# The Z-System User's Guide

by

Richard Jacobson
Bruce Morgen

# INTRODUCTION

The Z-System--ZCPR3, ZRDOS, and their related utilities--is a revolutionary microcomputer operating system. Since the release of ZCPR3 early in 1984 over 60,000 systems have been installed. Many of these installations now include ZRDOS. (ZRDOS is a more recent development; many users implemented ZCPR3 before ZRDOS was released in early 1985.) The popularity of the Z-System has been nothing short of spectacular.

Despite its widespread acceptance as the state-of-the-art operating system for 8-bit computers, some users have expressed a concern that they will have trouble learning Z-System.

The **Z-System User's Guide** addresses this concern, getting your past the learning bottleneck and allowing you to enjoy the benefits of the Z-System immediately.

The Z-System gives you a working environment and a degree of control over that environment unsurpassed by any other microcomputer operating system. Compared to the Z-System, CP/M-80 is crude and MS-DOS is lacking in many essentials.

Of course, to the extent that the Z-System provides you with an unparalleled level of control it also calls on you to use that control. While you will be able to treat the Z-System as if it were "plain vanilla" CP/M-80 with no difficulty at all, we believe that the modest demands made on you by the Z-System to extract its full power help generate its most valuable benefits--increased knowledge and enhanced productivity.

The **Z-System User's Guide** is intended to be usable by itself, though it assumes the existence of **ZCPR3: The Manual**, by Richard Conn. That volume is an exhaustive technical reference; this one addresses issues unique to the beginning Z-System user. The multitude of options presented by the Z-System made the problem of selection and proportion difficult, and, if you think some parts of the system are unjustly slighted or ignored, it may be said that our original scope, in consideration of the brief span of human life, has been greatly abridged.

For help and inspiration of various kinds we are indebted to Dave McCord, Jay Sage, Paul Pomerleau, Steve Cohen, Jeff Moro, and the Z-Node sysops. Richard Conn and Frank Gaude' might be called the godfathers of this manual, though their generous beneficence to the 8-bit community has far outrun that degree of responsibility.

Richard Jacobson
Bruce Morgen

Chicago, Illinois
Warminster, Pennsylvania
April 1986

# Z-SYSTEM CONCEPTS

## The Operating System

The Z-System is a revolutionary operating system, a description that raises two questions: (1) What is an operating system?  (2) How does the Z-System differ from CP/M?

## What Is An Operating System?

An operating system is a program that controls the basic housekeeping functions of your computer. The operating system makes characters appear on the screen, turns disk drive lights on and off, tells disk drives to read diskettes, and interprets the commands you enter at the keyboard. The operating system is responsible for the functions that keep your computer running and obeying your commands.  It provides communications among your computer's hardware components, from disk drives to the screen display, from the keyboard to printers and modems.

In many computers, part of the operating system is located in a ROM (Read Only Memory) IC.  This ROM or PROM (Programmable Read Only Memory) is a semiconductor device located on the main board of your computer, and may be programmed to perform many of the housekeeping chores we have been talking about.

Another part of the operating system is located on the outermost tracks of your "system" diskettes.  A system diskette is a diskette from which the computer's ROM "boots" the operating system or loads it into memory.  All computers come with a program to write an image of the operating system either to diskettes or to a hard drive.

When you turn your computer on, machine instructions in ROM tell your computer to read the information located on the system tracks into memory. If this is the only function performed by the ROM, it is called a "boot ROM."  In some computers the ROM is more than a "boot ROM."  If that is the case, after the image of the operating system on disk is read into memory, that memory resident system will call on the ROM to perform needed functions. Notwithstanding the essential nature of the ROM and its functions, when we speak of the Z-System, or any other operating system, we are referring to a powerful housekeeping program located on the system tracks.

The operating system, whether it is the Z-System or CP/M, has three major components: the basic input/output system (**BIOS**), basic disk operating system (**BDOS**) and console command processor (**CCP**).

The **BIOS** is the interface between software and hardware.  It contains the routines that communicate with disk drives, screens, printers, modems, and the like.  Since the BIOS is the only part of the operating system that communicates directly with hardware devices--and, in

some computers, such as Kaypros, the ROM--it is the only part of the operating system that is computer-specific.

The **BDOS** is located just below the BIOS in memory.  It acts like a powerful service agency, for programs and the rest of the operating system, handling such functions as reading from and writing to disks, maintaining the disk file directory structure, processing information that goes to the BIOS for final handling, and allocating disk space.

The **CCP**, which resides immediately below the BDOS in memory, is the program that runs in the absence of any other program.  It is the interface between software programs and the user. The CCP processes the commands you type in at the console keyboard.  When you see the familiar A> or A0> prompt, it is the CCP that put it there.  When that prompt appears, the CCP is telling you that it awaits your commands.

Some commands, such as DIR, REN, ERA, TYPE, and SAVE in a CP/M system, can be processed by the CCP directly.  These commands are called "resident commands" because they reside in memory for the duration of your session at the computer.

If the command is not one of these residents, the CP/M CCP will look on the current drive (the B drive if the B> prompt shows or the A drive if the A> prompt shows) and current user area (more on that later) for a COM file bearing the same name as the typed command.  Such COM files are called "transients" because their availability depends upon their presence on a disk.

After certain COM files have finished running, the message "Warm Boot" may appear on your screen.  "Warm Boot" tells you that the CCP is being read from the system tracks by your disk drives and the data is being loaded into the proper memory locations. Why does the CCP have to be reloaded into memory?  Many COM files take over the functions of the CCP when they run.  WordStar is a good example of such a program.  When WordStar is running it pays no attention to the CCP and it uses the CCP's space in memory for its own purposes.  When you finish using WordStar there is a "Warm Boot", or a re-reading of the CCP into memory, because WordStar has over-written the portions of memory where the CCP is located. If there were no "Warm Boot" after WordStar finished running, the operating system would no longer be able to process commands entered at the keyboard. Unlike the CCP, the BIOS and, in most computers, the BDOS, stay resident in memory during the entire time your computer is turned on.


## How Does The Z-System Differ From CP/M?

The Z-System has the same underlying structure as CP/M.  The BIOS of under the Z-System is similar to the BIOS under CP/M but with certain additional functions during the "cold boot" or initial startup of the computer.  The BDOS performs the same functions in a Z-System as in CP/M, again, with certain significant additions and improvements.  It has a different name, however--ZRDOS.  ZRDOS stands for "Z80 Replacement Disk Operating System."  The CCP in the Z-System is called ZCPR3 and is the crowning glory of the Z-System, with vast increases in functionality and power over CP/M's CCP.

While the Z-System is fully compatible with virtually all CP/M-based software, and virtually transparent to the user of a CP/M based system, that is where the similarity ends.

Before we go on to explore some basic Z-System concepts, let us emphasize that we will not discuss how to install Z-System on a computer currently running CP/M.  There are many ways to accomplish this, ranging from ground-up manual installation using the information supplied in **ZCPR3: The Manual**, supplemented by Echelon's Z-News, NAOG/ZSIG's ONE-EIGHTY FILE and the Z-Node network of Remote Access Systems (RAS), to automatic boot-up with Echelon's Vanguard or Z-Com packages.  Take your choice, depending on your level of microcomputing expertise, but from now on, we will assume you have access to a running Z-

System environment.

# Basic ZCPR3 Concepts

ZCPR3, the command processor of the Z-System, provides a more powerful, versatile and sensible user interface than CP/M's CCP. As a processor of the user's commands, ZCPR3 is unprecedented and unsurpassed in its power and ease of use.

# Multiple Command Lines

CP/M has a "one command at a time" limitation. If you want to look at the directory of a disk, erase the file `MYFILE.TXT`, and then look at the directory of the drive to confirm the erasure, you would have to enter three separate commands under CP/M, each followed by a carriage return (<cr>). Not so with ZCPR3. Under ZCPR3, you can enter one command line consisting of a string of commands, each command separated by a semi-colon, followed by a *<cr>*. Under ZCPR3, the command line would be:**DIR;ERA MYFILE.TXT;DIR*<cr>***.

# Directories and User Areas

A directory under Z-System is a logical concept. Physically a disk has only one area for its directory. A directory under CP/M or Z-System is a specific logical (not physical) place where files are grouped together. A user area is another CP/M technique to define a specific logical area where files can be grouped together. On a two drive floppy disk system there are two CP/M directories, A and B, and 16 accessible user areas, 0 through 15, on each drive. On some hard disk systems, such as the Kaypro 10, you have two physical disks, the hard drive and the floppy, but three logical directories, A, B, and C. Again, each logical directory has 16 user areas.

ZCPR3 uses the term directory in a slightly different way. Under ZCPR3, a "directory" is a logical area on a disk, whether hard disk or floppy. The ZCPR3 directory is defined by its drive letter and its user number. Thus, A15 and B4 are both ZCPR3 style "directories". "A15" means Disk A, User 15. "B4" means Disk B, User 4. This way of referring to ZCPR3 directories is called the "DU (disk/user) form." In addition, a mnemonic name can be assigned to a directory. The name "ROOT" could be assigned to A15 (which we have done on your distribution disks) and the name "WORK" to B0. This way of referring to ZCPR3 directories is called the "DIR" (named DIRectory) form.

The "DIR" form has a number of advantages over the "DU" form, including the simple and obvious advantage of being able to give some meaningful name to a Drive/User to help recall the type of files stored there. It is easier to remember "DBASE" than it is to remember that you stored your DBASE command files in B7. So you just assign the name "DBASE" to Drive/User B7 and when you want to log into that area you type "DBASE:*<cr>*." You will be there in a snap, while your CP/M-bound colleagues are still scratching their heads.

# The ZCPR3 Prompt

The CP/M prompt does not give you much information. If you are logged into the A drive you get an `A>` prompt and if you are logged into the B drive you get a `B>` prompt. Should you change user areas by typing, for example, `"user 15"` (you would not have to do that under ZCPR3), you would still see the same old `A>` prompt. Some computers show you what user area you are logged into, so you get a prompt like `A15>`. This is an improvement over the standard CP/M prompt but not nearly as useful as the ZCPR3 prompt available under Z-System.

Under ZCPR3, the user-installer can configure the prompt. The most popular setup, however, shows both the "DU" form and the "DIR" form. The prompt, A15:ROOT>, shows the logical drive, the user area, and the name of the directory.

# Command Processing

Command processing is what the CCP does after you (or a software surrogate like CP/M's SUBMIT) issue a command. For example, when you type "WS<cr>" the Console Command Processor (CCP) loads and runs WordStar.

Under CP/M command processing is simple, almost primitive. The system receives a command from the user or from a SUBMIT file. It then determines if the command is a CCP resident, such as TYPE, REN, ERA, DIR, or SAVE, and, if the command is a resident, runs it. If the command is not a resident, the CCP determines if the current disk and user area contains a COM file with the same name as the command. If so, it loads and runs the COM file. If not, the CCP issues an error message--the errant command followed by an uninformative "?".

Under ZCPR3 command processing is sophisticated, powerful, and flexible. The ZCPR3 command processor also lets you know a great deal more about what is happening to your commands than CP/M. After parsing (reading) the command line from the user or from a batch file, the ZCPR3 command processor performs the following operations:

1. It checks to see if the command is a Flow Command, a command that determines what, if any, further commands the system will obey. If the flow state is true, the system will continue to process commands. If not, it will flush the command and then continue. (Remember, we are dealing with multiple commands.)

2. It checks to see if the command is a ZCPR3 resident command (ZCPR3 residents are located separately in memory from ZCPR3 itself and are more numerous and powerful powerful than CP/M residents). If the command is a resident, the command processor will run it.

3. If the command is not a resident, the command processor will check to see if the command is "built-into" ZCPR3--a ZCPR3 intrinsic command. If ZCPR3 finds the command is an intrinsic, it will run it.

4. If the command is not an intrinsic, then the command processor looks along a "command search path" for a COM file bearing the same name as the command. The "search path" is a series of Drive/Users you can direct the system to search for COM files. If the desired COM file is found somewhere along the search path, it is loaded and run. Manipulated by the PATH utility, this feature is one of Z-System's most powerful virtues.

5. If the desired COM file is not found along the search path, "extended command processor", if available, is run. This extended command processor can be a memory based batch facility or some other appropriate program that then attempts to process the command.

6. Finally, if all else fails, the system will invoke an ZCPR3 error handler, if one has been installed. An error handler is a program that gracefully absorbs user input errors and optionally permits corrections to command lines. If an error handler has not been installed, ZCPR3 prints an error message, the bad command and the cryptic "?".

There are a great many other differences between CP/M's CCP and the Z-System's ZCPR3. We shall explore them more interactively later on.

# Basic ZRDOS Concepts

ZRDOS, which stands for Z80 Replacement Disk Operating System, is, as its name suggests, a replacement for the Basic Disk Operating System (BDOS) supplied by Digital Research. ZRDOS is fully compatible with CP/M, but offers significant improvements.

With ZRDOS you no longer have the problem of changing disks in the middle of a session and getting the dreaded "BDOS error" message which frequently means means, "Say goodbye to the work you have been doing for the last hour."  ZRDOS automatically logs in changed disks. This means you can pay attention to your work and not to typing Control-C every time you change a disk.

ZRDOS also permits you to write-protect certain files from non-privileged users.  You set the access password.  If another user does not have the access password and you have write-protected certain files, those files cannot be changed or erased by anyone but you.  In addition, ZRDOS allows you to set a file's directory entry to indicate that it has been archived.  With the ZRDOS utility called AC.COM you can back up only those files that have not yet been "archived."  Finally, if you set a disk drive to read/only status, that status is maintained after a warm boot.

