by
BLAIR & MOSS

$49.95
For Apple II*

PASCAL DISK QTILITY

A DISK UTILITY FOR EXAMINING, CHANGING, MODIFYING, ASSEMBLING, OR DISASSEMBLING PASCAL

DATAMOST™

*Apple is a trademark of Apple Computer, Inc.

PDQ
DATAMOST PASCAL DISK UTILITY

by
Richard Blair and Perry Moss

Documentation by
Scott Knaster

DATAMOST, Inc.™

NOTICE

PDQ, the DATAMOST Pascal Disk Utility runs under the Apple Pascal Language System. "Apple Computer Inc. and The Regents of the University of California make no warranties either express or implied regarding the enclosed computer software package, its merchantability or its fitness for any particular purpose."

This manual and the software herein described are sold on an "as is" basis. The entire risk as to its quality and performance lies with the buyer.

This product is copyrighted and all rights are reserved. DATAMOST INC. shall have no liability or responsibility to the purchaser or any other entity with respect to any liability, loss or damage caused or alleged to be caused directly or indirectly by this product, including but not limited to any interruption of service, loss of business and anticipatory profits or consequential damages resulting from the use of this product. Copying for purposes of backup is authorized to the purchaser and no other person or entity. Otherwise copying, duplicating or distributing this product is strictly forbidden.

If the PDQ disk becomes defective, DATAMOST INC. will replace the media on which it is supplied providing it is returned to a dealer within 30 days with proof of purchase. After 30 days return the disk with $5.00 for post-warranty replacement. The above warranty does not apply if the disk is defective due to abuse, neglect, or mishandling.

PDQ—DATAMOST PASCAL DISK UTILITY

CHAPTER		PAGE
	Introduction: Who Should Use PDQ?	1
1.	What Is PDQ?	2
	How to Use the Documentation	
	Warning about Backups	
	How to Use PDQ	
2.	The Editor: Patching Things Up	5
	Introduction to Concepts	
	How to Begin	
	Command Keys	
3.	The Mapper: Stalking the Wild Codefile	9
	Introduction to Concepts	
	How to Use	
4.	The Disassembler: A Clearer Picture	11
	Introduction to Concepts	
	How to Use	
	Example	
5.	The P-code Assembler: For Pseudo-Programmers	15
	Introduction to Concepts	
	How to Use	
	Pseudo-ops	
6.	The Transcend Unit: A Better Mousetrap	
	Introduction to Concepts	
	How to Use	
	Setting the Prefix	18

Appendices

A: Apple Pascal diskette directory structure 19

B: Pascal device numbers 21

C: Using the Editor with DOS and SOS diskettes 22

D: Case jumps and OTHERWISE how to do it 23

E: P-code standard procedures 27

F: Operating system calls 28

G: A 6502 code application as a detailed example 29

H: .INCLUDE Example 31

Introduction:

Who Should Use PDQ?

PDQ is intended to be used by anyone who has an interest in the intricacies of a sophisticated computer language and operating system; by anyone who has been programming in Pascal and would like to know more about how it all works; by a Pascal programmer who needs additional development tools; by anyone who is learning to program in Pascal and would like a fuller understanding than the standard documentation provides; or by anyone who has caught the personal computer programming bug and is looking for new and better ways to explore the deep, dark secrets of his or her Apple.

Requirements: What you should have

Hardware: Apple II or Apple II Plus,
each with 48K
Disk II (at least one)
Video monitor (of course)
Apple Language Card (or equivalent)
Apple Pascal (semi-optional)
80 character video card (optional)
    Sup'r'Term
    Videoterm
    Smarterm etc.
printer (optional)

Brainware: knowledge of Apple Pascal or desire to learn
knowledge of programming in general
knowledge of assembly language
    (only occasionally)

It would also be very useful to obtain a copy of Randy Hyde's **P-source**, published by DataMost.

Chapter 1

What Is PDQ?

Computer software can be a very complex beast. One of the most important factors in this complexity is the number of levels that software must traverse in order to be executed. As a high-level language programmer (or a prospective one), you probably see only Pascal source text most of the time. However, the path that this software must follow when it is executed is somewhat reminiscent of the mazes in "Adventure": the source text becomes p-code, the p-code is run through a p-machine and interpreted as machine language, and purists out there would point out that even machine language is not the final step, since it is converted, within the tiny world of the microprocessor itself, into microcode instructions. Whew! But, we're getting a little ahead of ourselves.

PDQ is a tool to help you see what goes on in the Apple Pascal Operating System and programs running under the control of that operating system. Specifically, PDQ consists of these tools: a disk/memory editor, a codefile mapper, a disassembler, and a p-code assembler. In addition, there is an improved version of the Apple Pascal unit called transcend, which performs several transcendental functions (hence the clever name). Here's a little about each part:

The Editor

The Editor lets you examine and change, byte by byte, any part of a diskette by file name or block number. In addition, you can look at and modify the Apple II's main memory. Yes, you can skip happily through your disk files, cheerfully turning them into useless garbage if you really want to (see Warning about Backups below).

