid keywords and syntax examples.

Only 16 program variables are allowed, designated by **%0** through **%15**. Parameters may not be passed to subroutines. The multiplication and division operators have not been implemented.

Keywords may be in either case but are converted internally to upper case. Strings for the Say command may be in upper and/or lower case and will not be converted. All keywords must be surrounded by white space; for example:

```
if ( %3 == 5 ) exit  is valid, whereas
if (%3==5)exit       is not.
```

The default radix is decimal. ASCII characters may be specified by preceding them with a single quote ['], hexadecimal digits by a preceding 0x.

```
'H          is an ASCII H.
'h          is an ASCII h.
'^H         is hex $08 or an ASCII control-H.
0x1234      is a hex $1234.
0xff        is hex $FF.
```

Pointers have been implemented for peeks and pokes; they are specified by a preceding asterisk [*]. To place the value of location $ABCD into variable 3, as an example, use:

```
%3 = *0xabcd
```

To poke the contents of variable 4 into location $1234, use:

```
*0x1234 = %4
```

Labels must begin with an alphabetic character and be at the start of a line (column 1). Only the first 32 characters are recognized. A colon [:] may optionally end a label.

Comments are in standard 'C' form; they are set apart by a leading /* and a trailing */. Comments cannot be nested.

```
/* This as an example comment. */
```

The following shows valid keywords and syntax use:

prefix **prefix** pathname
Sets new default prefix for project and exec commands. The default prefix on startup is the pathname containing the script file.

project **project** *pathname*
project *pathname @ x y*
Loads the resource file and projects it on the screen via a WWOMP Pinpoint call. The x and y coordinates are optional and must be within the ranges: x(0-79) and y(0-23). Finding no specific coordinates, the script processor will project at the default coordinates coded in the resource file.

key **key**
Gets a keystroke from the keyboard. The key value may be stored in a variable or a memory location with assignment such as:

```
%1 = key or *0x1234 = key
```

echo **echo**
echo *argument*
Controls whether keystrokes are echoed to the screen. Upon startup, keystrokes are not displayed onscreen. The *argument* can be **on** or **off**. Echo with no argument defaults to on.

if **if** (*condition*) **statement**
Evaluates the following conditional and executes the next statement if the conditional evaluates to true. The conditional itself must be surrounded by parentheses. Parentheses may be nested up to 255 levels. Valid 'C' assignment syntax is supported. An example of a valid command line:

```
if ( ( %1 = key ) == 'H ) home
```

else **else** *statement*
Executes the next statement if the result of the preceding conditional is evaluated to false. Toggles the conditional result so that in the following case:

if (*condition*) *statement1*
else *statement2*
else *statement3*
else *statement4*

If *condition* evaluates to true then *statement1* and *statement3* will be performed. If *condition* is false, *statement2* and *statement4* will be done instead.

while **while** (*condition*)
{
statement
}
Repeatedly processes a statement or collection of statements until condition is false.

home **home**
home *left top right bottom*
Clears an area of the screen video. Optional parameters must be in valid ranges: left and right(0-79), and top and bottom(0-23). Default arguments are 0 0 79 23.

home 10 10
will clear the window 10 10 79 23. Home with no arguments clears the entire screen.

gosub **gosub** *label*
Executes a named subroutine and returns to process the next line of code. Subroutines are of the form:

```
label:
{
        statement1
        statement2
}
```

{ }
The brace characters serve two purposes: The first is to delimit the contents of a subroutine, in which case the right brace [}] functions as a return-from-subroutine. Their second purpose is to group statements together for conditional execution after an **if** or **else** command. For example:

```
if ( condition )
{
        statement1
        statement2
}
else
{
        statement3
        statement4
}
```

goto **goto** label
Unconditionally jumps to *label* to continue processing code.

exec **exec** *pathname*
Begins executing another script file. Returns to caller if script file cannot be found.

bell **bell**
Generates a beep from the speaker.

say **say** " *string* "
Displays a string or the contents of a variable on the video screen. Strings are composed of any valid ASCII characters.