If you use programs with overlay files--Wordstar is one popular example-- you will love the ZRDOS public directory feature.  Public files are files that are treated by the operating system as if they were simultaneously present in all user areas of a given drive.  Those of you who own Kaypro 10's know about the absurd waste of disk space under CP/M of having to have a copy of the two big WordStar overlay files in every user area in which you work with WordStar. Otherwise, a screen full of bizarre error messages is the result.  Not so under ZRDOS, which provides a means of setting user areas 1 through 8 as public.  When you put WordStar and all of its overlays into a public user area, you find not only does WordStar work in any user area, but the "R" function (run a non-WordStar file) finally becomes useful.

Other features of ZRDOS are useful and will be mentioned in brief.  The Read Console Buffer Function (10) treats rubout (DEL) the same as backspace.  Single level re-entrance permits compact I/O packages.  Special ZRDOS-only utilities include Set/Display File Attribute, Dump, Compare, View, and Vtype programs.  There is also enhanced error handling with specific error numbers, which are fully explained by the DOSERR program.
Other than the public files facility, the features of ZRDOS may not be apparent to you.  But try this experiment.  After you have been using the Z-System for a few weeks, boot up your old CP/M disk.  No further comment on the topic will be necessary.

# STARTUP AND THE SYSTEM SEGMENTS

STARTUP is a Z-System concept that should be understood in order for you to use the Z-System most effectively.  Once you understand the STARTUP concept, you can use it to create custom Z-System configurations for any application.

## STARTUP Introduction

The file we are interested in is usually called STARTUP.COM, although some systems, like the Micromint SB-180 use similar filenames like START.COM and others, like the Ampro Bookshelf, allow you to change the name of the startup file with a BIOS configuration utility . Now type STARTUP*<cr>* or your system's equivalent.  It's almost like hitting the reset button.  Let's look more closely at STARTUP.COM and see exactly what it does.

Type ALIAS STARTUP*<cr>*.  You should see something like the following, though what you see will vary depending upon which computer you are using and the precise nature of your startup routine:

```
 ALIAS, Version 1.1
Alias Name: STARTUP
Old Alias Command Line:
 1 --> 15:;
 2 --> LDR SYS.ENV,SYS.FCP,SYS.RCP,SYS.NDR;
 3 --> ERROR23;
 4 --> VID HELLO;
 5 --> TIME CD;
 6 --> 0:;
 7 --> CLS
```

Hit a *<cr>* to abort the running of the ALIAS utility.  We are now ready to analyze STARTUP.COM.

## The STARTUP "Alias"

STARTUP.COM is an ordinary COM file.  Let's get rid of that point at the outset.  But it has some special characteristics.  First, STARTUP is an ALIAS, which means that while it is an ordinary COM file it has only one function--to pass a series of commands to the multiple command buffer.  A buffer is an area of memory that has been reserved for a specific use.  The multiple command buffer in the Z-System is an area of memory--about 200 bytes in most Z-System implementations--that has been reserved to hold a series of commands.  STARTUP passes a series of commands to that buffer and then the Z-System executes them.

In another sense, STARTUP is more than an ordinary COM file. It is an ALIAS and, most important, a Z-System utility you can create on the fly. A Z-System utility knows about the key Z-System addresses by virtue of its ability to locate and read a Z-System segment listing these addresses located in high memory. (High memory is the area of memory above the transient program area or "TPA" where programs generally run and the operating system itself.) This segment is called the environmental descriptor. Z-System utilities know where the multiple command buffer is located in memory because they extract that information from the data contained in the environment descriptor.

The Z-System tries to run STARTUP on cold boot. If the Z-System finds STARTUP, it runs it. The directive for the system to look for STARTUP on cold boot is "hard-wired" into the BIOS. (The term "hard-wired" means that specific code to make the system look for STARTUP on cold boot is coded into the BIOS). If the file STARTUP.COM does not exist on A0 when the system cold boots, you will see a "?" on your CRT. This message, as you know from CP/M, means the system cannot execute the command entered. If STARTUP is found in A0 it will execute.

Now let's look at the command line STARTUP passes to the command processor.

The first command is "15:", the user area on drive A that corresponds with the named directory "ROOT". It logs you into A15, where, by convention, Z-System utilities are usually located. All the remaining COM files invoked by STARTUP are contained in A15 or ROOT. Note the correct way to move from A0 to A15 is to type 15:<*cr*>. Typingr A15:<*cr*> works too. Please do not use the CP/M "USER 15" command. The Z-System will not recognize it and we will speak of it no more.

The next command is "LDR SYS.ENV,SYS.FCP,SYS.RCP,SYS.NDR". This is the second of seven commands that STARTUP passes to the system and by far the most significant. If you run LDR.COM with the double-slash help parameter, LDR //, you will see the following display:

```
A0:COMMAND>LDR //
ZCPR3 LDR, Version 1.3
 LDR Syntax:
        LDR <list of packages/data files>
 where entries in the list may be any of these types:
        FCP - Flow Cmnd Package         ENV - Z3 Environ
        IOP - Input/Output Package      NDR - Z3 Named Dir
        RCP - Resident Cmnd Package     Z3T - Z3TCAP Entry
 The ENV file must be first if LDR is not installed.
A0:COMMAND>
```

## System segments

The packages we are concerned with are of the type ENV, FCP, RCP, and NDR. They are frequently called SYS.ENV, SYS.FCP, SYS.RCP, and SYS.NDR. LDR loads each of the packages or system segments into their appropriate location in memory. A package can be loaded dynamically at any time by running the LDR utility. Each segment has a distinctive file type. LDR recognizes these file types and loads each system segment differently. The Z-System is configured to load each type of system segment starting at its own fixed memory address. To look at these addresses, run the Z3LOC utility. The command is Z3LOC Z<*cr*>. You should see the following output (the specific addresses will vary depending on which version of the Z-System you are running):

```
A0:COMMAND>Z3LOC Z
Z3LOC  Version 1.1
```

```
            ZCPR3 Element       Base Address
            -------------       ------------
            CCP                 CC00 H
            BDOS                D406 H
            BIOS                E200 H
            Env Descriptor      F800 H
            Pack:   FCP         F600 H
                    IOP         0000 H
                    RCP         EE00 H
            Buf: Cmd Line       FF10 H
                    Ext FCB     F9D0 H
                    Ext Path    0040 H
                    Ext Stk     EDD0 H
                    Messages    F980 H
                    Named Dir   FA00 H
                    Shell Stk   F900 H
                    Wheel Byte  003E H
```

The environment descriptor (Env Descriptor) in this example is loaded in memory beginning at F800H, the flow control package (FCP) at F600H, the resident command package (RCP) at EE00H, and the named directory buffer (Named Dir) at FA00H. These addresses show the starting point of areas in memory the Z-System reserves for specific packages and buffers.

The specific addresses hold true only for a particular implementation of the Z-System. There are many different ways of implementing the Z-System, serving different purposes and offering different capabilities and flavors of Z. Some Z-Systems are implemented with an Input/Output Package or IOP. You can tell from looking at the 0000H address for the IOP in the Z3LOC display above that this system does not contain an IOP segment.

The system segments are key features of the Z-System. Each system segment remains memory-resident until or unless a new system segment is loaded over it. For example, you might have several files on a disk of the type RCP, such as WORDPRO.RCP for word processing, ASM.RCP for assembler work, CALC.RCP for spreadsheets, and so on. Each file would contain a different set of resident commands to be used for different purposes. If you entered the command, LDR WORDPRO.RCP, the LDR utility would load the WORDPRO.RCP resident commands into the area of memory set aside for resident command packages and overwrite any other set of resident commands previously loaded. This is why we say the packages in the Z-System are dynamically changeable.

Now we shall turn to each of the packages and segments loaded by your STARTUP.COM file.

## Environment Descriptor

The environment descriptor is a data file. It is loaded into its reserved location in memory and provides information to all Z-System utilities about the configuration of the Z-System in which that utility is running. This information includes the addresses of the system segments themselves, the location of the message buffer (in which Z-System utilities leave messages to each other and the operating system about various system matters), the number of drives on line, the maximum user area, various flags showing whether utilities are to print certain information on the CRT, or run "quietly", and whether a utility can recognize DU and DIR designations or only DIR designations. In addition, the environment descriptor may include a ZCPR3 TCAP, or terminal capabilities entry, that describes the attributes of your terminal or console CRT, such as the codes needed to clear the screen, set the cursor position, invoke half-intensity video and the like. Finally, the environment descriptor contains the definition of two logical terminals and four logical printers. These terminals and printers can be selected by the CPSEL.COM utility. In short, nearly every significant characteristic of the Z-System is defined in the environment

descriptor.

# Flow Command Package

The flow command package contains ZCPR3 flow commands. Flow commands are memory resident commands that give you many of the capabilities of what the mainframe and minicomputer worlds call job control language. On a simple level, flow commands allow you (even if you are a devout non-programmer) to program a series of commands and set conditions for the execution of one or more commands. On a system level, flow commands set the flow state either true or false. If the flow state is true (the system will automatically test for this), all commands can be processed. If the flow state is false, only flow commands can be processed. The primary flow commands are `IF, ELSE, FI,` and `XIF. FI` (the "backwards IF") is the equivalent of ENDIF, for those of you who have done some programming. `XIF` exits all pending IF's. Let's try some flow commands.

Create a file called `HELLO` with your text editor or word processor and put some text into it. Type `DIR<cr>`. You will see the file `HELLO` in the directory listing. Type the following command line: `IF EXIST HELLO;TYPE HELLO;ELSE;ECHO HELLO DOES NOT EXIST;FI<cr>`. You should see the text you entered in the file `HELLO`. Now try this: `IF EXIST HELLOO;TYPE HELLOO;ELSE;ECHO HELLOO DOES NOT EXIST<cr>`. Your system will tell you politely the file `HELLOO` does not exist. (Now don't tell us you had a file called `HELLOO` sitting around on your disk all this time. We don't believe you.)

The following flow commands are typically available:

| COMMAND | SHORT FORM |
|---------|------------|
| IF | none |
| ELSE | none |
| FI | none |
| XIF | none |
| EMPTY | EM |
| ERROR | ER |
| EXIST | EX |
| INPUT | IN |
| NULL | NU |

The syntax of these flow commands is straightforward.

**IF** tests whether a certain condition is TRUE. If the condition is TRUE, the flow state is set to TRUE. This means any following command is permitted to run. If the condition is FALSE, the flow state is set to FALSE and only flow commands are permitted to execute.

Time for a demonstration. Type `IF IN` and answer the question with an `N`. Now try some normal commands (commands other than flow commands) like `DIR`. No, your computer is not broken. What has happened? IF IN prompts you for input from the keyboard. If you respond with `T, Y, <cr>`, or space (`<SP>`) the flow state is set to true and you can go on to execute other commands. If you respond with any other input, the flow state is set to false and you can only execute flow commands. In fact, you better execute a flow command right now in order to get your computer back into a more useful condition. Type `FI`. Now try some of your ordinary commands, such as `DIR`. Everything is "back to normal."

**FI** terminates the current IF level and drops down to the next "active IF state." Let's say you use a flow command that has the effect of setting the flow state to false. FI then drops down to the previous flow state, which, if true, now sets the system flow state to true. Simply put, FI has the same meaning as ENDIF in conventional programming terminology. There are nine flow states or levels in a Z-System, the empty or "no active IF" state (not to be confused with the "IF

EMPTY" flow command), which is always true, and levels 1 through 8, which can be set by the user.

It is critical to remember that the ZCPR3 command processor is always aware of the current flow state of the Z-System. If this state is TRUE, the command processor will allow any command to execute if the command is available to the user. If the flow state is FALSE, only flow commands may be executed. The IF command raises the system to the next flow level and sets the flow state TRUE or FALSE depending upon the result of testing some condition. When you used the `IF IN` command you were asking the system to test whether there was user input. Hitting `T,` `Y,` `<cr>`, or `<SP>` made the conditional test TRUE; hitting any other key made it FALSE. Because you hit "any other key" you raised the flow state one level and set it to FALSE. That is why the system would no longer obey any of your commands except a flow command. When you issued the command FI, you dropped the flow state down one level. Since the flow state one level down was true, you set the flow state to TRUE as well. If the previous flow state had been FALSE (as the result of a previous conditional test that resulted in a FALSE flow state), you would have needed another FI to get the system to obey your non-flow commands.

The **ELSE** flow command toggles the current flow state. If the current flow state is TRUE, ELSE toggles it to FALSE. If the current flow state is FALSE and the previous flow state is TRUE, ELSE toggles the flow state to TRUE. If the previous flow state is FALSE, ELSE does nothing.

The **XIF** flow command exits all pending IF's if the current flow state is TRUE. That is, it reduces the flow state to the 0 or empty state, which is always TRUE. If the current flow state is FALSE, XIF does nothing.

Flow commands are most obviously useful in batch processing, when you want the computer to do one thing under one set of conditions or another thing under another set of conditions.

Before we start you out with some finger exercises we better go through the rest of the "flow command vocabulary."

| FLOW COMMAND OPTIONS | SHORT FORM | MEANING |
|---|---|---|
| EMPTY afn | EM | If the indicated file is empty (size is 0K), the flow state is set to TRUE |
| ERROR | ER | If the ZCPR3 program error flag is SET, the flow state is set to TRUE |
| EXIST afn | EX | If the indicated file exists, the flow state is set to TRUE |
| INPUT | IN | The user is prompted for input and if the response is T, Y, <cr>, or <SP> the flow state is set to TRUE--any other response sets the flow state to FALSE |
| NULL afn | NU | If the 2nd entry (afn) is on the command line left blank, the flow state is set to TRUE |

The "afn" in the flow command option table means "ambiguous file name" or, in effect,

whatever file you choose it to mean, including a wild-card file designation. Remember card games! Wild-cards can have whatever value you assign to them. Finally,a leading tilde (~) before a condition negates the effect of the condition. If the condition is FALSE, the flow state is set to TRUE, and if the condition is TRUE, the flow state is set to FALSE. Thus, IF ~EXIST MYFILE.TXT sets the flow state to TRUE if the file MYFILE.TXT does not exist.