The Mapper

This tool will show you in detail a lot about a codefile, including a segment dictionary giving each segment's number, name, kind, code type, and more, plus information about each procedure within each segment, more than enough information to put you on a first-name basis with your codefiles.

The Disassembler

The Disassembler takes p-code instructions produced by the compiler or the p-code assembler and emits (a technical term meaning "spits out") p-code mnemonics and pseudo-ops. This lets you take a look at how the compiler implements Pascal statements. Using this information, you can learn how to make your Pascal programs more efficient. In short, the Disassembler turns raw p-machine language into "p-assembly language", and 6502 stuff into its assembly language.

The P-code Assembler

The P-code Assembler, faithful companion to the Disassembler, gives you a rare opportunity to flail away at the p-machine at a level below the compiler. Once you become familiar with the operation of the p-machine, you will be aching to try some p-code programming yourself. The P-code Assembler lets you write your own p-code programs.

For those of you out there who felt a little (or a lot) lost during the preceding discussion, a word of reassurance is in order. Almost everything you need to know is contained in (1) this manual, (2) Apple's excellent manuals, or (3) PDQ itself. After all, one of the purposes of this software is to help you learn about the Apple Pascal system. So, if you think a p-machine is a device for processing vegetables, don't worry.

How to Use the Documentation

This manual is your guide to the successful use of PDQ . However, in order to get the maximum use out of the system, we will periodically refer you to the Apple Pascal Operating System Reference Manual for discussion on various topics. Each chapter contains a section called Introduction to Concepts. This section will explain the concepts used in that chapter and tell you where to go for further information.

Warning about Backups

PDQ contains some superb tools. In particular, the Editor is a lot like a surgeon's knife: it can do wonderful work, but if you're not careful (and

sometimes, even if you are) you can cause a fatal wound to your diskette. Unlike the surgeon, however, the programmer can protect her(him)self. Before using the Editor, make a backup copy of the diskette you'll be working on! Just to be safe, back up the PDQ diskette, too.

—A public service message.—

Chapter 2

The Editor: Patching Things Up

The Editor lets you read, examine, and change information on diskette or in memory. You can view the information in hexadecimal or ASCII character formats. You can select a block at an absolute location on a physical unit, within a file, or in memory. You can move freely through the block as you edit it. You can search through the information for an occurrence of a given value.

Introduction to Concepts

The fundamental concept used in the Editor is the block. A block is a chunk of 512 bytes read either from a physical device (normally a diskette) or from the Apple's memory. An Apple Pascal diskette contains 280 of these chunks, numbered astutely from 0 to 279. The more mathematically inclined among you will note that 280 blocks of 512 bytes each gives 143,360 bytes or 140K, the capacity of a 16-sector Apple diskette. It's nice to know that these things all work out, isn't it?

Note that when editing the Apple's main memory, 512 bytes is equal to two pages, a page being 256 (hex 100) bytes.

How to Begin

To enter the Editor, select E from the menu. As you enter the Editor, you will be asked for a filename. If you want to edit a diskette file, enter its name, preceded, if necessary, by a unit name or volume number, e.g. "#5:ROGER". If you want to edit the Apple's memory, enter a dollar sign instead of a file name. If you want to edit an absolute block number on the diskette, press return without typing anything and the prompt line will change and ask you for a unit number instead of a file name. In the Pascal operating system, the disk drives are, in order, units 4, 5, 9, 10, 11, and, if you're one of those many people who have 6 drives, 12. Enter the unit number of the drive you want to read from. If you press return without typing a unit number, the Editor will give up on you and return to the menu.

No matter which option you choose, the editor will next display the 512 bytes of block 0 in the chosen file, unit, or in memory. If you're using

the Apple's standard 40 column display, note that you can only see the first 13 1/2 bytes of each line. To see the rest of the picture, press ConTRoL-A, but then if you've been using Pascal with a 40 column screen, you already know that.

The Information Lines

At the top of the screen you will see some information about the block being edited. First, of course, is the name of the file or unit number that you typed in when you started. At the end of the first line is the current block number being edited. This starts out at 0, but can change as you move to other places on the disk or in memory. On the second line is the current location within the block of the cursor, which should be down there blinking away even as you read this. Following the byte number is a message which tells you if characters you type will be interpreted in upper or lower case. The next message, window, tells you the blocks which the Editor has already read and will not require a further disk access for. The Editor, being a very intelligent creature, will always try to stay a step ahead and minimize disk access by reading blocks surrounding the one you're editing. The last information line has a carat pointing either left or right. This indicates the direction of the search when you search for values in the block. The value after "target:" (initially blank) is the value to be searched for.

If you're looking at the hexadecimal display, you can change the value that the cursor is on by entering a new byte with the numbers or A through F. If you have the ASCII display, you can enter any ASCII character. Note that control characters are displayed in inverse.

And now that you know that, we'll explain how to use it all.

Command Keys

Control characters

B — Pressing ConTRoL-B lets you select the block number to be edited. If you are editing a file, the block number will be relative within the file. If you are editing a unit or memory, the block will be absolute. See also the warning under the ConTRol-N option which applies here as well.