%
The percent sign [%] is a special designator within strings used to show the value of a variable. To display the percent sign, just double it:

```
say "The variable %%4 = %4."
```

Assuming %4 contains 1234, the above statement will produce:

```
The variable %4 = 1234.
```

The caret [^] is another special designator used to flag control characters. It too can be displayed by doubling the character.

locate **locate** *x y*
Sets the cursor position in preparation for a Say command. Coordinates x and y must be in valid screen ranges. Default x and y are both zero.

sleep **sleep n**
Goes to sleep for *n* units of time, then continues processing. Each unit of time is approximately forty milliseconds, so **sleep 25** will delay for about one second.

exit **exit**
Ends script processing and returns to VRE edit mode.

Relationals

The following relational operators are supported:

==	equal to
!=	not equal to
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
&&	logical and
||	logical or

Operators

The following operators are supported:

=	assignment
!	negation
+	addition
-	subtraction
&	bitwise and
|	bitwise or
^	bitwise exclusive-or
+=	additive assignment
-=	subtractive assignment
&=	logical and assignment
|=	logical or assignment
^=	logical exclusive-or assignment

In addition, the post-increment and post-decrement operators are also supported.

```
%n ++ is the same as %n += 1
%n -- is the same as %n -= 1
```

Remember to adhere to the correct syntax as the script processing language does minimal error-checking. A sample script file is included on the Toolkit disk.

Quick-Keys

Most of the VRE commands have quick-key equivalents. The ♁ keystrokes are listed in the various command windows. For example: to refer quickly to the icon table, type ♁-I and the table will appear instantly.

Error Codes

The VRE will output ProDOS code numbers in event of ProDOS errors. Refer to the *ProDOS Technical Reference Manual* for the specific code translations.

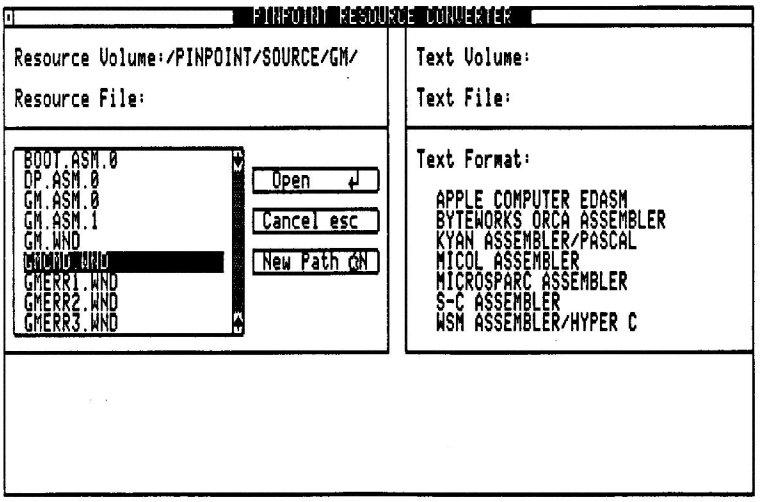
Leaving the VRE

Select "Quit" from the File window or type ♁-Q. Pinpoint will return to the previous application.

The Resource Converter

The Resource Converter, as its name implies, converts binary resource files created with the Video Resource Editor to specific source code formats.

Figure 6.4: The Resource Converter



Opening the Resource Converter:

Open the Pinpoint menu. Press ⌘-P for Pinpoint.
If using RunRun, select the Accessory window.

Choose Resource Converter from the Pinpoint menu or RunRun. Highlight it and press RETURN.

The Pinpoint Resource Converter window will appear onscreen. The converter program searches the online ProDOS devices for possible video resource files.

Converting a Resource File

File choices are presented for selection. Highlight one and press RETURN to choose a resource file or open a directory file. To search a different volume, type ⌘-N. The current prefix for the file is shown at the upper-left hand corner of the screen.

Selecting a Text Format

After a resource file has been selected, Resource Converter asks for a filename to write the newly converted code to. "Text Volume:" shows the current prefix for the code file; pressing ⌘-N will find another volume. Type a name for the file and press RETURN, or hit ESCAPE to go back and choose another resource file.

The Resource Converter needs to know what code format to write. Highlight the appropriate compiler or assembler format and use RETURN to complete the conversion, or ESCAPE to cancel.

Quitting the Resource Converter

Press ESCAPE to return to the last application.

Chapter 7: Writing Desktop Accessories

About this Chapter

This chapter discusses the process of building desktop accessories from a programming language perspective. Building desktop accessories in PASCAL, BASIC, and Assembly are discussed. This chapter is of interest to the programming audience.

What are Desktop Accessories?