Don't worry if this seems a bit complex, it really isn't and anyway we are going to do some finger exercises together so that your fingers can learn the flow commands even if the rest of you feels a little balky at this point.

Assuming you keep your ZCPR3 utility SHOW.COM in a named directory called ROOT, log into your ROOT directory by typing ROOT:*<cr>*. Now get a directory listing with DIR*<cr>*. You see the file SHOW.COM listed in the directory, so you know it exists on your disk. Try this flow command: IF EX SHOW.COM;DIR;FI. You should see the directory listing on your CRT. What is happening? You are telling the system if the file SHOW.COM exists, bump the flow state up one level and set it to TRUE. Since SHOW.COM obviously exists, the flow state is set to TRUE. Hence, the program DIR.COM is permitted to execute. The terminating FI knocks the flow state down one level to the 0 state, which is always TRUE. (Obviously, the directory name, ROOT, and the file, SHOW.COM, are arbitrary. You may perform this experiment in any directory, with any file.)

What happens if you pick a file that does not exist? Let's try it. IF EX NOFILE.TXT;ECHO IT EXISTS;ELSE;ECHO IT DOES NOT EXIST;FI. Voilá! The resident ECHO command just told you what you already knew--the file NOFILE.TXT does not exist. Let's try another variation: IF ~EXIST SHOW.COM;ECHO THE FILE SHOW.COM DOES NOT EXIST;ELSE;ECHO IT DOES;FI. It does exist. The first condition set the flow state to FALSE, because the file SHOW.COM does exist, and the next command, not being a flow command, is disabled. Remember, when the flow state is FALSE, only flow commands can be executed. The next flow command is ELSE. If the current flow state is FALSE, which it0 is, and the previous flow state is TRUE (that's true), then ELSE toggles the flow state to TRUE. Thus, the next command can execute. That command, ECHO IT DOES, echoes "IT DOES" to your screen.

Flow commands are primarily used to add intelligent, automated decision-making to Z-System batch processing and not in interactive finger exercises like we have been doing. But we will be getting to batch processing--the use of aliases and Z-System's powerful SUBMIT/XSUB replacement, ZEX--a bit later in the game.

For now, you have the rudiments of flow control under your belt and we can go on to other matters. Remember, we have just loaded the flow control package by way of STARTUP.COM. On to the next package, another ZCPR3 system segment.

## Resident Command Package

Resident commands permanently (as long as the computer is on or until another RCP file is loaded) reside in memory, unlike transient commands, which are loaded and run from COM files. The ZCPR3 resident commands are loaded into memory from a disk file by LDR.COM. The resident command package is typically called SYS.RCP. If you have a suitable macro assembler, like Echelon's ZAS, or a friend with one, you can assemble different set of residents and load different resident command packages on the fly.

To see the available resident commands, type H*<cr>*. The display should look something like this:

```
B0:WORK>H
SYS 1.1A
```

```
                        CLS    CP     ECHO   ERA
                        LIST   P      POKE   R
                        REN    TYPE
                  B0:WORK>
```

These ten commands are resident in memory after this particular SYS.RCP has been loaded as part of the multiple command line passed to the system by STARTUP.COM.  Because of user preferences and hardware differences, various Z-System implementations will have somewhat different RCPs.

| COMMAND | MEANING |
|---------|---------|
| CLS | Clears screen. |
| CP | Single-file copy command. |
| ECHO | Sends output to the console, including escape sequences. |
| ERA | Erases files. |
| LIST | Prints a file or group of files on the printer. |
| P | PEEK. Used to examine an area of memory. |
| POKE | Changes contents of memory. |
| R | Resets disk system (logs in new disk). |
| REN | Changes the name of a file. |
| TYPE | Displays contents of an ASCII file or group of ASCII files on the CRT. |

Many of the resident commands are self-explanatory, but a few merit some brief discussion.

**CP**

CP copies a single file from one DU (Drive/User) to another DU or from one DU to the same DU with a different filename.  CP is very fast but does no error checking.  If you need or desire error checking, use the transient MCOPY utility.

Examples:

```
CP MYFILE.TXT=OLDFILE.TXT
```
Copies the file OLDFILE.TXT to a new file in the same DU called MYFILE.TXT.

```
CP WORK:=ROOT:MYFILE.TXT
```
Copies the file MYFILE.TXT from the ROOT DIR to the WORK DIR.

```
CP WORK:=A15:MYFILE.TXT
```
Copies the file MYFILE.TXT from DU A15 to the WORK DIR.

```
CP B0:=MYFILE.TXT
```
Copies the file MYFILE.TXT from the logged-in DU to DU B0.

```
CP B0:=A15:MYFILE.TXT
```
Copies the file MYFILE.TXT from DU A15 to DU B0.

**ECHO**

ECHO "echoes" messages to the CRT.  It can also send escape sequences to the CRT. These features can be used in aliases and other command files for some interesting visual effects.

Try this one:

```
ECHO ^Z I HAVE JUST CLEARED THE SCREEN
```

or:

16

```
ECHO ^Z
```

`ECHO ^Z` is the clear-screen command for our terminals. Your's may require a different string. If so, `ECHO [whatever your clear-screen string happens to be]` will work. Try sending some control strings to your terminal with the `ECHO` command. Are you getting some ideas for great video effects?

```
ECHO i am enjoying my z-system.
```

In fact, you just learned something about how ECHO converts lower case characters to upper case.

### ERA

ERA erases files. The syntax is `ERA dir:afn` (where afn means ambiguous file name). You can also enter `ERA dir:afn I`, where the I or inspect option offers yes/no prompting for each file in the list.

Examples:

```
ERA ROOT:*.* I
```
    Erase all files in the ROOT directory and prompt for each file--ask before erasing.

```
ERA B0:*.bak
```
    Erase all files in B0 with an extent of BAK-- do not ask.

```
ERA MYFILE.TXT
```
    Erase the file MYFILE.TXT--do not ask before erasing.

### LIST

LIST prints a single ASCII file or group of ASCII files on the LST: device (usually a printer).

Example:

```
LIST B7:*.ASM
```
    Prints all files in B7 with extent ASM.

```
LIST ROOT:MYFILE.TXT
```
    Prints MYFILE.TXT located in ROOT.

### P

The PEEK command shows bytes in memory. Typing "`P`" without specifying a memory address displays the next 256 bytes of memory. The command "`P EA00`" displays 256 bytes of memory starting at the address EA00. Another "`P`" displays the next 256 bytes. "`P EA00 EAFF`" displays memory from the address EA00 to the address EAFF.

### POKE

The POKE command changes bytes in memory.

POKE puts specific pieces of data into specific locations in memory. Be very careful with the POKE command because it can put any value anywhere in memory, including areas of memory containing operating system.

The format of the POKE command is `POKE ADDRESS VALUE`, where "`ADDRESS`" is an address in memory and "`VALUE`" is the value to be placed at that address. The command "`POKE 800 0`" puts the hexadecimal value 0 at the hexadecimal address 800H. `POKE 800 "THIS IS A TEST"` (the trailing quotation mark optional) places the ASCII values for the characters "THIS IS A TEST" at the memory location starting at

800H. `POKE 800 1 2 3 "THIS IS A TEST"` would put the values 1, 2, and 3 into memory starting at 800H, followed by the ASCII values for the characters "THIS IS A TEST" immediately after the 3. Note that once you start putting "text" into memory (using a quotation mark to mark the start of the text), everything that follows is interpreted as a text character. There's no going back to poking hexadecimal numerical values.

Let's confirm the POKE command does what it claims.

        `POKE 800 "Z IS A POWERFUL SYSTEM`*`<cr>`*

Now for confirmation:

        `P 800`*`<cr>`*

You should be able to see the ASCII values you just poked with the PEEK command--check the right side of the display, it's modeled after good old DDT. Of course, the POKE command works, but it's sometimes nice to see for yourself.

The POKE command is of enormous value to the Z-System user, since it permits the modification of programs in memory. For example, if you run WordStar, you know you can use the install program to make many different versions with many different default values. If you know where the bytes are in WordStar that control these defaults, you can load WS.COM in memory with the `GET 100 WS.COM` command and then poke those bytes with the appropriate values. If you had a way of poking many bytes with one command, you would be able to use just one copy of WordStar and poke the appropriate values into WS.COM at runtime. We will show you how to do that later on. Just remember for now that the technique we have been discussing is called the "Poke and Go" technique. It is one of the many creative things that users have done with the Z-System to save time, effort, disk space, and achieve higher levels of performance from their hardware and software.

## R

R resets the disk system. Ever get the dread BDOS error telling you a disk was R/O for read-only? Well, you should not get such an error with ZRDOS but you cannot be too careful about things. At least some computer users feel that way. R will make sure there is no question about your newly inserted disk getting logged in properly. It is also useful to make sure that disk space information provided by directory programs is accurate for a newly inserted diskette.

## REN

REN renames files.

`REN MYFILE.TXT=MY.DOC`
    renames the file MY.DOC to MYFILE.TXT.

You don't have to be in the same DU as the file to rename it. `REN ROOT:MYFILE.TXT=MYFILE` renames MYFILE in ROOT to MYFILE.TXT.

If you try to rename a file to the name of a file that already exists, you are asked if the file that already exists should be deleted. You can rename files set to system status but not files set to read-only status.

## TYPE

The resident TYPE command is used to display files on your CRT or "console".

`TYPE MYNOVEL.TXT`
    displays MYNOVEL.TXT on the console and pauses after a screenful, waiting for any keyboard input to continue with another screenful. This is called paging.

`TYPE MYNOVEL.TXT P`

displays the file without paging, just like the resident TYPE command under CP/M.

TYPE accepts an ambiguous file name, so you can display several consecutive files with one command, such as TYPE *.DOC, which displays consecutively all of the files of the type DOC, no matter what their name might be.


## Named Directories

The STARTUP program loads, as its last package, a set of named directories.

Named directories give names to Z-System directories. Z-System directories, as we have discussed, are logical directories known by a drive letter and a user area number. A0, B3, A15, and B0 are all Z-System directories. Using the named directory facility of ZCPR3, you may refer to each directory, such as A0, by a name of your choice. The named directory package on your distribution disk, SYS.NDR, gives names to two directories. A0 is called COMMAND and A15 is called ROOT. We picked those names; you may prefer others.

Computing is more comfortable and productive with named directories. When you pick names you may base your decisions on what feels comfortable to you. Many people like to call B0 WORK on their floppy based system. On a hard drive system you might want to segregate work areas according to what kind of task you will be performing in that area. B3 might be called ASM because that might be where you do your assembler work. You might want to pick your name as the name of a directory you do your work in if there are many people using the same computer. With a floppy system, you might want to tailor the names of your directories to the nature of the task as defined by the disk you are using in the A drive. So, for example, when we use our word processing disk, we might want to call B0 WORDPRO.

The utility that allows you to change the names of your directories is called **MKDIR**. MKDIR creates named directory files for subsequent loading by the LDR utility. Like all Z-System utilities, its use is explained in a built-in help file invoked by typing MKDIR //.

This is what MKDIR looks like when you invoke it:

```
B0:WORK>mkdir
MKDIR, Version 3.2
MKDIR Command (? for Help)? ?
MKDIR Commands are --
C -- Change Directory (Add/Rename/Delete Entries)
I -- Initialize Directory
P -- Print Directory
R -- Read Directory File
S -- Status of MKDIR Environment
W -- Write Directory File
X -- Exit Program
MKDIR Command (? for Help)? R
Name  of  File  (<RETURN>  =  B    0:  NAMES        .NDR)?
A15:SYS.NDR
MKDIR Command (? for Help)? P
DU : DIR Name - Password      DU : DIR Name - Password
---- --------   --------      ---- --------   --------
A  0: COMMAND  -              A 15: ROOT      -
B  0: WORK     -
MKDIR Command (? for Help)? S
** MKDIR Status **
3 Entries in Directory
Working File Name: SYS      .NDR
No Changes made to Directory since Startup
```

```
                File has been loaded
                MKDIR Command (? for Help)? X
                B0:WORK>
```

   This working session shows you how the MKDIR "R" command reads the .NDR file of your choice.  We selected SYS.NDR located in A15: or, in named directory parlance, ROOT.  The "P" command prints the named directories to your CRT.  If you want to change any of the named directories, or add new ones, you use the "C" command.  If you wanted to name A0 BASE instead of COMMAND, you could type A0:BASE (in the "change" mode invoked by the "C" command of MKDIR) and the name would be changed.  In order to save your changes and additions to a file, you exit the "change" mode with the "X" command followed by a carriage return and then use the "W" or write command.  Since you have already told MKDIR that you are dealing with SYS.NDR, located in A15, you would just hit a carriage return after the "W" command to save the new version of SYS.NDR to your disk in drive A.  To load the new named directories into memory you type LDR SYS.NDR.  And you have your newly defined set of named directories.