G — ConTRoL-G sends you to the byte number on the information line. You can then enter a byte number from 0 to 511 and the cursor will move to the location you specify. If you enter a byte number greater than 511, you will be rudely ignored.

I — ConTRoL-I acts like a tab key and moves the cursor five bytes forward.

L — ConTRoL-L acts like it does in the Apple Pascal editor by moving the cursor down one line.

N — Pressing ConTRoL-N, which stands for new file, moves you to the filename prompt on the first line and lets you select something new to be edited.

Warning: if you've made any changes to the current block, you'll see the disk light come on (if you're editing a disk file) and know that the changes are now permanent. If you were editing at an absolute block number you will be given one last chance to change your mind. The message "<space to write>" will be displayed which allows you to press space to apply the changes or any other key (such as ESCape) to save your file from total annihilation. The various safety options such as !ABORT and <space to write> may cause you some confusion as to what really is on the disk. To find out what the current block actually looks like on the disk you may use ConTRol-N to re-access the file (or move outside the window then back again) and so force the editor to reread the block.

Moral: what is in the window buffer may not be what is in the file.

O — As in the Apple Pascal editor, moves the cursor up 1 line.

P — If you have a printer plugged in to slot 1 (unit #6) and it's turned on, loaded with paper, and ready to go, the current screen (80 bytes, remember) will be sent to the printer.

Q	Pressing ConTRoL-Q while you're looking at the ASCII screen lets you enter a character which would normally be interpreted as an editor command, such as ConTRoL-G.
V	Pressing ConTRoL-V lets you toggle the case of characters being typed between upper and lower.
Y	ConTRoL-Y switches the display between its two modes: hexadecimal and ASCII.

Special keys

ESCape	Lets you exit from the Editor and return to the menu.
!	Causes the shocking word ABORT to appear on line 4. This means that the block you're editing will not be updated. Note that ABORT goes off if you make any more changes.
< >	These keys set the direction for the search command. (see below)
=	Search command. You are prompted for a target string to search for. Enter an ASCII string and press return. The Editor will search for the string within the current window, starting at the cursor position and proceeding in the direction set by < or >. If the pattern exists, the cursor will stop at the character in front of the pattern. Note that a response of $ to the target prompt will repeat the previous target.
+	Displays the next block.
–	Displays the preceding block.
– – –>	Moves the cursor one space forward.
<– – –	Moves the cursor one space back.
RETURN	Moves the cursor to the beginning of the next line.

Chapter 3

The Mapper: Stalking the Wild Codefile

(some parts are executable)

The Apple Pascal codefile is a very complex thing. Due to some of the fancy things that can be done with Apple Pascal, the codefile has to carry a lot of information around with it. This information includes code type, whether the code needs other codefiles such as libraries before it can be executed, entry points for procedures, and more. The Mapper lets you examine all this information in your (and other people's) codefiles.

Introduction to Concepts

The format of an Apple Pascal codefile is described in great and wondrous depths by the Apple Pascal Operating System Reference Manual on pages 266 through 270. It is highly recommended that you read that section as many times as necessary before using and while using the Mapper. The Mapper is basically a tool for extracting this information from a codefile.

How to Use the Mapper

To enter the Mapper, select M from the menu. After the hilarious greeting message appears, you will be asked to enter the name of your codefile. If you respond by pressing return without typing anything, the Mapper will assume you don't want to be here and will return you promptly to the menu. After the Mapper determines that your file exists, you will be prompted for an output file name. Just pressing return here will cause the output to go to the console.

The output will begin by telling you the segment number and name of the first segment in the codefile. Following the segment name is the segment kind, which is one of the following:

Linked: fully executable, with no unresolved external references.

Hostseg: The outer block of a Pascal program with external references.

Unitseg: A regular unit.

Seprtseg: A separately compiled procedure, such as an assembly language codefile.

Unlinked_intrins: An intrinsic unit with unresolved external references.

Linked_intrins: A ready-to-use intrinsic unit

Dataseg: An intrinsic unit's data segment

Next, the Mapper will give you the code type contained in this segment. The possible values are:

Unidentified or unknown type
P-code
6502 Machine language
along with a version number.

The next item will tell you which intrinsic units are required by the segment. Then, the Mapper will display the code block address and byte length.

Following the segment information, the Mapper will display information about each procedure in the segment, including its number, lexical level, file offset, and enter IC.

You may wish to compare the information provided by the Mapper to that produced by the Libmap program provided by Apple.

Chapter 4

The Disassembler: A Clearer Picture

The Disassembler takes as input the p-code produced by the compiler or our P-code Assembler and produces an assembly-style mnemonic source listing of the code. In addition, the Disassembler will automatically start disassembling 6502 machine language if it encounters 6502 code.

Introduction to Concepts

The Apple Pascal pseudo-machine, called the p-machine by lazy-fingered documentors, is a software-simulated "machine" which has as its machine language p-code, or pseudo-code. For an in-depth discussion of the p-machine and its operation, consult appendix A in the Apple Pascal Operating System Reference Manual.