Desktop accessories are ProDOS system files that adhere to a few constraints. Pinpoint was designed from the start to accommodate a variety of accessories. The ProDOS Filer included in the Toolkit is a good example of how capable the Pinpoint dispatcher is at managing the resources of the Apple II.

Memory Usage

A desktop accessory should occupy the main memory of the Apple II, from address $2000 up. If the desktop accessory requires memory below $2000, then it should take steps to save this portion of memory prior to its use, and recover that memory before exiting. A desktop accessory should not use memory above its defined top-of-memory.

If you are writing a desktop accessory in KYAN PASCAL or MICOL BASIC, you should define the top-of-memory as $BF00. Both compilers require that a runtime library be loaded into high memory to function properly. Pinpoint will save the area of memory from $2000 through $BEFF away to disk and recover it upon exit.

Language Choices

Currently desktop accessories can be written in three languages — Assembly, PASCAL, or BASIC. Assembly language provides the greatest access to the Pinpoint dispatcher library calls, and generates the most efficient code. Assembly language is, however, non-trivial and very machine specific. It does not offer you the flexibility of manipulating floating point (or 16-bit integer) mathematics very easily. PASCAL and BASIC are high level languages that allow for rapid development of personal desktop accessories. The two compilers supported are the KYAN PASCAL compiler *(Kyan Software, 1850 Union Street #183, San Francisco, CA 94123 / 415-626-2086)* and the MICOL BASIC compiler *(Micol Systems, 9 Lynch Road, Toronto, Ontario, Canada M2J 2V6 / 416-495-6864).*

A viable option for developing commercial desktop accessories is to prototype the initial user interface first using the script language of the VRE facility. You could then build a working version in PASCAL or BASIC. A final step may be to rewrite the program in Assembly language for the final distribution of the product.

Using KYAN PASCAL

KYAN PASCAL is the core of a powerful software development system. It conforms to the ISO PASCAL standard, level 0. It can be used in conjunction with the Pinpoint Toolkit to generate desktop accessories. It is intended primarily to generate primary applications.

The primary disadvantage to using the KYAN PASCAL system for generating desktop accessories is the size of the runtime library. It is approximately 12K in size. Given a RAM drive, 12K is not a very big problem. Another constraint of using PASCAL (or BASIC) is that you cannot access the printer, graphic, or font libraries because they reside in the same memory locations as the Runtime Library.

PASCAL Source Code Files

On this source code disk there are three files related to KYAN PASCAL — STDLIB.S, PINPOINT.LIB, and PCPEXAMPLE. The file "STDLIB.S" is a replacement for the standard library found with the KYAN PASCAL disk. It contains convenient initialization and quit code for PASCAL accessories. The file "PINPOINT.LIB" contains several PASCAL procedures for connecting to various Pinpoint dispatcher call functions.

The file "PCPEXAMPLE" is a brief PASCAL desktop accessory that sets various printer attributes for an Epson RX printer. It serves as a good example of a PASCAL desktop accessory. It can be customized for your particular printer.

Starting Up a PASCAL Accessory

When a PASCAL desktop accessory starts up, it first attempts to locate the runtime library so that it can load it into memory in location $9000-$BEFF. The ProDOS directory, where the accessory is located, is initially searched for the runtime library. If it is not located there, the RAM drive is searched for it. The file STDLIB.S contains the Assembly language source code for this search logic. It can be modified to suit your particular needs. You can contact KYAN Software about locating the runtime library on the end of your code as well.

Using Video Resources with KYAN PASCAL

The Resource Converter described in Chapter 6 allows you to take the video resources created by VRE and convert them to KYAN Assembly code. The resource converter generates a PASCAL function that returns a pointer to the actual resource. As an example, if you wished to include a resource called "MainMenu", you would have entered PASCAL code as follows:

```
#i pinpoint.lib
#i MainMenu
...
Project(MainMenu)
```

The project function takes the pointer sent back from the function Main menu and passes it down to an Assembly language call that invokes the Pinpoint dispatcher.

The Pinpoint Flag

By including special code in your source file at the top, you can automatically create a Pinpoint accessory file (system file). In order to do this:

Include the following three lines at the top of your program:

```
#a
_Pinpoint equ 1
#
```

This will create a system file which can only be used as a Pinpoint accessory (after being installed as an accessory).