   Before we end our discussion of MKDIR lets take a look at a session where the directory was changed:

```
A0:REMOTE>mkdir remote.ndr
MKDIR, Version 3.2
MKDIR Command (? for Help)? P
DU  :  DIR Name - Password DU  : DIR Name - Password
----   --------   -------- ---- --------   --------
A  0:  SYSTEM    -               A 1: SB180     -
A  2:  TEXTPROC -               A 3: ASSEMBLE -
A  4:  ZRDOS     -               A 5: DRICPM    -
A  7:  SYSOP     - HORRORS! A 8: PUBLICS   - WONTWORK
A 15:  ROOT      - NOTHERE!
B  0:  UPLOADS  -               B  1:          ZCPR3     -
B  2:  UTILITY  -               B  3:          COMPILE   -
B  4:  DOCUMENT -
MKDIR Command (? for Help)? C
** MKDIR Change Mode **Directory Entry (?<RETURN> for Help)? a1:naogzsig
Renaming SB180
Directory Entry (?<RETURN> for Help)? x
MKDIR Command (? for Help)? X
Directory has changed since last Write
Do you want to write Directory to Disk (Y/N)? Y
Name of File (<RETURN> = A  0: REMOTE  .NDR)? test.ndr
Writing Directory to Disk ... Done
```

   Notice how MKDIR asks you about the output filename before creating the new NDR file.  If the user had answered the "Name Of File" query with a carriage return, the original REMOTE.NDR in A0 would have beend overwritten.  By initially creating the new named directory file as TEST.NDR, the user is wisely leaving the original file alone until he tries out the modified version, after which he can use the REN or CP command--or a utility program--to change the names around.

   The PWD utility allows you to see what named directories you have defined and loaded into memory.  This is what PWD looks like when you invoke it.

```
B0:WORK>pwd
PWD, Version 1.0
DU : DIR Name     DU : DIR Name     DU : DIR Name     DU : DIR Name
---- --------     ---- --------     ---- --------     ---- --------
A 0: COMMAND      A 15: ROOT
B 0: WORK
B0:WORK>
```

We see that there are three working directories loaded from our disk:  COMMAND, ROOT, and WORK.  Each named directory is associated with a drive and user area.  These drives and user areas are listed before the name under PWD.  Essentially, PWD combines the functions of the "R" and "P" commands of MKDIR and allows you to take a quick glance at the directories currently active in case you have forgotten.

**CD**, the change directory command, gives you another way of moving from named directory to named directory.  You have seen that we can move from B0 to A15 by typing A15:*<cr>*. We can do the same thing by typing ROOT:*<cr>*.  CD allows us to type CD A15:*<cr>* or CD ROOT:*<cr>* to accomplish the same thing.  You may wonder why we would want to go through the trouble of typing CD in addition to the other stuff, when we could just type ROOT:*<cr>*.  Without going into any deep under-the-hood stuff, one advantage of using CD to move to a new named directory is that when CD takes you into the new directory, the first thing it does is look for a file called ST.COM.  If it finds ST.COM, it runs it.  ST.COM can be any program you pick, as long as you rename it to ST.COM.  It can also be (and usually is) an alias. An alias is a COM file--like STARTUP itself (the name ST comes from "STartup")--that simply passes a series of commands embedded in it to the system for execution.  So, you can have an automatic series of commands executed when you log into a new named directory with CD.  We will have more to say about aliases later.

Named directories allow you to compute the way you think.  You may find it difficult to remember that you do your assembler work in B3, but you will have less difficulty remembering the named directory called ASM.  Let's say you want to copy certain files from you ASM named directory to your BACKUP directory.  It is easy to remember that your drive and user area for backups is called BACKUP and that your drive and user area for assembler work is called ASM.

To copy a file called BIGPROG.ASM from your ASM directory to your BACKUP directory you would simply type:

```
CP BACKUP:=ASM:BIGPROG.ASM
```

The resident CP command does no error checking.  Modern hardware is quite reliable and many users trade off error checking for the speed of CP.  If you wanted to do automatic error checking on the read and write process you could use the MCOPY program and type:

```
MCOPY BACKUP:=ASM:BIGPROG.ASM
```

The Z-System directory utilities also know about named directories.  To see what files are in your ROOT directory, you might type XDIR ROOT:, XD ROOT:, or just DIR ROOT:, according to which Z-System directory utility you are working with.  The other Z-System utilities know about named directories as well.  Named directories provide you with a powerful tool to bring computing into the realms of normal everyday English.  This is just another example of how the Z-System provides the most practical and natural user interface available today on microcomputers.

Well.... Here we are.  At the end of STARTUP.  And we have just started our journey into Z.  At this point you should feel comfortable with the concepts of flow control, resident commands, and named directories.  We will move on now to other important matters, including the intrinsic commands, the multiple command line, and aliases.

# OTHER Z-SYSTEM COMMANDS

## Intrinsic Commands

ZCPR3 itself contains certain "built-in" commands wholly separate from the flow control and resident commands. They exist whether or not flow control and resident command packages are loaded. We call them "intrinsic" commands.

Your system contains the following intrinsic commands:

**GET**      - Load a file into memory.
**GO**       - Re-execute the last transient command.
**JUMP**     - Branch to an address in memory.
**NOTE**     - Make a comment (disable further command interpretation).
**SAVE**     - Write memory image to disk.

These commands are the cornerstones of some of the most powerful Z-System applications.

### GET

The GET command is used to load a file anywhere in memory. As a practical matter, since COM files are always loaded at the address 100H you will use the GET command to load COM files at their running address. Thus, the command `GET 100 WS.COM` loads WordStar at its running address of 100H. In combination with the POKE resident and the GO intrinsic command, GET provides the capability of modifying a program on the fly, just before running it. (We'll develop the WordStar application more fully in a moment.)

### GO

The GO command is used to re-execute the last program loaded into memory at 100H without having to reload it. If you type "`DIR`", and, after viewing the directory listing, type "`GO`" you will reexecute DIR.COM. "Big deal," you say? Big deal indeed! In addition to giving you a quick and easy way to re-execute programs, the GO command is another cornerstone of the poke and go technique.

The poke and go technique permits you to have one copy of WordStar on your word processing disk and, yet, be able to load many different versions of WordStar with many different default values. Assuming WS.COM and its overlays are on drive A, you could enter the following series of commands:

```
A0:COMMAND>GET 100 WS.COM
A0:COMMAND>POKE 36B 4D
A0:COMMAND>POKE 36E 0
A0:COMMAND>GO
```

This series of commands loads WordStar at its running address of 100H, changes the byte at address 36BH to 4DH, changes the byte at address 36EH to 0, and executes (runs) WordStar as modified in memory. If you know the addresses of your WordStar patch points (and you can find this by calling your local Z-Node for a file listing the patch points of your version of WordStar) you can use this poke and go technique to patch WordStar while it is loaded in memory, just before you run it. Goodbye WINSTALL.COM! Of course, there are more efficient ways to handle the poke and go technique than manually, at the command prompt, as we shall see later.

## JUMP

The JUMP command is used to branch or "jump" to any address in memory. Here's an interesting demonstration: type "JUMP 0" and hit a carriage return. You have just used JUMP to warm boot your system! The point is that the JUMP command can be used to branch to any address in memory and, if it finds code there to execute, it will start executing it. An even easier way to induce a warm boot with JUMP is just to type "JUMP" with no operand (command tail) at all, the "0" is implied. The code found at location 0 is a vector (8080/Z80 JUMP instruction) to the BIOS routine that warm boots (reloads) ZCPR3.

## NOTE

The NOTE command allows you to type comments without getting error messages from the system. Try this:

```
THIS IS A NOTE<cr>
```

What happens? You get an error message. Now try this:

```
NOTE THIS IS A NOTE
```

No error message! Why? The NOTE command has, in effect, disabled command interpretation. The system no longer thinks you are trying to execute a command called "THIS". The NOTE command allows you to comment batch files without provoking error messages. Although semi-colon at the beginning of a command line also disables command interpretation, as under CP/M, NOTE is more useful because it can be included on a multiple command line.

## SAVE

The SAVE command is used to save the contents of the transient program area (TPA) to a disk file. We use it primarily in the modification of programs with a debugger, such as Echelon's ZDM and DSD, or Digital Research's DDT. This is primarily a command for advanced users who are modifying their system or "patching" COM files.

SAVE accepts two arguments (which is to say that there are two pieces of information you can key in after the word "SAVE" to tell it exactly what to do): the number of pages to save and the file name to which you want the pages of memory saved. If the file name is followed by an "S" then the Z-System assumes that you want to save a certain number of 128 byte sectors. If the "S" is omitted, the number refers to the number of 256 byte pages you wish to save.

Let's look at an actual terminal session using the SAVE command in the creation of a file for SYSGENing a new system. Assume that the existing operating system is in a file called Z3TB58-3.SYS. (In case you are curious, the authors have Advent/Plu*Perfect TurboROMs installed in their Kaypros, so the Z3TB part of the file name refers to a Z3 system for the TurboROM and the 58 reminds us this is a 58K system.) After overlaying a modified version of ZCPR3, we will create a file called Z3TB58-4.SYS, which contains our modified operating system. Here goes:

```
B0:WORK>DDT Z3TB58-3.SYS
DDT VERS 2.2
NEXT  PC
2500 0100
```

```
                         -IZCPR3.HEX
                         -R3F00
                         NEXT  PC
                         2500 0000
                         -G0
                         Warm Boot
                         B0:WORK>SAVE 36 Z3TB58-4.SYS
```

Let's look closely at what we have done.

First, the Digital Research Dynamic Debugging Tool (DDT), as supplied with most CP/M-based computers, is invoked. We ask DDT to load the file Z3TB58-3.SYS.

Second, we use the `"I"` command to set up a file control block for the file ZCPR3.HEX. In non-computer parlance, we have told the system that we are about to read in the file ZCPR3.HEX.

Third, we use the `"R"` command to read in the file ZCPR3.HEX at the appropriate offset for our system. On this particular system, for a Kaypro with an Advent TurboROM, we use an offset of 3F00. The appropriate offset depends on what system you are generating.

Fourth, we use the `"G0"` command to warm boot the system. At this point, the file Z3TB58-3.SYS is in memory, overlaid by the file ZCPR3.HEX at the appropriate offset.

Finally, we use the SAVE command to SAVE 36 256-byte pages to the file called Z3TB58-4.SYS.

How did we calculate how many pages to save? The answer is simple. Look at the hexadecimal number under the NEXT after DDT loads Z3TB58-3.SYS. It is 2500. To calculate the number of pages to SAVE, take the two leftmost numbers and convert them to decimal. 25 hexadecimal or 25H equals 2 X 16 plus 5 X 1. The answer is 32 plus 5 or 37. If the number under NEXT ends in two zeroes, we subtract 1, so the number of pages to save is 36. If the last two numbers under the NEXT were not two zeroes, we would not have subtracted 1. This calculation is essential to any significant work in modifying the operating system. But...

Z-System even makes this easy--no calculation required if an "h" is added to the hex base page count. SAVE 25h Z3TB58-4.SYS<cr> does the trick!


## Line Editing and Output Control

ZRDOS supplies the line editing functions of the Z-System. The line editing functions are the commands you have available to edit a command line when you are at the system prompt, such as `A0:COMMAND>`. These are your line editing functions:

`DEL`      Delete the last character typed at the console. This is the same as BACK SPACE.
`Ctrl-H` Delete the last character typed at the console and backspace one character.
`Ctrl-U` Delete the entire command line.
`Ctrl-X` Delete the entire command line and backspace to the beginning of the current line.
`Ctrl-E` Marks physical end of line. The cursor is returned but command line is not sent until RETURN key is hit.
`Ctrl-M` Terminates input with a carriage return.
`Ctrl-J` Terminates input with a line feed.
`Ctrl-C` Z-System reload ("warm boot")

The following are the output functions:

`Ctrl-P` Copies all following console output to the LST: device. The output goes to the LST: device (your printer, usually) and the console until the next Ctrl-P is hit.

(It is worthwhile to note that many ZCPR3 and ZRDOS tools output characters via BIOS rather than ZRDOS.

The Echelon package IOREC (if your Z-System supports IOP segments) or a public domain utility like SWAP or REDIR is recommended for capturing BIOS console output to the LST: device or a disk file.)

`Ctrl-S` Stops console output until next character is typed. Typing Ctrl-C returns control to the Z-System.

Your input lines can be quite long. On the distributed system the input line can be about two and one half screen widths. You can have such a long input line because the Z-System allows you to put multiple commands on an input line, separated by a semi-colon (";").

## The ZCPR3 Multiple Command Line

The multiple command line, with the many techniques of bringing a series of commands to the attention of the command processor, is another linchpin of the Z-System.

What is the multiple command line? ZCPR3 allows the user to put a series of commands on one command line, separated by a semi-colon. For example:

```
A0:COMMAND>WORK:;DIR *.BAK;ERA *.BAK;DIR;COMMAND:
```

is a valid command line. ZCPR3 interprets this command line to be a user request to log into the named directory called WORK, show a directory listing of files of the type BAK, erase all such files, and show another directory listing to insure that all such files have in fact been erased, then finally back to the original logged named directory, COMMAND.

The multiple command line bestows a number of benefits.

First, you can key in a series of complex commands, hit the return key, and leave the system to do something else until all the commands have been executed.

Second, you can put any multiple command line (up to the buffer space allowed, which is about two and one-half average screen widths) into a simplified batch processing file and reduce a lengthy, complex command to one keystroke. Alias files put the multiple command line into an actual running program. The ZEX batch processor works with the multiple command line as well and Z-System menus achieve much of their power from the ability to reduce a multiple command line to one user keystroke.

Thus, the multiple command line, in conjunction with various forms of <u>scripts</u> gives you the kind of power undreamed of under CP/M. A script is a sequence of commands stored on disk that can be executed as if it were just one command. Thus, the sequence of commands:

```
A0:COMMAND>WORK:;DIR *.BAK;ERA *.BAK;AC BACKUP:=*.TXT;ERA
*.TXT;COMMAND:
```

can be reduced to the single command script, `CLEANUP`. We will see how to do that with **ALIAS** files in the next chapter.

# ALIASES

## What is an Alias?

An alias is another name for someone or something. "Bill Jones," we say, "Also known as the 'Masked Man'." The "Masked Man" is Bill Jones's alias or other name.