How to Use the Disassembler

To enter the Disassembler, select D from the main menu. You will be asked for the name of a file. As always, pressing return will take you back to the menu. You can also disassemble memory by entering a dollar sign. If you enter a file name, you will be asked, in turn, for a codefile slot number, procedure numbers, whether to prepare an assembler input file, and the name of the file to be listed to. Responding to the slot number question by simply pressing return will take you back to the menu. Responding to the other questions by just pressing return causes a default value to be used: 0 for procedure number, dump, and list to the console.

Once you have given the Disassembler all the requested information, it will faithfully begin turning your codefile into p-code mnemonics. Also, if any procedures are written in 6502 machine language, the Disassembler will produce 6502 assembly language for that procedure.

For an in-depth discussion of the p-code instructions used in the Apple Pascal p-machine, see appendix A in the Apple Pascal Operating System Reference Manual.

In addition to the mnemonics produced by the Disassembler, certain pseudo-ops are also generated. Curiously, these are some of the same

pseudo-ops which can be used by the p-code and 6502 assemblers (see next chapter for details on p-code pseudos).

NOTE THAT THE DISASSEMBLER WILL ONLY PRODUCE CORRECT RESULTS ON FULLY LINKED CODEFILES. IN ADDITION THE DISASSEMBLER CAN ONLY WORK WITH FILES THAT ARE NO BIGGER THAN ABOUT 10K.

6502 specifics

6502 procedures are always initiated with .PROC <label>,0. This shows that the disassembler does not really know how many words of parameters are required so it assumes zero. It also assumes .PROC instead of .FUNC since that information, like parameter size, is resolved by the linker.

Labels: three kinds of labels are produced by the 6502 disassembler. They are SEGMxxxx, PROCxxxx, and GLOBxxxx where the four x's stand for segment relative addresses. These represent addresses known to the whole SEGMent, just to the current PROCedure, and GLOBal data segment references respectively. In other words the type of the label tells you something about its scope. .REF and .DEF pseudos are only generated for SEGMxxxx labels since they represent communication between assembly (6502) procedures and functions. Every procedure will have a .REF for all external labels whether it actually contains a reference to that address or not. The xxxx addresses are always relative to the beginning of the segment.

.BYTE: The disassembler generates a .BYTE when invalid instruction codes are encountered, or when another instruction refers to the second or a subsequent byte of an instruction. For example, the byte sequence "A9 FF" would normally be interpreted as "LDA #FF"; but if another instruction, such as a load, refers to the second (FF) byte, then the following is produced instead:

```
        .BYTE A9
<label> .BYTE FF
```

Another handy feature which the disassembler provides for you is automatic display of relative branch destinations. After any branch

instruction is disassembled, the instruction which is the destination of the branch will be displayed in the comment portion of the instruction. If this destination is in turn another branch, its destination will be displayed as well, and so on.

Note that "$" is not the location counter specifier as the Apple manual states. Use an "*" instead. Hence .EQU * instead of .EQU $.

Note that the Disassembler will generate a text file that is about 8 times longer than the code file that is disassembled. Thus if you wish to use the editor on a resulting text file it would be wise to disassemble the code file procedure by procedure.

Example: SYSTEM.STARTUP

In order to get acquainted with the Disassembler, let's run through an example together. Come on—don't be shy! Put in your PDQ disk and turn on the power. Press any key (just as it says) and you'll be looking at the menu. Press D to disassemble a codefile.

After the screen clears, the Disassembler will start asking its questions by wanting to know the name of the codefile to use. Just to be sure we're all getting the same output, we'll use the "greeting" program, SYSTEM.STARTUP. Type in SYSTEM.STARTUP and press return. If you didn't do anything silly like take out your diskette or pour peanut butter on the RAM, you should see that this codefile has only one segment (number 0) and it's called STARTMEU. Since this is the only segment we have to choose from, enter 0 and press return. Next, you're asked which procedures you want to look at. Enter another 0 so that we can look at all of them. Press 1 for a dump (sounds terrible, doesn't it?) of the disassembly. Just press return when asked for the listing filename. This will send the listing to the screen.

Now, when the Disassembler starts shooting lines to the screen, press ConTRoL-S to freeze the display. Here's what the first few lines look like:

```
                        .LEX    0
                        .PARAM 4
                        .DATA   0
                        .PROC   1
        UJP             P22
P2      CXP             28,1
        LSA             6,PEDUTI
        NOP
        CXP             28,2
EXIT    SLDC            28
        CSP             22
        UJP             P27
P22     SLDC            28
        CSP             21
```

What does this mean? Well, first up are the pseudo-ops. The .LEX 0 tells us that this procedure takes place at lexical level 0. The .PARAM 4 tells us the number of words of parameters in the procedure. The .DATA 0 says that there is no data space, and the .PROC 1 says that this is the first procedure in the segment. Information on these items is provided in the next chapter and also in appendix B of the good old Apple Pascal Operating System Reference Manual.

The next 3 lines are simply p-machine "assembly language" instructions. To discover their meaning, we refer to Apple's appendix A, Operation of the P-Machine. The first instruction, UJP P22, means "jump unconditionally to location P22." Note that in the actual p-code, the destination of the jump is given as an offset from the current location, but our friendly Disassembler has computed the destination for us automatically. If you look farther down in the listing, you'll see P22, the jump's destination. The next instruction, CXP 28,1, means "call external procedure 1 in segment 28." In order to get a good idea of how the Disassembler works, it's handy to write and compile some simple Pascal programs and then look at them with the Disassembler. It's really fascinating to see how your Pascal programs are translated to p-code!