Video Provisions

When you use the Pinpoint version of STDLIB.S to generate a desktop accessory, it occupies about 2K of overhead in your system file. This area is used to initialize the machine as required for KYAN PASCAL. In addition, a copy of the current 80 column video screen is automatically saved away. Upon exiting the PASCAL desktop accessory, the screen is automatically recovered. There is no real need to capture the video and recover it using KYAN PASCAL or MICOL BASIC. Assembly language accessories must do this on their own, however.

Using MICOL BASIC

You can write desktop accessories in BASIC provided that you use the MICOL BASIC compiler to convert your BASIC program into a binary object file, and that it is linked together with some initial Assembly code to form a ProDOS system file. Desktop accessories built in BASIC do not have access to the Pinpoint dispatcher, however.

MICOL BASIC differs syntactically from standard ProDOS BASIC in disk I/O activity as well as printing activities. The compiler supports a structured form of programming, so programs tend to be more readable.

Runtime Library

MICOL BASIC is similar to KYAN PASCAL in that it needs a runtime library to carry out many activities. The library is called "LIB" and can be found on your MICOL compiler disk. The library file should reside in the same directory as the desktop accessory. It is loaded by the initial header code residing at $2000 in your accessory system file.

Setting up an Accessory

Construct an accessory by first writing your program in BASIC and getting it to work as desired. Next you should convert it where necessary to MICOL BASIC standards.

The second line in your program should read as follows:

```
20 @LOMEM=8960, HIMEM=31999
```

This sets the memory limitations on the program. The bottom of memory is set at $2300. This area is used to store the memory in the $300 and $800 pages of primary memory.

Once your program compiles correctly, it will result in a binary file. In order to create and save a desktop accessory, carry out the following actions:

```
Bload header,a$2000              Loads in the Pinpoint Header
Bload accessory.bin,a$2300       Loads in object code of accessory, can be any name

Create accname.pp,tsys           Creates an accessory file
Bsave accname.pp,tsys,a$2000,l$xxxx
```

The value of $xxxx above must be determined by you.

After this is complete, use the Accessory Mover to make an entry for your desktop accessory. The load size specified should be large enough to load the entire system file into memory. The memory top should be set to $BF00.

A version of the Printer Control accessory is included on the source code disk under the BASIC directory. It is titled "PCPEXAMPLE". The header code for MICOL accessories is found under the file "HEADER.ASM".

Using Assembly Language

Assembly language is the most optimal language in which to write desktop accessories. This is due to the fact that Assembly language generates very small, precise code. Being small, the desktop accessory does not require much memory; therefore it loads quickly.

The examples of desktop accessories included in the Toolkit are carried out in Assembly language. The ProDOS Tools/EDASM assembler was used.

Unlike KYAN PASCAL or MICOL BASIC, however, no provisions are made for automatically saving the video screen upon entry and recovering it upon exiting. This is the responsibility of the Assembly language desktop accessory. This provides maximum flexibility and minimum space for the accessory.

Using Assembly language, you have access to all of the functions found within the Pinpoint dispatcher. These functions are totally optional, and do not have to be used at all. They serve more as a convenience than anything else.

Using Assembly Resources

The Resource Converter provides maximum flexibility for Assembly output. You can convert your binary resources directly to Assembly output for use with your code. If you do not see your assembler listed, use an assembler with a syntax that is very similar and convert the output as necessary.

Pinpoint Function Library

The function numbers that correspond to each dispatcher call are stored in a library called "PINPOINT.H". This file contains a series of equate statements that establish the values for each function call. For instance, to project an resource to the video screen, you could issue the code:

```
LDX #PROJECT
JSR DISP.SVC
```

The "Pinpoint.H" library is used in the source code examples of desktop accessories.

Chapter 8: Advanced Installation

The Installation Facility

The Advanced Installation Facility found on the Toolkit disk is slightly different than the one distributed with the Pinpoint desktop accessories. It allows you to install the Pinpoint dispatcher on your personal startup system file. When your primary application is started up again, Pinpoint will be loaded in the upper 12K of auxiliary memory, and the dispatcher "polling" code will be located in the $300 page.

The Advanced Installation Facility will not make programs simply "recognize" Pinpoint desktop accessories and avail them for use. This is a remarkably difficult task, given the nature of the Apple II, and the non-standard methods of monitoring keyboard events. The Pinpoint dispatcher will be installed in advance of the primary application. It is, however, the responsibility of the primary application to detect the presence of Pinpoint, and act accordingly.