Similarly, a Z-System **ALIAS** is also another name for something. The "something" is a series of commands ultimately sent to the command line buffer for processing by ZCPR3.

Strictly speaking, an alias is a COM file, created by the ALIAS utility, containing one or more commands, separated by semi-colons, which are passed to the command line buffer.

The ALIAS utility puts virtually any series of commands into a COM file, limited only by the size of the command line buffer. Whenever you run the COM file created by the ALIAS utility, the same series of commands will be processed by the ZCPR3 command processor. This series of commands has an alias, another name, and that other name, the alias, is the name of the COM file you have created. What once had many names is now reduced to a single name. What once took a great deal of typing now takes only the amount of typing needed to key in the name of the alias COM file and it command tail, if any.

The series of commands stored in the COM file are called scripts and the COM file in which the script is stored puts the script into the command line buffer when the name of the alias is used as a Z-System command. The COM file can also pass parameters from the command line in a manner much like CP/M's SUBMIT--with extensions - and the fully expanded (actual command line parameters substituted for the formal symbols) script will be executed. This will become clearer when we discuss the details of implementing aliases.

## The ALIAS Utility

The ALIAS utility creates a COM file that passes a command or series of commands to the operating system. The alias utility is invoked as follows:

```
ALIAS            -- define an alias COM file
ALIAS filename  -- display the commands stored in "filename.COM" and optionally edit
                         those commands.
```

What follows is an actual terminal session demonstrating the use of the ALIAS utility:

```
B0:WORK>ALIAS
ALIAS, Version 1.1
  Input Alias (RETURN to Abort)
  --> WORK:;DIR *.BAK;DIR *.REL;ERA *.BAK;ERA *.REL;DIR
```

26

```
  Name of ALIAS Command (RETURN to Abort)? CLEANUP
  Alias Created
B0:WORK>
```

We have now created the alias CLEANUP, consisting of a series of commands, separated by semi-colons.  Now we can use the ALIAS utility to examine the commands stored in CLEANUP:

```
B0:WORK>ALIAS CLEANUP
ALIAS, Version 1.1
Alias Name: CLEANUP
Old Alias Command Line:
1 --> WORK:;
2 --> DIR *.BAK;
3 --> DIR *.REL;
4 --> ERA *.BAK;
5 --> ERA *.REL;
6 --> DIR
Input Alias (RETURN to Abort)
-->
B0:WORK>
```

The ALIAS utility shows each command on a separate line for the convenience of the user.


## ALIAS Variables


Below is a summary of the variables (formal parameter symbols) you may use within the body of an alias:

```
$0Name of Alias
$nParameter from Command Line (where 'n' is an integer from 1
     through 9)
$*Tail of Command Line (everything that follows the verb)
$DHome Disk (drive you started from)
$UHome User (user area you started from)
$Fn   FILENAME.TYP of System File 'n' (where 'n' is an integer
     from 1 through 4)
$Nn   FILENAME of System File 'n'
$$The character $
```

A parameter is whatever you type in after the verb. The verb is the command itself.  When you type WS to invoke WordStar, WS is the verb.  When you type WS MY.DOC, MY.DOC is a parameter.  The parameter follows the verb and tells WordStar which file you want to edit.  The variable $0 is the name of the alias itself.  The variable $* is the tail of the command line --everything after the verb, regardless of the number of parameters.

The system files referred to above are names of files that are stored in memory.  There are four such names permitted by ZCPR3.  The SETFILE utility defines these names.


## Examples of Aliases


Now, let's look at a bunch of aliases from a number of users on various levels of Z-System expertise.  Some will be presented as actual ALIAS.COM screens, others will be simple lists of commands that you can make into aliases with ALIAS.COM, using the standard semi-colon command separator.

**Example Number 1:**

You are getting tired of typing the same commands over and over again to assemble and load a file:

```
ASM myfile.bbz
LOAD myfile
```

The Z-System solution is to make an alias:

```
ASM $1.BBZ;LOAD $1
```

Let's call the alias ASSEMBLE.COM. Now when you assemble a file called TEST.ASM, you type `ASSEMBLE TEST`. That is the equivalent of typing `ASM TEST.BBZ;LOAD TEST`, but you have saved keystrokes.

**Example Number 2:**

You have a hard drive system and you want to be able to backup only those files that have changed in a particular named directory. Try this alias:

```
AC BACKUP:=$1:*.* /A
```

You might call this alias ARCHIVE.COM. If you type "`ARCHIVE TEXT`" the archival backup utility, AC, will copy those files in the named directory TEXT whose archive bit has not been set (i.e. new files in that directory) to the BACKUP named directory.

**Example Number 3:**

Every time you install a utility you have to type "Z3INS ROOT:SYS filename.COM", where "filename" is the name of the new utility. What about an alias to cut down on keystrokes?

```
Z3INS ROOT:SYS $1.COM
```

Let's call the alias I.COM. Now, to install a utility, filename.COM, type "`I filename`" and you are done. With the Z-System, a little thought now saves a lot of thought later.

**Example Number 4:**

You are running WordStar, you use XtraKey to load your reconfigured keypad, you like 78 columns as default, word wrap off, and hyphen help off. Watch this:

```
ROOT:;
XK WS.XPD;
$D$U:;
GET 100 COMMAND:WS.COM;
POKE 36B 4D;
POKE 36E 00;
POKE 34F 00;
GO;
XK $$X
```

This alias, which we call WSTAR.COM, logs into the ROOT directory, invokes XtraKey, loads the XtraKey file with our particular keypad configuration, returns to the DU we were logged into when we typed WSTAR, loads WS.COM into memory at its running address of 100H, pokes the appropriate values into WordStar to tailor the defaults to our liking, and starts WordStar running. After we finish with WordStar, XtraKey removes itself from memory and the alias is completed.

**Example Number 5:**

You want an easy way to check the contents of a diskette in drive B, regardless of whether it is double- or single-sided, examining all user areas. Remember that the ZCPR3 intrinsic JUMP command causes a warm boot when invoked with no operand, see 5.1.

```
JUMP;XDIR B:*.* U
```

Call this alias something like DIRB.COM.

28

**Example Number 6:**

This is an edited version of a user-submitted alias called NEWDISK.COM.  When you make a new system disk you will want certain files on it without question--such as your STARTUP.COM, your packages, LDR.COM, and perhaps other files.  Why key in all those file names?

```
B0:WORK>ALIAS NEWDISK
  ALIAS, Version 1.1
  Alias Name: NEWDISK
  Old Alias Command Line:
    1 --> CP B:=A:SYS.ENV;
    2 --> CP B:=A:LDR.COM;
    3 --> CP B:=A:STARTUP.COM;
    4 --> CP B:=A:SYS.FCP;
    5 --> CP B:=A:SYS.RCP;
    6 --> CP B:=A:SYS.NDR;
    7 --> CP B:=A:PWD.COM;
```

The alias uses the RCP-resident CP command to copy all files this user considers essential on system disks.

**Example Number 7:**

Another one from the same user.  He hates to key in a series of commands to perform compilations with his C compiler (yes, some people use high level languages even though all of us who aspire to write efficient CP/M-compatible programs should be be learning modern assembler techniques--with Richard Conn's SYSLIB it just isn't that hard).  This is his alias:

```
  B0:WORK>ALIAS COMPILE
  ALIAS, Version 1.1
   Alias Name: COMPILE
   Old Alias Command Line:
    1 --> CC $1 $2 $3 $4;
    2 --> A:M80 =$1;
    3 --> L80 $1,A:CRUNLIB/S,$1/N/E;
    4 --> ERA $1.MAC;
    5 --> ERA $1.REL
```

Assuming that this user does his work in C language while logged into the B drive, can you detect an unnecessary command here?  What powerful element of the Z-System is this user forgetting?  PATH search ability, of course!  In line 2 he does not need the 'A:M80'.  His search path is A0 A15.  Even though he is logged into drive B, the command processor will search drive A, user 0 for the file M80.COM.  Another suggestion for new Z-System users.  Forget your old habits.  Expand your thinking to include new time-saving shortcuts.

**Example Number 8:**

This one is from an advanced user.  In fact, he is a major Z-System contributor.  In order to assemble the ZCPR3 utilities, he recommends the following:

```
  M80 =$1;L80 $1/P:100,VLIB/S,Z3LIB/S,SYSLIB/S,$1/N/E
```

Let's look under the hood at his ALIAS, called M8.COM.

You run M8 by entering M8 NEWUTIL, where NEWUTIL is a file of the type MAC.  A MAC file is an assembler source file formatted for assembly by Microsoft's M80 assembler.  (Many Z-System users have replaced M80 with the fine ZAS assembler distributed by Echelon, and ZAS is certainly a good choice if all you have is ancient ASM.COM.)

What does M8 do and how does it do it?  The first command passed by the ALIAS is M80 =$1.  If you type M8 NEWUTIL the alias will know that the first parameter is NEWUTIL and will expand the command to M80 =NEWUTIL.  This command assembles the file NEWUTIL.MAC, creating a file NEWUTIL.REL.  The linker, L80, is called and NEWUTIL

is linked (the ALIAS substitutes NEWUTIL for $1 during its expansion), with a starting address at 100H, importing necessary routines from the three standard Z-System libraries: VLIB, Z3LIB, and SYSLIB.

**Example Number 9:**

You have two printers--a parallel Okidata dot matrix and a serial printer. You are using the version of Z-System with redirectable I/O and you have two versions of WordStar--one for the parallel Okidata printer and one configured for the serial printer. Watch this (commands are on the left):

```
SCRIPT          MEANING
IF OKI=$2       Check to see if 2nd param is OKI
DEV L LPT       If so, assign LST device to LPT
WSO $1          and run OKI version of WS
ELSE            If not...
DEV L TTY       Assign LST to TTY
WST $1          and run TTY version of WS
FI              and set flow state true
```

Call this ALIAS something like WSTR.COM. If you type `WSTR NEW.DOC`, there is no 2nd parameter, so `OKI=$2` is false and the alias will expand to `DEV L TTY;WST NEW.DOC`. If you type `WSTR NEW.DOC OKI`, `OKI=$2` (where $2 expands to OKI) is true and the alias will expand to DEV L LPT;WSO. The parallel Okidata version of WordStar will run. Of course, you will need to implement an IO package to run this ALIAS, as well as the appropriate BIOS mods, but if you have a system with IOP's you are in business.

## Explorations

The alias scripts listed above are just a few of the myriad possibilities. Any time you find yourself using repeated keystrokes for any process, or any time you find that you have forgotten some complex command and have to look it up, THINK ALIAS--begin to look at programs as tools, linked together by command scripts to perform tasks. The alias facility is one of the most powerful aspects of the Z-System and one that will assist you to achieve maximum efficiency in your computing.

# ZEX, THE MENUS, AND VFILER

## Introduction to Advanced Command Processing

In this chapter we will take a look at four of Z-System's most powerful--some might say intimidating--programs: ZEX, the memorybased command file processor, MENU and VMENU, the most popular "front ends" for the Z-System, and VFILER, which combines screenoriented file management with user-programable menu-style options. We will not attempt to document every feature of these tools; that is done comprehensively in **ZCPR3: The Manual** and the ZCPR3 Help File System. Our sole goal is to convince you to try your hand at mastering them. They were designed specifically to give the Z-System user a means to customize the computerized workplace, tailor it to individual requirements. They provide programability, without programming in the conventional sense. If approached with careful thought and an inquiring mind, they allow you to personalize your computing environment to an extent no other microcomputer operating system can touch.

## ZEX

ZEX is a memory-based command file processor, combining the functions of CP/M's SUBMIT and XSUB, but adding speed and power far beyond what those programs offer under CP/M. ZEX is often preferable to an alias as a command script execution tool for a number of reasons:

1. The length of command scripts in ZEX files are not limited to the size of the ZCPR3 multiple command line buffer. The expansion of command line parameters within an alias makes it relatively easy to come up against this restriction.

2. ZEX has many more features and options than are offered by ALIAS (or, for that matter, SUB). Screen displays and user prompts are customizable into a very polished user interface.

3. ZEX can feed scripted commands to an interactive program as well as to the ZCPR3 command line. ZEX can provide input to a text editor, word processor or debugger. For example a ZEX file could invoke WordStar, feed WordStar the internal commands for a find-and-replace operation, exit to system level, invoke MCOPY or CP to put copies of the modified file into a number of designated DIRs, load a system image with DDT or Echelon's ZDM, perform a patching operation from inside the debugger, exit and do a sysgen operation, all unattended.

4. ZEX has more powerful conditional execution capabilities than ALIAS or SUB, allowing true forward-only conditional branching through use of GOTO.COM, a special ZCPR3 tool. This feature adds another dimension to the use of flow control in command

scripts.

5. ZEX allows default parameters to be established, using them in expanding the script if the corresponding parameters on the originating command line are not supplied.

6. Like SUB, but unlike an alias, ZEX execution may be stopped easily from the keyboard with a Control-C.

ZEX has two modes: interactive mode and command file mode.

To enter the interactive mode, type ZEX<cr>. In the interactive mode, the user enters command lines until there are no more commands desired to be entered. At this point, entering an empty command line (hitting RETURN on an empty line) will cause ZEX to start executing commands in the order they were entered. You cannot pass parameters from the initial command line in the interactive mode, but you can establish them in mid-session. Here is one of ZCPR3 author Richard Conn's demonstrations of ZEX in interactive mode. You can get a "feel" for ZEX by just typing what Richard did at your own keyboard.