Chapter 5

The P-code Assembler: For Pseudo-Programmers

The p-code Assembler turns the function of the Disassembler around. It converts p-code mnemonics and pseudo-ops into p-code. It is not the exact inverse of the Disassembler, however: remember that the Disassembler also works with 6502 code. If you want to assemble 6502 code, Apple provides a very nice assembler with the Pascal system.

Introduction to Concepts

If you think that assembly language is the bizarre form of English that tells you how to put together toys made in Taiwan, you had better hold off on using the Assembler for a while. You may wish to get a good book on assembly language, such as Datamost's **Assembly Language** or Osborne/McGraw Hill's **6502 Assembly Language Programming**. No matter what you know, you should probably read appendices A and B on the p-machine in the Apple Pascal Operating System Reference Manual. If you feel comfortable with assembly language in general, read (thoroughly!) appendices A and B in the Apple Pascal Operating System Reference Manual. These will describe how the p-machine works and give you a good introduction to their use. As a programmer, you should get some ideas for things to do with p-code after you read the appendices.

This assembler will produce only two kinds of codefile. The first is a fully linked segment, which resides in segment 1. The other kind of code file that can be produced is a linked intrinsic (any segment #).

How to Use

The best teacher for using the Assembler is the Disassembler. After all, what better way to learn about putting things together than by taking them apart? Write a few simple Pascal programs, compile them, and examine them with the Disassembler. After a while you may wish to try writing some p-code yourself. To write a p-code program, use the Apple Pascal editor. Instead of starting from scratch, you will often use the Assembler to edit a file which you previously created with the Disassembler. This is a handy procedure to use if you want to make changes to a compiled program and you don't want to (or can't) recompile the source text.

Pseudo-ops

The pseudo-ops which can be used with the Assembler are the same ones produced by the Disassembler, but with a few extra ones for more power. All operands are in decimal.

.LEX n	The procedure's absolute lexical nesting level. Optional, default is 0.
.PARAM n	The number of bytes of parameters passed. Optional, default is 0.
.DATA n	The size of the data, in bytes. Optional, default is 0.
.PROC n	The procedure number within the segment. This should be the LAST pseudo-op before P-code. This is a REQUIRED Psuedo-op.
.INTRI n[,n]	Lets you specify required Intrinsics in range 7-31. Optional.
.SEGNUM n	Specifies the desired segment number. Optional, default is 1. Default for intrinsic unit is 25.
.SEGNAM string	Specifies the segment name (the string should not be quoted). Optional, default is 'DEFAULT'.
.INTERFACE	Indicates that the following text is to be used as the interface section of a linked intrinsic unit. All text up to and including the word IMPLEMENTATION is copied directly into the text portion of the code file. Optional, produces intrinsic unit.
.END	Specifies the end of the assembly. Place it at the end of the file. Required.
.INCLUDE	Lets you assemble from more than one file. Use it in place of the .END at the end of each file. In reality it chains the files together. Optional.

The above pseudo-ops should be placed at the beginning of each procedure with the exception of .END and .INCLUDE which go at the end of a file.

Although you may have any number of procedures, up to 149 per segment, they MUST be numbered properly. For example if you have 4 procedures, they must be numbered 1-4; however order doesn't count (e.g. 4,2,3,1).

CONVENTIONS

Comments begin with a '*'.

All standard procedure mnemonics have a '(P)' appended to them. This notation is used to show that they are standard procedures and may also be called by a CSP. Other mnemonics are as in the Operating System Reference Manual.

Spaces are used as delimiters between the label, operator, and operand fields.

Mnemonics that deal with strings (i.e. LSA,LPA n,<chars>) use the next n characters as the string, going to the next line(s) if necessary. The next n characters are taken from the text file no matter what they are; i.e. part of a comment, the next instruction, or a pseudo-op.

XJP instructions may use either labels or self relative addresses (e.g. $-34) as jump table entries. See appendix D.

All labels MUST begin in column one.

The labels ENTER and EXIT have special meanings to the assembler. They are used to specify the starting and termination points of the procedure, respectively. EXIT is used by the p-machine to return from a procedure or function via the exit(<proc>) standard procedure. ENTER defaults to the first p-code instruction while EXIT is assumed at the RBP or RNP return instruction if not explicitly used.

See appendices D,E,F for further information and examples.

Chapter 6

The Transcend Unit: A Better Mousetrap

Transcend is a Pascal unit which replaces the unit of the same name in the SYSTEM LIBRARY supplied by Apple. It improves the system's performance in executing certain transcendental functions by using the ROM routines from Applesoft BASIC. If you have an Apple II Plus, you have Applesoft in ROM in your Apple. Users of the Apple II will not be able to utilize this unit (sorry).