File Modifications

Pinpoint is placed at the beginning of the system file that it is installed on. Figure 8.1 below illustrates this point.

Figure 8.1: Enhanced Startup File

$0000 $34FF	Pinpoint Dispatcher and Preboot
$3500 EOF	Original System File Contents

The Pinpoint preboot logic first detects whether an Apple Memory Expansion Card™ is installed in the machine (or if a software RAM drive is to emulate one), and copies desired desktop accessories up to the RAM drive if it is there.

It then relocates the Pinpoint dispatcher to the upper portion of auxiliary memory ($D000-$FFFD) as well as the $300 page of main memory. Once it has completed relocating itself, it relocates the original contents of the system file to $2000. In a standard application such as AppleWorks, the code is then analyzed for keyboard polling, and modifications to the code occur on the fly. Control

is then passed to the application. The primary application really had no idea of the previous events, and goes about its startup logic.

Using the Advanced Installation Facility

The Advanced Installation Facility can be run from the RunRun application list by selecting "Pinpoint Installation". If you have not modified your program path, it will prompt for the Pinpoint disk, as it contains the install program.

Optionally, the program can be run from BASIC by first setting the prefix to the disk (or ProDOS subdirectory) that contains the installation software, and typing:

```
-AINSTALL.SYSTEM
```

Just Like Pinpoint

This manual assumes you are familiar with the Pinpoint desktop accessories, and have used the installation facility found with Pinpoint in the past. This manual does not cover the various aspects of the installation set-up such as the modem and printer characteristics, standard location of the desktop accessories, etc.

The Advanced Installation Facility is upwardly compatible with Version 1.3 of the Pinpoint Installation Facility. If you have been using an earlier version of Pinpoint you should update it to Version 1.3 in the future. For now, this version contains specific drivers for numerous printer interfaces and printer.

Installing Pinpoint on a Personal Application

You can install the Pinpoint dispatcher on your personal system startup file providing that your system file and the Pinpoint dispatcher ($3500 bytes long) can fit in memory.

First, you should customize Pinpoint for your computer system. Now you're ready to put it onto your application.In order to install Pinpoint on your system file:

Select the Program Setup option from the menu.	Press 3 at the Pinpoint Installation menu.
Press RETURN.	
Highlight the Personal Application option.	Position to the corresponding option using the arrow keys or the number.
Press RETURN.	
Enter the full pathname of your personal system file.	

The Advanced Install Facility will search for your personal file. If it is located it will be updated to contain the Pinpoint dispatcher and preboot code. If Pinpoint had been installed on your application before, it will simply re-update the Pinpoint logic and leave your program intact.

Recognizing Pinpoint

An application must detect that Pinpoint has been installed in order to avail the desktop accessories. A fairly reliable test is to look at the auxiliary RAM memory locations $D000 and $D002 (bank 0). They should contain a value of $80. In addition, the $300 page should contain the polling logic. In particular, the contents of $32d should equal $AD, an LDA instruction.

A typical test would be as follows:

```
LDA $C083          ;Set up Aux Bank to view
LDA $C083
STA $C009
LDA #$80           ;Test for BRA instructions
CMP $D000
BNE BOGUS
CMP $D002
BNE BOGUS
STA $C008          ;Back to Main Memory
LDA $C08A          ;ROM intact
LDA $32D           ;Poller logic begins with LDA
CMP #$AD
BNE BOGUS
...                ;Pinpoint is installed
```

Providing that such a test was passed, and you are confident that Pinpoint is resident in auxiliary memory, then you should redirect all LDA $C000 logic to $32D. This will return the results of the A-register with the keyboard strobe value. If you hit an ⌘-P key, however, control will be passed to the Pinpoint dispatcher.

Appendix: Other Pinpoint Products

Other Pinpoint Products

Pinpoint™ Desktop Accessories $89.00

A line of desktop accessories that "pop-up" within AppleWorks, Point-to-Point, InfoMerge, and other ProDOS applications.