**Example Number 1:**

This example illustrates ZEX in the interactive mode.

```
B0:WORK>zex
ZEX, Version 3.1
1: ^$ this is fun              <-- Define 3 parameters
2: echo $1 $2 $3
3: ^$ hello from happy acres   <-- Define 4 parameters
4: echo $1 $2 $3 $4
5:
(ZEX Active)                   <-- ZEX is running now

B0:WORK>echo this is fun
THIS IS FUN
B0:WORK>echo hello from happy acres
HELLO FROM HAPPY ACRES
B0:WORK>
(ZEX Completed)
By Your Command >
```

Let's examine what is done in the terminal session example. First, ZEX is invoked with no command tail. The ^$ ZEX directive (two characters, not Control-$) designates the subsequent entries, separated by spaces on a single line, as ZEX parameters. The RCP ECHO command then is told to display the these newlyestablished parameters. Next we use "^$ again to name four new parameters and use ECHO to display then. Entering a RETURN (carriage return) on an empty line tells ZEX to begin processing the commands, which it does, executing both ECHO commands and "hiding" the ZEX directives from the screen.

To enter the command file, or batch processing, mode, enter "ZEX filename parameters", where "filename" is any ZEX batch file with the extent of SUB or ZEX and "parameters" are any appropriate user supplied parameters. In this mode, ZEX will search along the path for filename.ZEX or filename.SUB and, if it finds such files, it will begin processing them. If a directory in the user's path contains both a ZEX and a SUB file of the same name, ZEX will process the ZEX file.

ZEX may be aborted at any time with the entry of Control-C from the console. ZEX does not support nesting, so if the ZEX processor encounters another invocation of ZEX in a batch file, it will abort. Many powerful, embedded, extended control commands are provided, making ZEX considerably more effective than CP/M's XSUB when acting as a "robot

console" substitute for user input to application programs.

The following ZEX command file uses ZCPR3 flow control to choose between two assemblers, ASM.COM, the standard CP/M assembler, and Echelon's ZAS.  By the way, a powerful, modern assembler like ZAS is a must if you want to tinker "under the hood" of Z-System and its utilities.  The MLOAD program is a highly enhanced substitute for the CP/M LOAD command by NightOwl Software and is available free, by modem from Z-Nodes or from users groups, including NAOG/ZSIG.

**Example Number 2:**

This file uses the filetypes *.ASM and *.Z80 to identify two different dialects of assembly language and chooses the appropriate assembler.  An output filename different from that of the source file may by specified on the originating command line.

```
;
;   Echelon ZAS 2.2a and DRI ASM 2.0 Assemblers
;         with NightOwl MLOAD 2.4 Loader
;
;     ^& Suppress FALSE IF Printout
;
IF NUL $1
ECHO  ** File Name Required for Assembly **
ELSE
IF EX $1.Z80    ;NOTE Zilog Mnemonics?
ZAS $1 H        ;NOTE Perform Assembly With ZAS
ELSE
ASM $1.BBZ      ;NOTE No, Use ASM (hope for no macros)
FI
IF INPUT   Abort if Errors Exist (^C), <cr> to Load
ERA $1.BAK      ;NOTE Cleanup BAK File
IF ~NUL $2      ;NOTE There's An Output Filename?
ERA $2          ;NOTE Then Erase Its Current Owner And
MLOAD $2=$1     ;NOTE Load Hex File To That Filename
ELSE
ERA $1.COM      ;NOTE No, So Just Cleanup Old COM File
MLOAD $1        ;NOTE And Load Hex File Normally
FI
FI
ERA $1.HEX      ;NOTE Cleanup HEX File In Any Case
FI
;
;   Operation Complete
;
```

Lets's step through the file, which is called ASMBL.ZEX.  All the lines beginning with semicolons and all NOTE commands will display to the screen but not cause any action by ZCPR3.  The ZEX directive, ^&, is obeyed by ZEX even though it is ~preceded by a semi-colon.  It tells ZEX not to output any lines to the screen when the IF state is FALSE.  The first line checks for the absence of a command line operand and prints an error message if there is none.  Notice that the IF state is then toggled to FALSE with an ELSE command, so if there is no operand only flow commands will execute.  If the operand exists, it is used as the filename of following IF EX command.  If you executed the file with `ZEX ASMBL FILE` this line would expand to `IF EX FILE.Z80`.  The *.Z80 filetype is for Zilog-mnemonic source files, which are handled by ZAS.  If FILE.Z80 does not exist, then the following ELSE command assumes (with some risk) that FILE.ASM exists, a file using Intel mnemonics to be assembled by ASM.

Next comes an FI (ENDIF) command which exits the IF level based on file types. Either assembler produces a *.HEX output file, so at this point we should have FILE.HEX and some kind of screen message indicating success or failure of the assembly process. This is where the IF INPUT command, with its questioning console message, asks for a go-ahead (<cr>, a carriage return) or an abort signal (^C, a Control-C). If the go-ahead is issued, the next command erases the backup (*.BAK) file left by the editor program that produced the source file. The IF ~NUL $2 test that follows means, in effect "if there is a second command line parameter", and, if so MLOAD will be instructed to use it as the output filename. This would allow you to assemble FILE.Z80 or FILE.ASM into MYFILE.COM or somesuch. Without the second parameter, the name of the source file would be used for the output *.COM file. One exercise left to you will be tracing the flow of the script when the all-important first parameter is missing--suppose you just type ZEX ASMBL, what would happen and why?

It is important to realize that the number of characters involved in ASMBL.ZEX precludes it being implemented with ALIAS, so a major advantage of ZEX should be apparent now. ZEX can handle long scripts with verbose console messages, ALIAS can not.

ASMBL.ZEX makes very little use of ZEX's special internal directives--it is basically a flow control exercise that could have been implemented with SUB, at a substantial speed penalty. The next example, slightly edited from one written by ZRDOS author Dennis Wright, shows how nicely ZEX can control the console display and illustrates the power of ZEX internal directives. It is a sophisticated example of ZEX usage, built around the ability of the ZAS assembler to "report" assembly errors to ZCPR3 via the program error flag, a reserved byte in the ZCPR3 message buffer area of memory. It is also an illustration of the way most ZCPR3 and ZRDOS utilities are built --the mainline assembly language source file is assembled to a relocateable object module, which is linked with subroutines extracted by the linking loader, ZLINK, from subroutine libraries. The Echelon publication, ZCPR3: The Libraries, explores this technique in more detail.

### Example Number 3:

This is a conditional assembly script using flow control with the Echelon Z-System assembler, ZAS, and linking loader, ZLINK. The "structured" style of this file should make it easy to follow the various levels of active IFs involved. The further to the right a command begins, the higher the active IF level is.

```
^.
^#
^<^|
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;^|
;                                                         ;^|
;ZASLINK.ZEX -- Z-System ZAS Macro Assembler and ZLINK Linker;^|
;              with ZAS error checking.                    ;^|
;                                                         ;^|
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;^|
^>
^&
if ~exist $1.Z80    ;note if file does not exist^|
   ECHO ^G          *** $1.Z80 NOT FOUND ***^|
else^|
   ZAS $1^|
   if ~er              ;note if no assembly errors^|
     ERA $1.COM^|
     ZLINK $1,A:VLIB/,A:Z3LIB/,A:SYSLIB/ $$C100^|
     ERA $1.REL^|
     ERA $1.BAK^|
     Z3INS A15:SYS $1.COM^|
     ECHO              ZASLINK COMPLETE^|
```

35

```
    else^|
        ECHO ^G *** FATAL ERROR IN ASSEMBLY, ZASLINK ABORTED ***^|
    fi                ;note if ~er^|
fi                    ;note if ~exist^|
^#
```

Rather than go through ZASLINK.ZEX step-by-step together, it is probably more useful at this point to examine it yourself while referring to the following summary of the extended control commands available under ZEX. The flow control is straightforward, and more automated than ASMBL.ZEX because it uses ZAS's ability to tell ZCPR3 when there is an assembly error.

| Cmd Meaning | | Cmd Meaning | |
|---|---|---|---|
| `|` | insert <CR> | `^|` | insert <CR> <LF> |
| `^:` | rerun command file | `^.` | suppress print of chars |
| `^#` | toggle ZEX msgs | `^$` | define default params |
| `^?` | wait for user <CR> | `^/` | ring and wait for <CR> |
| `^*` | ring bell | `^"` | allow user input |
| `^<` | display chars only | `^>` | stop display |
| `$&` | suppress displays when IF state is FALSE | | |
| `;;` | ZEX comment | `$n` | 1<=n<=9 for param |
| `$$` | =$ | `$^` | =^ |
| `$|` | =| | `^c` | insert ctrl char c |

These commands may be embedded in the text of the command file or supplied by the user in the interactive mode and they will be executed after processing by ZEX commences.

The following ZEX commands are character insertion commands. That is, they are used merely to insert characters into the ZEX command stream:

```
|   inserts a <CR>         ^|  inserts a <CR> <LF> pair
$$  inserts a single $     $^  inserts a single ^
$|  inserts a single |     ^c  inserts a control character
```

`^*` - causes ZEX to ring the console bell

`;;` - a ZEX comment

`^<` and `^>` - characters between these "brackets" are echoed to the user during execution, but are not processed as part of the command stream

`^#` - toggles suppression of informative ZEX messages

`^.` - causes console output to cease until the next `^.` is encountered

`^:` - causes ZEX to restart execution of the entire command stream from the beginning

`$n` - where n is a value from 1 through 9--causes the specified or default parameter to be substituted from the command line

`^$` - the rest of the line is treated as a new set of parameters, separated by blanks

`^?` - ZEX stops processing and waits for <sp> or <cr> to continue processing (Control C aborts)

`^/` - like `^?`, but rings the console bell to summon the user

`^"` - causes ZEX to stop taking input from the command stream

```
                and wait for user input (which is toggled off by the
                ">" character
```

As you can see by the sheer number of features, ZEX is a command processor of great power but also great complexity.  We suggest that experimentation is the best way to learn its power.

As a final example of ZEX command files, here is a script that combines the automated assembler selection of ASMBL.ZEX with the ZCPR3 error checking of ZASLINK.ZEX. This edited version of ASMREL.ZEX also introduces the GOTO utility, which allows ZEX to conditionally skip over any number of command lines, resuming execution at a GOTO label, which takes the form of ;=LABEL.  This allows ASMREL.ZEX to include extra subroutine libraries in the linkage as an initial command line option.  ASMREL.ZEX is based on M80.ZEX by Richard Conn.

**Example Number 4:**

This ZEX command file employs two assemblers, one designed to use the ZCPR3 program error flag and the other modified to do so.  In addition to two default subroutine libraries, the user has the runtime option of including as many as four special-purpose libraries in the linkage on the initial command line.

```
;
;   Echelon ZAS 2.3 and DRI RMAC 1.1 Assemblers
;       with NightOwl PROLINK 1.5 Linking Loader
;
;     ^& Suppress FALSE IF Printout
;
IF NUL $1
ECHO  ** File Name Required for Assembly **
ELSE
IF EX $1.Z80    ;NOTE Zilog Mnemonics?
ZAS $1          ;NOTE Perform Assembly With ZAS
ELSE
RMAC $1 $$-S PZ;NOTE No, Use Modified RMAC
FI
IF ~ER
ERA $1.BAK      ;NOTE Cleanup BAK File
ERA $1.COM      ;NOTE Cleanup Old COM File
IF ~NUL $5      ;NOTE Link 4 Additional Libraries
PROLINK         ORIGIN         100&LINK         $1&SEARCH
$2,$3,$4,$5,Z3LIB,SYSLIB&EXIT
GOTO DONE
FI
IF ~NUL $4      ;NOTE Link 3 Additional Libraries
PROLINK ORIGIN 100&LINK $1&SEARCH $2,$3,$4,Z3LIB,SYSLIB&EXIT
GOTO DONE
FI
IF ~NUL $3      ;NOTE Link 2 Additional Libraries
PROLINK ORIGIN 100&LINK $1&SEARCH $2,$3,Z3LIB,SYSLIB&EXIT
GOTO DONE
FI
IF ~NUL $2      ;NOTE Link 1 Additional Library
PROLINK ORIGIN 100&LINK $1&SEARCH $2,Z3LIB,SYSLIB&EXIT
GOTO DONE
ELSE            ;NOTE Standard Link
PROLINK ORIGIN 100&LINK $1&SEARCH Z3LIB,SYSLIB&EXIT
;=DONE           Done with Link
FI              ;NOTE on IF ~NUL Tests
FI              ;NOTE on IF ~ER
```

```
ERA $1.REL
FI               ;NOTE on IF NUL
;
;  Operation Complete
;
```

These examples give some indication of the scope and power of ZEX, but it is up to you to find additional suitable applications.  Suffice it to say that ZEX can be an invaluable tool in the automation of computing tasks.  Like much of the Z-System, it performs simple tasks well and is powerful enough to grow with the user as he or she gains knowledge and confidence. Start out simply, and keep experimenting.  ZEX (and Z-System) will reward your efforts richly.


## The MENU Subsystem

ZCPR3's "menu subsystem" relies on two menu-oriented command processors--MENU and VMENU.  Both of these utilities read their data from ASCII files, which can be produced by any standard text editor, including WordStar.  MENU uses files of the type MNU; VMENU uses files of the type VMN.  The format of these files can be checked by the MENUCK and VMENUCK utilities.

MENU is a line-oriented processor.  VMENU is screen-oriented and relies on the presence of a ZCPR3 TCAP.  Otherwise, the two programs are quite similar.


## MENU

MENU is a ZCPR3 shell that reads a text file of the type *.MNU, prints the screen according to the display(s) set up in that file and processes commands from the scripts it contains.

The syntax of MENU is:

```
MENU <-- run MENU.MNU
```

or

```
MENUufn <-- run menu contained in file with unambiguous file
        name
```

When MENU is called by the user it looks for the file MENU.MNU in the current directory. If it finds this file it loads it and begins processing.  If it does not find the file, it searchs for file name with .MNU type; failing this, it aborts.  Let's examine a MENU.MNU and explain some MENU features as we go along.