Introduction to Concepts

DATAMOST's version of the Transcend unit is called Transwitch. It is extremely simple to use, since all you have to do is install it in your library file and (as they used to say on those oven commercials) it does the rest. There are no other new concepts necessary (we thought we'd wind up with an easy one).

How to Use

In order to use Transwitch, you must execute the Apple-supplied utility program called LIBRARY. It's normally supplied on the APPLE3: diskette that comes with Apple Pascal. The documentation for this utility is in the omnipresent Apple Pascal Operating System Reference Manual on pages 186 to 193. Take particular note of the example, Installing a Unit or Routine into a Library, which starts on page 188.

Once you've installed the Transwitch unit in your library, your work is done. From then on, any programs you write which use transcendental functions will automatically use Transwitch.

Setting the Prefix

The prefix is simply a string of up to eight characters which is tacked on to the beginning of any filename entry you make. It is initially set to a colon so that the prefix volume of the system {set from the F(iler} is used. You may change it from the main menu level of the PDQ. The appearance of a colon in your file entry will cause the current PDQ prefix to be ignored.

Appendix A

Apple Pascal diskette directory structure.

This is the Pascal declaration for the Apple II Pascal disk directory. Note that the directory begins at block 2. This information can be very handy for use with the Editor.

```
CONST
maxdir =77;
      (* maximum number of entries in directory *)
vidleng = 7;
            (* number of characters in volume id *)
tidleng = 15;
          (* number of characters in title id *)
fblksize = 512;
        (* standard disk block length *)
dirblk = 2;
  (* directory starts at this disk block address *)

TYPE
     daterec = PACKED RECORD
     month : 0..12;
           (* 0 implies meaningless date *)
     day : 0..31;
     year : 0..100;
          (* 100 implies temporary file *)
  END;

vid = string [vidleng]; (* volume id *)

dirrange = 0..maxdir;

tid = string [tidleng]; (* title id *)

filekind = (untypedfile, xdskfile, codefile,
            textfile, infofile, datafile,
            graffile, fotofile, securedir);
```

```
direntry = PACKED RECORD
(* this is the directory itself *)
            dfirstblk : integer;
              (* first physical disk address *)
            dlastblk : integer;
            (* block following last used block *)
            CASE dfkind : filekind OF
                  securedir, untypedfile :
                     (filler1 : 0..2048;
                      dvid : vid;
                      deovblk : integer;
                  (* number of blocks in volume *)
                     dnumfiles : dirrange;
                      (* number of files in dir *)
                     dloadtime : integer;
                            (* not used *)
                     dlastboot : daterec;)
                      (* most recent date *)
                  xdskfile, codefile, textfile,
                  infofile,datafile, graffile,
                  fotofile :(filler2 : 0..1024;
                       status : boolean;
                       dtid : tid;
                            (* title of file *)
                       dlastbyt : 1..fblksize;
                  (* no. of bytes in last block *)
                     daccess : daterec;)
                      (* date last modified *)
        END;
      END;

directory = ARRAY [dirrange] OF direntry;
```

Courtesy International Apple Core.

Appendix B

Pascal device numbers

Unit number	name
#1	CONSOLE:
#2	SYSTERM:
#4	(diskette in slot 6, drive 1)
#5	(diskette in slot 6, drive 2)
#6	PRINTER:
#7	REMIN:
#8	REMOUT:
#9	(diskette in slot 4, drive 1)
#10	(diskette in slot 4, drive 2)
#11	(diskette in slot 5, drive 1)
#12	(diskette in slot 5, drive 2)

Reprinted, with permission, from Apple Pascal Operating System Reference Manual.

Appendix C

Using the Editor with DOS and SOS diskettes

Although the Editor (and the rest of PDQ) is written under the Apple II Pascal Operating System, it can be used effectively to edit other Apple 16 sector diskettes, namely Apple II DOS 3.3 and Apple /// SOS diskettes. Although SOS diskettes have a completely different directory structure than Apple II Pascal diskettes, the system of block numbering is the same. This means that you can use the Editor with SOS diskettes as long as you only use the unit read and write options, and not the filename read and write (see chapter 2 for details).

Using DOS 3.3 diskettes is a little more complicated, since DOS does not use the block method of partitioning a diskette. Instead of 280 blocks, DOS divides a diskette into 35 tracks of 16 sectors each, a total of 560 sectors in all. Intuition tells us that since there are twice as many sectors on a diskette as there are blocks, a sector must be half the size of a block, or 256 bytes. Intuition is correct. Pascal blocks correspond to DOS sectors in the following way: the first block on each track is located in sectors 0 and 14. The next 6 blocks are, respectively, in sectors 13-12, 11-10, 9-8, 7-6, 5-4, and 3-2. The last block on the track is in sectors 1 and 15. Here's a Pascal procedure to convert block numbers to track/sector numbers for you:

```
procedure blocktosectors (block : 0..279;
                          var track : 0..34;
                          var sector1, sector2 : 0..15);
begin
   track := block div 8;
   sector2 := (7- (block mod 8)) * 2;
   sector1 := sector2 + 1;
   if sector2 = 0  then sector2 := 15;
   if sector1 = 15
     then sector1 := 0;
end;
```

Appendix D

Notes on the XJP instruction. Implementing an OTHERWISE for CASE statements.