- **Appointment Calendar** — Print schedules to take with you. MonthView, DayView and Summary displays.
- **Calculator** — Four math functions: +, -, *, /. 16 digit display with scientific notation.
- **Communications** — Instant modem communications from AppleWorks desktop with simplified auto logon macros; no complicated control codes required. Downloads (records) files in AppleWorks' AWP file format.
- **Telephone Dialer** — Finds phone numbers anywhere on *any* screen. No phone directory or databases to re-enter.
- **GraphMerge** — Paste up to 16 single or double hi-res graphic images, mix & match, anywhere in AppleWorks documents.
- **Notepad** — Powerful baby word processor produces AppleWorks files. Use Notepad files inside AppleWorks documents, or with GraphMerge.
- **QuickLabel** — Envelope addresser/labeler, with cut-and-print utility. Position address block on an electronic envelope, then print on a real envelope, label or paper.
- **Typewriter** — Ideal for short letters, forms, file folder labels, and name badges. Compose, edit, then print, just like an IBM Memory Writer™ typewriter.

Pinpoint Spelling Checker™ $69.00

This additional Pinpoint desktop accessory enables you to spell check AppleWorks word processing documents, spreadsheets and database files *without ever leaving AppleWorks.*

- Spell checks words, paragraphs or entire documents of any size in memory.
- Suggests alternative words.
- Corrects and reformats automatically.
- Requires Pinpoint Desktop Software.

Point-to-Point™ $129.00

The #1 communications software program for the Apple II!

- AppleWorks style integrated communications with all the right tools for most any job.
- Exclusive Extended Xmodem Protocol (EXP) with CRC-16 transmits/receives multiple files in batches with original file names, extensions, and file attributes intact. Ready to run, without misspelled file names or tiresome file conversions.
- 14 powerful macros commands automate logon sequences; system operating commands automatically upload and download electronic mail or program files. Pattern recognition feature even makes decisions based on what it "sees" coming across the screen.
- Built-in text editor, or send AWP or text files directly.
- Simultaneous formatted printing while online. Instant "PrintScreen" prints anything and everything displayed on your screen.
- Impressive file handling utilities manage files, subdirectories and paths without typing, even while online!
- Install Pinpoint Desktop Accessories for ever greater versatility, or use RunRun and Toolkit to add valuable desktop accessories anytime.
- Smart utilities strip line feeds from downloaded files, and convert TXT to AWP files; plus 8 indispensable file type conversions save double work!
- Support for dozens of modems, including the newest 2400-baud units, gets your new system working right the first time. Interface cards include old standbys, plus the emerging Apple II interface technologies of tomorrow.
- Written by **Gary B. Little**, an expert Apple II and Macintosh programmer, and author of countless articles for *A+, Call APPLE, Nibble* and five recent Apple books.

ProFILER™ *CALL*

A single disk database manager and report generator. With this *one* program you can easily design, organize, file, search, sort, calculate, merge, and print using floppy or hard disk.

- Get up to 1,500 records per floppy or 65,000 on a hard disk; 250 fields per record!
- Multiple files per disk, multiple index files, full screen editor.
- Pinpoint and AppleWorks compatible.
- Optional data conversion program for PFS users (DOS 3.3 and ProDOS versions).

Pinpoint Apple IIe Upgrade Kit $29.00

- Permits Pinpoint and many 'new' Apple IIe software programs to run on plain Apple IIe with 128K
- Solves many of the compatibility problems older software programs have with the Enhanced Apple IIe
- Includes Mouse Text Video ROM, exact copy of Apple's Character Generator ROM #342-0265 approved by Apple.
- Includes GTE or Rockwell 65C02 microprocessor.
- Does NOT include Apple's proprietary 'CD' and 'EF' ROMs required for most mouse-based software. This Kit does not "Enhance" the Apple IIe. This Kit is only to be used on Apple IIe's.
- This upgrade is performed at user's own risk.
- Handy chip puller included

Pinpoint RAM Enhancement Kit $29.00

- Configures partitioned RAM disks.
- Automatically uploads non copy protected ProDOS applications and files to partitioned RAM disks.
- Menu driven setup. Catalog with toggles displays programs/files to be uploaded.
- Includes a ProDOS RAM switcher for AppleWorks with Pinpoint and other ProDOS applications uploaded into RAM, such as Business Works accounting software, Point-to-Point communications software, or InfoMerge mail merge and print formatting program.
- Requires Pinpoint Desktop Software.

Pinpoint Modem Enhancement Kit $ 29.00

- Advanced modem software enhancement for Pinpoint's Communications window.
- Logon macros with default communications setups.
- Keyboard and video character remapping, simple file encryption program.
- Selective character filtering.
- Required for Novation, Microcom and other non-standard external/internal modems or interface cards.