**Example Number 1:**

This MENU.MNU organizes a group of programs into a coherent and easy to use disk cataloging system.  The Echelon DISCAT package is a much more refined implementation of this concept.

```
#DPX
        Master Catalog System
Defaults:     Master Catalog on drive A:
        Update disk on drive B:
off
        1 )   Create a new Master Catalog file
        2 )   Label a disk Volume
```

38

```
        3 )   Update the Master Catalog
        4 )   Find specific file(s) with a match key
        5 )   List the entire Master Catalog
        6 )   View the disk directory
        7 )   Purge a disk Volume
        8 )   Examine a disk Volume
     Enter your selection   (1 - 8):
   #
   1 era mast.cat;zex create                                   <<
     "List filenames.ext to be excluded, separated by commas: "
   2 era b0:-*.* i;save 0 b0:-"Volume label                    <<
     (VOLUME.NBR, 'VOLUME'=7 characters or less): "
   3 mcat b:;echo CONTROL-C ABORTS MAST.LST UPDATE;xcat
   4!find51 mast.lst "Enter matching key (string): "
   5!type mast.lst
   6!dir b:
   7 zex purge ."Enter volume number to be purged (all 3       <<
     digits): ";echo CONTROL-C ABORTS MAST.LST UPDATE;xcat
   8!find51 mast.lst "Enter volume number to be examined (3 digits): "
   ##
   on
```

The letters after the first pound sign (#) are the available options for this menu.  The letter D means the menu options are displayed, the P means the display is paged, so that a display of less than a full screen is padded out with carriage returns, and the X option allows exiting the menu to ZCPR3.  The only MENU option this file does not use is C, which displays the command line built by MENU as it is executed, useful mostly as a debugging aid.

Everything from the line below the options line to the next pound sign is the actual menu display seen by the user.  That's all, it's that simple.  MENU takes care of the command line seen at the bottom of the screen, which for a multi-screen MENU.MNU would look like the following:

```
Command (<CR>=Menu,^C=ZCPR3,*=1st Menu,<=Prev Menu,>=Last Menu) -
```

The actual commands MENU sends to ZCPR3 for execution are in the section of MENU.MNU that starts below the second pound sign and ends with the line above the double pound sign.  The commands are not displayed and the syntax rules are (necessarily) strict.  The first character of each command line is its "trigger", the single keystroke that will invoke it.  The actual command line starts immediately after this character, no spaces are allowed.  The command lines themselves are very similar to what is entered into a ZCPR3 or ALIAS.COM command line, with extensions.  Because some of the command lines in the file are too long for normal book margins, so we have taken the liberty of breaking them up  with the double left-pointing "<<" arrows to indicate where they join together--just remember, an actual MENU command line must be <u>on a single line only</u>.

The first of these command lines, triggered by the character "1", initializes a new master disk catalog file, MAST.CAT.  The old file is erased and a ZEX command file is invoked to do the actual initialization.  The text between the quotation marks is not part of the command line; the quotes are an important MENU feature that allow the user to provide command line input.  In this case, the user enters a list of file names to be excluded from the MAST.CAT file, separated by commas.  MENU sends this information to ZCPR3, which in turn passes it on to ZEX, which in this case operates WordStar from the inside to do the actual work.

The second command line, triggered by "2", also uses the quoted user input feature of MENU, this time to provide input to the ZCPR3 intrinsic command SAVE.  SAVE creates a properly named empty file that acts a disk volume label.

Triggered by "3", the next command line is the one that actually invokes the disk cataloging utility, MCAT, followed by an ECHO command reminding the user how to abort XCAT, the final command in the line.

Pressing "4" will invoke a string search utilty, again using the quoted user input feature, to scan the MAST.LST file for any inserted ASCII string.

The next command simply pages MAST.LST to the screen with the resident type command. No user input is involved.

If the user presses "6", the result will be a simple directory of the B drive, which in this case is where the disk to be cataloged would normally be (assuming a two-drive floppy-based system.

The sixth command line, triggered by "7", invokes another ZEX file that runs WordStar, this time on a find-and-delete mission to eliminate all references to the disk volume number provided by the user through the quoted input feature.  XCAT is then invoked to update MAST.LST after an ECHO reminding the user how to abort that process.

The last command, accessed by pressing "9", uses the string finding utility to display the contents of an entire disk volume.

A more complete discussion of MENU and MENU.MNU preparation is found in **ZCPR3: The Manual**.  Like the other major ZCPR3 programs, it rewards those who study and experiment.

The following is a list of MENU commands other than those programmed in MENU.MNU:

```
Command      Function
<CR>         Refresh Menu Display (RETURN Key)
^C           Exit to ZCPR3 (Control-C)
*            Jump to the First Menu
< or ,       Jump to the Previous Menu
> or .       Jump to the Next Menu
$            Jump to the System Menu (Password Required)
other        Menu Option or Invalid Command;
             letters are automatically capitalized, so a=A
```

## VMENU

VMENU operates much like MENU, except that it is screen oriented, requires, therefore, a TermCap (part of SYS.ENV in some Z-Systems), and is much more powerful.  VMENU reads a *.VMN file and derives its display(s) and commands from it.

When VMENU is invoked, it loads the names of the files in the current directory, loads the *.VMN file, and displays up to sixteen files in the current directory plus the first menu in the menu file.

VMENU is invoked as follows:

```
    VMENU            <-- run MENU.VMN on all files in dir
```

or

```
    VMENU afn        <-- run MENU.VMN on files selected by afn
```

or

```
VMENU afn ufn  <-- run menu (ufn) on selected files
```

VMENU's MENU.VMN files closely resemble MENU's MENU.MNU files, with slight variations.  The following MENU.VMN is a master menu of options for word processing magazine articles.

**Example Number One:**

Opening menu for multi-section word processing of magazine articles.
```
#X
      ^AWRITNG, EDITING AND TEXT TRANSMISSION SYSTEM MENU^B
      ^A------------------------------------------------^B
 F - GO TO FEATURES SECTION            P - GO TO PRODUCTS SECTION
 H - GO TO HIGHLIGHT SECTION           E - GO TO LEADING EDGE SECTION
 O - GO TO OUTLOOK SECTION             M - GO TO MISCELLANY
 W - WORKING DISK DIRECTORY            D - SYSTEM DISK DIRECTORY
 S - SEE LIST OF SECTIONS              R - RUN THE POINTED `COM' FILE
                  Z - RUN ANY ZCPR3 COMMAND LINE
 #
 F SHCTRL P;B0:;VMENU
 P SHCTRL P;B1:;VMENU
 H SHCTRL P;B2:;VMENU
 E SHCTRL P;B3:;VMENU
 O SHCTRL P;B4:;VMENU
 M SHCTRL P;B14:;VMENU
 S!CLR;PWD
 D!CLR;XDIR A:*.* U
 W!CLR;XDIR B:*.* U
 R$PN "ANY PARAMETERS?  OTHERWISE, <cr> "
 Z"YOUR COMMAND LINE PLEASE: "
 ##
```

It should be clear from our previous examination of  MENU.MNU this file is for general purpose and housekeeping use.  It plainly needs little explanation, but a few points are worth noting.  The ^A and ^B in the screen display area of the file turn screen highlighting on and off, a feature also usable in MENU.  In the command line area, note the use of exclamation point (!) as the second character in several command lines.  This causes VMENU (or MENU) to pause before reasserting control over the system, handy for viewing directories, etc.  A VMENU-only feature is the $PN parameter, which is the filename only of the pointed-to file in VMENU's file display.  Most of the commands here simply disengage VMENU by "popping" the ZCPR3 shell stack with the SHCTRL utility and log into the designated DU. VMENU is then reinvoked, using the MENU.VMN file found in the new work area.  Here is what a working area MENU.VMN might look like:
```
   #X
           ^AWRITING, EDITING AND TEXT TRANSMISSION MENU^B
 D-SHOW FILES IN THIS SECTION      |S-SET WORKING FILE TO POINTER
 E-WRITE ON/EDIT WORKING FILE      |T-TRANSMIT WORKING FILE
 C-CHECK SPELLING OF WORKING FILE |M-EXAMINE/MAINTAIN FILES
 ?-TIME, DAY AND DATE DISPLAY      |W-CONVERT WORKING FILE TO WS
 A-CONVERT WORKING FILE TO ASCII   |K-KILL WORKING FILE & BACKUP
 R-REMOVE ALL BACKUP FILES         |G-GET BACK ERASED WORKING FILE
 N-NAME OF WORKING FILE            |X-ERASE SPELLING CHECK WORKFILE
 L-LISTING OF SECTION NAMES        |Q-QUIT TO ANOTHER SECTION
                  B-BACK TO SYSTEM MENU
 #
 D!CLS;;DIR
```

```
S SETFILE 1 $PF
E WS $F1
T erab0:send.txt;cls;cpb0:send.txt=$F1;mex114ez read A0:xmit
C TW $F1
M VFILER
?!CLR;TIME
A CLS;RUN FILTX $F1
O CLS;RUN RESTORE
W ECHO ERASING OLD BACKUP FILE;ERA $N1.BAK;ECHO RENAMING WORKING   <<
  FILE TO N1.BAK;REN $N1.BAK=$F1;ECHO CREATING WORDSTAR FILE;<<
  WSDOCON $N1.BAK $F1;ECHO CONVERSION COMPLETE!
R ERA *.BAK
K ERA $N1.* I
G UNERASE $F1
N!CLR;ECHO;ECHO;ECHO;ECHO;ECHO;ECHO;ECHO;ECHO;DIR $F1
L!CLR;PWD
X ERA ERRWORDS.TXT
B SHCTRL P;SYSTEM:;VMENU
Q"SECTION NAME PLEASE: ":
z!"OK - do it!> "
##
```

In examining this MENU.VMN, it might be useful just to step back and absorb the complexity of some of the command lines that VMENU is called upon to execute with single keystrokes. The line triggered by "W", for instance, completely changes the way the program it services, an ASCII to WordStar text conversion utility, does its file handling, while sending helpful progress reports to the screen. The "T" command links the VMENU environment with a file-controlled communications program. The use of the SETFILE utility in conjunction with VMENU's pointer capability is another interesting twist. Some general reference information on VMENU follows.

The full Menu Command Line looks like the following:

```
Command (<CR>=Menu,^C=Z3,*=1st Menu,<=Prev Menu,>=Last Menu) -
```

The VMENU Commands are:

| **Command** | **Function** |
|---|---|
| ^R | Refresh Menu Display (RETURN Key) |
| ^C | Exit to ZCPR3 (Control-C) |
| * | Jump to the First Menu |
| < or , | Jump to the Previous Menu |
| > or . | Jump to the Next Menu |
| other | Menu Option or Invalid Command; letters are automatically capitalized, so a=A |

The internal *.VMN file commands are:

| **Command** | **Function** |
|---|---|
| :nn | Goto Menu nn, where the first menu is Menu 1 |
| ! | Wait after command line is executed before processing the menu |
| "Prompt" | Prompt the user for input and accept it |

The VMENU variables are:

| **Variable** | **Expands to** |
|---|---|

```
$D          Current Disk
$U          Current User
$Fn         FILENAME.TYP for System File n
$Nn         FILENAME for System File n
$Tn         TYP for System File n
$PF         FILENAME.TYP for Pointed-to File
$PN         FILENAME for Pointed-to File
$PT         TYP for Pointed-to File
$$          $
```

Note: System Files can be defined by the SETFILE command.

The Highlighting Embedded Characters are:
```
^A    Turn ON Highlighting
^B    Turn OFF Highlighting
```

Note: It is recommended that if highlighting is turned on, it should be turned off in the same line.

The following ASCII characters may NOT be used as commands since they are used elsewhere:
```
<SPACE>   #    %    ,    .    <    >    *    <DEL>
<Any Char Less than Space>
```

VMENU interacts intensively with the ZCPR3 System Files which are defined as a part of the ZCPR3 Environment Descriptor. There are four System Files, and three of them are used by VMENU for various purposes:

| File | Purpose |
|------|---------|
| 2 | Name of Current File |
| 3 | Name of Menu File |
| 4 | Name (containing wild cards) used to Select Files for VMENU File Display |

# VFILER

VFILER, the ZCPR3 file management shell, is a modern, screen-oriented file manipulation utility that also has considerable user programability. Although it is not quite as fully customizable as MENU or VMENU, its abundance of useful built-in features and programability has made it one of the most popular Z-System tools. If MENU and VMENU can be thought of as "clean slates" that the user writes upon to serve special needs, VFILER is a fully-realized working environment to which the user can add a number of custom functions.

VFILER has two standard, built-in screens, one a full-screen display of the files in the current directory with a pointer feature similar to VMENU's, the other a full menu of the standard VFILER functions that looks like this:

```
-- Tagging Commands --  --------- File Operations -----------
  T - Tag File              C - Copy File        D - Delete File
  U - Untag File            F - File Size        R - Rename File
  W - Wildcard T/U          Q - File usQ
                    G - Group Copy/Delete/FSize/usQ/Tag/Untag
                -- File Print & View --   --- User  Functions --
-
-- Cursor --  P - Print V - View   0-9 - Execute  # - Help
        ^E
```

```
        ^            -- Movement Commands --   ---- Miscellaneous ----
  ^S <-+-> ^D     <SP> - File Forward          A - Toggle Alpha Sort
       v           <BS> - File Backward         H - Help File
      ^X             +  - Screen Forward    L,N - New DIR
                     -  - Screen Backward     S - Disk Status
   -- Screen --    J - Jump to a File       Z - ZCPR3 Command
     ^A  Left      E - Refresh Screen   ^C,X - Exit
     ^F  Right
```

Typing a question mark (?) or forward slash (/) toggles between the two screens.  Regardless of which screen is being displayed, the currently pointed file is displayed in the upper right hand corner.  Many of the functions of VFILER will be familiar to those who have used CP/M programs like Robert Fisher's SWEEP, Frank Gaude's DISK7 and Dave Rand's NSWP.  As a matter of fact, DISK7 was the basis upon which Richard Conn designed VFILER, although it has plainly evolved far beyond its CP/M origins.