The best way to explain some of the idiosyncrasies of the XJP instruction is by way of example. Listing D-1 is a Pascal program that counts the occurrences of the vowels a,e,i, and the letter b in a text file. The case jump will be explained and at the same time we will see how to implement an 'OTHERWISE' for the case statement.

Refer to listing D-2 and the Apple Pascal Operating System Reference Manual, pg. 240 during the following explanation.

The first parameter, W1, of the case jump is a word aligned word and is the minimum index of the table, in our case 65 which corresponds to 'A'. Note that since this parameter is word aligned, and a 'filler' byte may have been inserted between the XJP instruction (172) and the parameter, EXTREME care should be taken when making changes to the code. If this warning is not heeded the self relative locations of the jump table may become wrong. If in doubt simply replace the byte offsets with labels, and add labels to the proper code locations prior to reassembly.

The second parameter gives the maximum index of the table, in our case a 73 which corresponds to 'I'.

The third parameter is an unconditional jump instruction which shows up as a label on the disassembly. The Pascal compiler will always make this a jump past the the case table. In order to implement the 'OTHERWISE' simply change this label to one corresponding to one of the cases. The case for 'B' was added for this reason. See listing D-3.

Following the third parameter is the case table. In listing D-3 the self relative location for 'B' was changed to a label by way of example. In most cases if the code is changed substantially this should be done to all the locations in the jump table for reasons discussed earlier.

It is apparent from listing D-2 that sometimes the CASE statement is not very efficient, especially if the difference between the minimum and maximum index is large and there are only a few entries which correspond to statements. The remainder of the table entries result in

jumps to the unconditional jump that is the third parameter. The original draft of the program had a space as one of the cases, but this resulted in a case table that was 64 words longer, only one of which performed a meaningful action.

Listing D-1 and associated output

```
(*COUNTS VOWELS A,E,I IN TEXT FILE*)
PROGRAM VOWELCNT;
VAR INPFILNM:STRING[10];
      CH:CHAR;
      A,E:INTEGER;
      I,OTH:REAL; (*USED BY REASON OF EXAMPLE ONLY*)
      OUT,INP:TEXT;
(*$I-*)

BEGIN
  A:=0;E:=0;I:=0;OTH:=0;
  REWRITE(OUT,'#6:');
  REPEAT
    WRITE('TEXT FILE? ');READLN(INPFILNM);
    IF INPFILNM<>'' THEN
     RESET(INP,INPFILNM);
  UNTIL (IORESULT=0) OR(INPFILNM='');
  IF INPFILNM<>'' THEN
  WHILE NOT EOF(INP) DO
  BEGIN
    READ (INP,CH);
    IF CH<>' ' THEN
    CASE CH OF

    'A': A:=A+1;
    'E': E:=E+1;
    'I': I:=I+1;
    'B': OTH:=OTH+1; (*WILL BE USED FOR OTHERWISE*)
    END;(*CASE*)
  END;(*WHILE*)
 WRITELN(OUT,'      ','A      E      I      OTHERS');
 WRITELN(OUT,'      ',A,E:7,I:9:1,OTH:9:1);
END.
```

```
      A      E      I      OTHERS
      19     52     45.0       5.0
```

Original Pascal program with output.

Listing D-2.

```
            .LEX 0
            .PARAM 4
            .DATA 1230
            .PROC 1
                .
                .
                .

            UJP             P226
            LAO             12
            LAO             12
            LDM             2
            SLDC            1
            FLT
            ADR
            STM             2
            UJP             P226
P201        XJP             65,73,P226,
                        $-  47    ; BYTES (SLDO   )
                        $-  22    ; BYTES (LAO    )
                        $-   6    ; BYTES (UJP    )
                        $-   8    ; BYTES (UJP    )
                        $-  48    ; BYTES (SLDO   )
                        $-  12    ; BYTES (UJP    )
                        $-  14    ; BYTES (UJP    )
                        $-  16    ; BYTES (UJP    )
                        $-  49    ; BYTES (LAO    )
P226        UJP             P136

                .
                .
                .

P371        RBP         0
```

Disassembly of VOWELCNT showing the area of code to be modified.

Listing D-3 and associated output

```
                .LEX 0
                .PARAM 4
                .DATA 1230
                .PROC 1
                .
                .
                .
                .

        UJP             P226
OTHE    LAO             12
        LAO             12
        LDM             2
        SLDC            1
        FLT
        ADR
        STM             2
        UJP             P226
P201    XJP         65,73,OTHE,
                    $-    47  ; BYTES (SLDO    )
                    OTHE
                    $-     6  ; BYTES (UJP     )
                    $-     8  ; BYTES (UJP     )
                    $-    48  ; BYTES (SLDO    )
                    $-    12  ; BYTES (UJP     )
                    $-    14  ; BYTES (UJP     )
                    $-    16  ; BYTES (UJP     )
                    $-    49  ; BYTES (LAO     )
P226    UJP             P136

                .
                .
                .

P371    RBP               0

                    A     E      I       OTHERS
                    19    52     45.0    434.0
```

Disassembly with modifications made for OTHERWISE. shown prior to reassembly.