InfoMerge™ $79.00

On-screen, direct-print, mail-merge and print formatting program that works with AppleWorks. Automatically select names from AppleWorks database, or manually scan files forward/backward, add multiple keyboard inputs (great for invoices or forms), perform dynamic field calculations, preview on screen before printing, or just print everything. One copy each or one hundred or mix & match. Presto!

Optionally, print to disk files for use by other programs, such as telecommunications software like Point-to-Point or Pinpoint Communications Window. Pop-up any Pinpoint Desktop Accessory for added functionality.

Micro Cookbook $49.95

- Big 8th Edition. Over 100,000 copies sold.
- Ask the computer what's for dinner. Enter ingredient, nationality, course or whatever, and Micro Cookbook offers up a menu with delicious possibilities.
- Choose from Micro Cookbook's 150 tested recipes (supplied on the program disk) or add your own.
- Selects recipes from any individual recipe disk or path, or automatically searches up to 8 others — over 4,000 recipes online for instant recall.
- Automatically adjusts recipes for different serving sizes.
- Automatic shopping list preparation.
- Valuable on-line reference: nutrition facts, food selection and storage, and ingredient substitution suggestions.
- Lightning fast operation with unparalleled ease of use.
- Fourth-generation, bit-mapped database design finds plurals, truncations, multi-variant relations (7 simultaneous index keys), overlooks request errors, and generally works hard so you don't have to.
- Versions available for Apple II+ (64K DOS 3.3), Apple IIc/IIe (128K DOS 3.3), Apple IIc/Enhanced Apple IIe with 128K (ProDOS), Commodore 63/128,™ and IBM PC,™ XT™ and jr.™

Optional Recipe Disks $19.95

- How smart cooks eat better and waste less.
- Optional recipe disks, over 3,000 exciting recipes, add variety to your menu.
- Crowd-pleasing favorites include:

Appetizers	Holiday Meals
Desserts	Food Processor Cooking
Meatless Meals	Wok Cooking
Kids Cookery	California Beef
Breads & Spreads	Special Diets (includes food allergies)
Daily Bread & Beyond	Daily Bread & Beyond, Food Processor
Microwave Cooking	Soups & Salads

...and more on the way!

New for Fall 1986

Pinpoint Toolkit™ $69.00

Programmers resource and toolbox for writing Desktop Accessories all your own. Includes several handy desktop accessories for just about anyone, not just for programmers:

- Accessory mover for adding new accessories anytime.
- Pop-up version of ProDOS Filer.
- Screen printing facilities, text or graphics depending on printer type.
- Enhanced printing for Desktop Appointment Calendars.
- Decimal, Hexadecimal, or Binary Calculator with print tape.

Outstanding tools for any serious ProDOS programmer include:

- Comprehensive Pinpoint Internal Architecture Documentation.
- Video Resource Editor for accessory development and prototyping.
- Pinpoint's newest interactive desktop environment, RunRun, for expanded limited multi-tasking capabilities.
- Commented source code to several Pinpoint Desktop Accessories provided as programming examples.
- Programmer utilities require 256K or greater Enhanced Apple IIe or Apple IIc, ProDOS Assembly language experience, or use of MICOL BASIC, or KYAN PASCAL.

New for Fall 1986
(continued)

Pinpoint Instant Business Letters $49.00

Business letter templates as AppleWorks word processor documents.

- Ready to use and professionally designed to meet your everyday business needs.
- Perfect for mail merging with Pinpoint's InfoMerge.

Pinpoint Document Checker $69.00
(with Spelling Checker only $99.00)

The fast, efficient stand alone spelling checker for larger AppleWorks documents. Up to 30 words per second!

- Uses the same dictionaries as the Pinpoint Spelling Checker Accessory.
- Selectable auxiliary dictionaries for customized professional needs.
- Lists word occurances. Remembers and automatically uses any correction.

Call for information about Macro Keys and Graphic Templates.

Contact Pinpoint for complete product information and scheduled release dates.

Index

Pinpoint Publishing
5901 Christie Ave.
Emeryville, CA 94608
(415) 654-3050
Telex 245-8579 MCI
CompuServe 76244,123

P.O. Box 13323
Oakland, CA
94661-0323