Appendix E

Standard Procedures

Some of the standard procedures as documented in the Apple Pascal Operating System Reference Manual have incorrect procedure numbers, the following is a correct list.

Table E-1

Standard procedures called by CSP # orMnemonic if given.

#	Mnemonic	Function
0		iocheck
1	NEW(P)	new
2	MVL(P)	moveleft
3	MVR(P)	moveright
4	EXIT(P)	exit
5		unitread
6		unitwrite
7	IDS(P)	idsearch
8	TRS(P)	treesearch
9	TIM(P)	time
10	FLC(P)	fillchar
11	SCN(P)	scan
12		unitstatus
21		use
22		free
23	TNC(P)	truncate
24	RND(P)	round
32	MRK(P)	mark
33	RLS(P)	release
34		ioresult
35		unitbusy
36	POT(P)	pwroften
37		unitwait
38		unitclear

Appendix F

Pascal operating system procedures

The following is a partial list of useful Pascal operating system calls. All are called by a CXP 0,#.

Table F-1

Pascal built in's

#	function
2	execerror
3	initfile
4	RESET(f);
5	REWRITE/RESET(f,'filnm');
6	CLOSE
7	GET
8	PUT
9	
10	EOF
11	EOLN
12	read integer
13	write integer
14	
15	
16	read char
17	write char
18	read string
19	write string
20	
21	advance input to eoln
22	write eoln
23	CONCAT
24	INSERT
25	COPY
26	DELETE
27	POS
28	BLOCK READ/WRITE

The gaps in the table reflect deficiencies in the knowledge of this writer rather than in the operating system design.

Appendix G

Sample Application of the 6502 dissassembler

The game paddle function is provided in the Applestuff unit and documented in the Apple Pascal Operating System Reference Manual on pages 143-4. You may "extract" this function and incorporate it into your own Pascal host without having to Use the Applestuff unit.

The first order of business is to boot the PDQ diskette. After a few moments of disk whirring you will see the title page appear on the screen. Press the space bar (or any other key except RESET) and the list of options will be displayed. Select "D(issassemble a codefile" by pressing D. You will then be asked for the

Code file ($ for mem., <CR> quits):

In response to this you should enter System.Library and press <return>. A numbered list will be produced which reads as follows:

0 LONGINTI
1 PASCALIO
2 TRANSCEN
3 CHAINSTU
4 APPLESTU

Slot number:

At this juncture you should reply with a 4 to specify Applestuff which is in the fifth slot. Press <return> and you will get this cryptic message:

Procedure range #[– #] (0 for all):

The procedure number of the paddle function is 2 so enter it and <return>. After the next prompt line specify assembler input (instead of a combination hex-symbolic dump which of course is quite magical) by punching 2 again. You don't want to <return> here since PDQ knows that only one character is called for. When it asks for the

Listing filename:

you will need to type in a name--something like "paddle" to tell the program where to send the dissassembly text. After an "N" reply to an interface inclusion the dissassembly should be under way and a message

should come up which tells you that procedure two is being processed. After some disk action interspersed with dramatic pauses the menu will come up again.

At this point you should exit PDQ. It may be easiest to boot up your regular Pascal Apple1: or what have you since System.Editor and System.Assembler are absent from the PDQ disk.

Enter the system's regular text editor and load your new Paddle text file. Compare this source code with that of the original listing starting on page 143 of the operating system manual. The only point where the code actually produced is different is in the first line. The .Proc SEGMxxxx,0 should be replaced with .Func PADDLE,1. The dissassembler had no information to tell it how the procedure was to be called so you must inform the assembler that you want a function with one word of parameters. If there were any references to SEGMxxxx inside the body of the function you would need to replace these with PADDLE as well.

After the editing session you may proceed to assemble PADDLE and merge it with a Pascal host. You might find it informative to produce an assembly listing to compare the 6502 code values with the listing in the OS manual mentioned above.

As a sample host the program given in the OS manual on page 149 would work if the sections which pertain to TTLOUT were removed. The linking procedure would be the same with the appropriate filename changes.

You now have a means of using the paddle function in a program without having to load the entire Applestuff unit. This is a useful method for conserving space when the original source text is unavailable.

Appendix H

.INCLUDE example

The following shows how to organize multiple files for reassembly. Note that the .TEXT should not be added to the file name.

```
        .LEX 0
        .PARAM 4
        .DATA 0
        .PROC 1
            .                       <- -TEXT1.TEXT
            .                           first procedure
            .
            .
            .
EXIT    RBP 0

        .INCLUDE TEXT2      <- -chains to second file

        .LEX 1
        .PARAM 4
        .DATA 6
        .PROC 3
            .                       <- -this is TEXT2.TEXT
            .
            .
EXIT    RNP 0
    .INCLUDE TEXT 3         <- -chains to third file

        .LEX 1
        .PARAM 4
        .DATA 100
        .PROC 2
          .                         <- -TEXT3.TEXT
          .
          .
EXIT    RNP 0
        .END                <- -end of input file
```

8943 Fullbright Ave., Chatsworth, CA 91311. (213) 709-1202