Using VFILER's built-in commands is quite straightforward, once you grasp that most of its operations are directed at the currently pointed file or at a group of "tagged" files.  Programming the 10 (numbered 0 through 9) User Functions is optional, but here's where you get to add some of your own needs and preferences to the general-purpose functionality of VFILER.  These functions are programmed by creating a text file called VFILER.CMD, which is similar in concept to MENU.VMN and MENU.MNU, but somewhat different in syntax.  Another notable difference is that VFILER looks for VFILER.CMD along the command search path if it is not found in the current DU.

**Example Number 1:**

```
TITLE:  General CMD Macro File
0 xdir %d%u:*.* ogoh'Options (p=to printer, d=disk, u=all users):'
1 get 100 ws.com;poke 392 ff;go %$
2 spell %$ $slic;review;markfix %$ $m#;if ex errwords.txt;  <<
  ws %$;era errwords.txt i;fi
3 vdo %d:%f
4 %d%u:;nulu -o %f -l
5 CAT:;catscan 'Master Catalog Search (e.g. *.doc): ' *.*;%d%u:
6 tmaker get income10.ei e;ac BACKUP:=%d:income10.ei /a
7 mex
8 DBASE:;dbase mail;%d%u:
9 protect %$ 'Attribute (CR=r/w, r=r/o, s=system): '
#
>>> SIMPLE WRITING, DATABASE and COMMUNICATIONS Menu <<<
-----------------------------------------------------------------
0 - Directory of Current Disk                              (XDIR)
- 0
1 - EDIT Current File (non-document mode)         (WordStar) - 1
2 - Spelling Check and Correction                     (SPELL+) -
2
3 - Edit Current File using VDO                           (VDO25)
- 3
4 - Enter Current Library (.LBR)                     (NULU151) -
4
5 - Scan Master CATALOG from Console                 (CATSCAN) -
5
6  -   Financial   Tally
(T/Maker) - 6
7 - Telecommunicate via modem                              (MEX)
- 7
8 - Mailing List Management System         (dBASE II & MAIL-ACG) - 8
```

```
9- Set Current File Attributes (R/O,R/W,System)        (PROTECT) - 9
-----------------------------------------------------------------
```

Note that in VFILER.CMD files, the command lines come first, the reverse of MENU.MNU and MENU.VMN convention, and extend to the first and only pound sign. All text after the pound sign is the Help screen shown when the VFILER user types "#" from either of the two standard VFILER screens. This section can be no longer than the length of the screen involved. You should also have noticed that all command lines begin with their corresponding trigger digit. Only the numerals 0 through 9 are recognized, and, unlike with MENU and VMENU, they must be followed by a space after which the actual command line begins. Lines that--like this file's TITLE--begin with anything other than the 10 single-digit numerals are treated as what programmers call "source code comments"--VFILER ignores them, but they can be handy notes to yourself if and when VFILER.CMD is re-edited.

The first command line, starting with "0", invokes the ZCPR3 XDIR program with a number of options. Any extra XDIR option(s) is supplied by the user via a quoted input function. Note that VFILER requires apostrophes (') rather the quotation marks (").

Triggered by a "1", the second User Function uses a POKE & GO technique to bring up WordStar (version 3.3) in non-document mode to edit the currently pointed file. The %$ parameter will be expanded by VFILER to a DU:FILENAME.TYP command line tail. WordStar (and most other non-ZCPR programs) will ignore the user number.

Pressing "2" brings up the next function, which uses programs from Oasis Systems' The WORD Plus package to check spelling on the currently pointed file. Note the use of the flow command IF EX to test for the presence of The WORD Plus's error file, ERRWORDS.TXT, and invoke WordStar if errors occurred. In that case, ERROWRDS.TXT is erased, with user confirmation, on exit from WordStar.

The fourth command line uses an alternative editing program which is faster but less versatile than WordStar, the public domain program VDO. Since VDO has trouble handling the DU: form, the pointed file is specified with the %d:%f parameter combination, which VFILER expands to D:FILENAME.TYP.

Triggered by a "4", the next command invokes Martin Murray's library file management utility, NULU, which attempts to open the pointed file as a Novosielski-style *.LBR file and list its contents. NULU's internal error checking is depended on to catch any input errors.

Pressing "5" temporarily logs into the named directory CAT: and invokes the Echelon DISCAT program CATSCAN with quoted user input defining the search via wildcard file specification. The %d and %u VFILER parameters, followed by a colon, return to the directory from which the function was requested.

The next line, triggered by "6", brings up the data file INCOME10.EI with the T/Maker multi-function program and employs the ZRDOS utility AC to place a copy of that file in the named directory BACKUP: if any changes have been made.

The next command line invokes the MEX communications program. You can, of course, substitute MDM740, COMM7, Crosstalk or the appropriate TERM3 command line as preferred.

Triggered by "8", the ninth User Function logs into the named directory DBASE: and brings up the dBase II command file MAIL.CMD, returning to the original directory when finished.

The final command line, triggered by "9", augments VFILER's file manipulation capability by invoking the ZCPR3 file attribute setting utility, PROTECT, with quoted user input.

# TALKING TO TERMINALS AND PRINTERS

## Introduction

ZCPR3's environment descriptor includes a TCAP (terminal capabilities entry) and buffers for the definition of peripheral devices.  The TCAP, the peripheral definition buffers, and the utilities that read both of these data buffers offer a flexible, powerful system to control terminals and printers.

The TCAP, cornerstone of the Z System's ability to interact with terminals, is a data area in high memory that describes the attributes of terminals or console CRT's (screens).  By "attributes" we mean the control codes or sequences needed to perform terminal functions such as clearing the screen, moving the cursor and erasing to the end of a line. Screen-oriented Z Tools know about the TCAP and perform a variety of screen oriented functions exploiting their knowledge of the TCAP entry for your terminal. SHOW and VFILER are two notable Z System utilities that use the data in the TCAP for video emphasis and refinement.  There are many similar Z Tools and you will discover them as you explore the system.

The peripheral definition buffer has space to define the characteristics of two CRT's and 4 printers .  Certain Z System utilities, such as PRINT  and PAGE, read information from this buffer.  PRINT reads the data describing one of four printers (you select the printer with the CPSEL utility) - information about the ability of the printer to formfeed, the rows of text on one page, and the width of the printer.  The PAGE utility, a fancy version of the resident TYPE command, reads the data describing one of two terminals for screen width and height. Note that when we speak of printer and terminal definitions in the peripheral definition buffer we are talking about logical printers and terminals. The word "logical", when referring to a hardware device such as a printer or terminal, does not refer to the actual physical characteristics of the device but to its attributes as set forth in the peripheral definition buffer. Your printer may have physical width of 132 columns using 10 pitch type but you can define it as having a logical width of 80. From the perspective of ZCPR3 and its utilities, such as PRINT, your printer will now function as if it had only 80 columns.

## The TCAP

The TCAP, your terminal capabilities entry, occupies 128 bytes of memory and contains all the information needed by videooriented Z Tools to work correctly with your terminal.

### Getting TCAP Information Into The Environment Descriptor

You have three ways of getting information about your terminal into the 128 byte buffer at the top of the environment descriptor.

## SYS.ENV

First, you can put your terminal definition in SYSENV.LIB (the configuration file for SYSENV.ASM) and create a new SYS.ENV (SYSENV.COM is renamed to SYS.ENV after it is assembled and loaded). As you recall, an .ENV file is loaded with LDR.COM.  Here is the SYSENV.LIB entry for one of our computers, a Kaypro 484:

```
B0:WORK>PAGE ASMFIL.DOC PL
PAGE,  Version 2.0
PAGE File: B  0: ASMFIL  .DOC
 1:                              ;
 2:                              ; Terminal Capabilities Data
 3:                              ;
 4: envorg2:
 5:   DB    'Kaypro 484       '  ;Name of Terminal
 6:   DB    'K'-'@'              ;Cursor UP
 7:   DB    'J'-'@'              ;Cursor DOWN
 8:   DB    'L'-'@'              ;Cursor RIGHT
 9:   DB    'H'-'@'              ;Cursor LEFT
10:  DB    00                   ;CL Delay
11:  DB    00                   ;CM Delay
12:  DB    00                   ;CE Delay
13:  DB    'Z'-'@',0            ;CL String
14:  DB    1bh,'=%+ %+ ',0      ;CM String
15:  DB    18h,0                ;CE String
16:  DB    1Bh,'B1',0              ;SO String
17:  DB    1Bh,'C1',0              ;SE String
18:  DB    0                    ;TI String
19:  DB    0                    ;TE String
20:
21: ds    80H-($-envorg2)          ;  make  exactly  80H  bytes
long
22:
23: ;
24: ;   End of Environment Descriptor
25: ;
26: endm
```

Just a few words of explanation.  `CL` (lines 10 and 13) indicates the clear screen sequence, `CM` (lines 11 and 14) the cursor motion sequence, `CE` (lines 12 and 15) the clear-to-end-of-line sequence. `SO` (line 16) means begin highlighting, `SE` (line 17) end highlighting, `TI` (line 18) terminal initialization, and `TE` (line 19) terminal de-initialization.  `"'Z'-'@'"` means Control Z. (The clear screen character for the Kaypro 484 is Control Z).  The percent character (`%`) tells the cursor motion interpreter (a subroutine found in screen-oriented Z-System utilities) that the next character is an interpreter directive.  The `"%+ "` sequence (line 14), for example, tells the cursor motion sequence interpreter to add 32 or 20H to the current value of the row or column and output that value in binary. (32 decimal is an ASCII space.) The formula for defining cursor motion on the Kaypro 484 is `"ESC = ROW + 32, COLUMN + 32."`  The formula is expressed in terms the cursor motion sequence interpreter can understand on line 14.  These concepts may appear complex, but complexity is defeated by play.  You may want to play a little at this point using these techniques to get data into your TCAP.  Here is an assignment.  After reviewing the listing above, see if you can figure out how to change the standout mode from half-intensity video to reverse video. Clue? The sequence to enter reverse video on the Kaypro 484 is `"ESC B 0"` and the sequence to exit reverse video is `"ESC C 0"`. Can you do the same things with your own terminal?

# TCMAKE

The second method of generating terminal data and loading it into the TCAP relies on the TCMAKE utility. TCMAKE lets you define your terminal attributes and store the definition in a system segment we have not yet discussed - a .Z3T file. If you are using a Wyse terminal, you might use TCMAKE to create a file WYSE.Z3T, which could be loaded into the TCAP slot with LDR.COM. The command would be LDR WYSE.Z3T. TCMAKE is menu driven. We will not go into the details of its use. That information is available in **ZCPR3: The Manual**, pp. 309-324.

# TCSELECT

The third and final technique to get data into your TCAP, using the TCSELECT utility, is by far the easiest and the one we recommend at the outset. TCSELECT selects a terminal definition from a standard database file of terminals (over 40) and either creates a .Z3T file for loading by LDR.COM or directly loads the data into the memory resident TCAP. The standard database file is usually called Z3TCAP.TCP.

TCSELECT selects an entry for a terminal from the Z3TCAP.TCP file and either loads that entry into the TCAP or writes it to a file of the type .Z3T for loading into memory with LDR.COM. This is what the "//" help parameter for TCSELECT shows you (remember that almost all Z Tools use the "//" help parameter):

```
B0:WORK>tcselect //
TCSELECT, Version 1.1
     TCSELECT - Select Entry from Z3TCAP.TCP
Syntax:
   TCSELECT outfile  -or-  TCSELECT outfile.typ
      where "outfile" is the file to be generated by
      the execution of TCSELECT.  If no file type is
      given, a file type of Z3T is the default.
Syntax:
   TCSELECT
      where this alternate form may be used to store
      the Z3TCAP entry for the selected terminal directly
      into the Z3 Environment Descriptor.
```

Now let's run TCSELECT and pick a terminal, using the option to write the terminal of our choice to a .Z3T file:

```
B0:WORK>tcselect myterm.z3t
TCSELECT, Version 1.1
** Terminal Menu 1 for Z3TCAP Version 1.5   **
A.   AA Ambassador          K.   Apple //e PCPI
B.   ADDS Consul 980        L.   Apple ///
C.   ADDS Regent 20         M.   Bantam 550
D.   ADDS Viewpoint         N.   CDC 456
E.   ADM 2                  O.   Concept 100
F.   ADM 31                 P.   Concept 108
G.   ADM 3A                 Q.   CT82
H.   ADM 42                 R.   DEC VT52
I.   Apple //e ALS          S.   DEC VT100
J.   Apple //e MS           T.   Dialogue 80
Enter Selection, + for Next, or ^C to Exit - +
** Terminal Menu 2 for Z3TCAP Version 1.5   **
A.   Direct 800/A           K.   HP 2621
B.   Epson GENEVA           L.   IBM 3101
```

```
C.  Epson QX-10          M.  Kaypro II
D.  General Trm 100A     N.  Kaypro 10
E.  Hazeltine 1420       O.  Micro Bee
F.  Hazeltine 1500       P.  Microterm ACT IV
G.  Hazeltine 1510       Q.  Microterm ACT V
H.  Hazeltine 1520       R.  NorthStar Advant
I.  H19 (ANSI Mode)      S.  Osborne I
J.  H19 (Heath Mode)T.  P Elmer 1100
Enter Selection, - for Last, + for Next, or ^C to Exit - N
Selected Terminal is: KP-10 Kaypro     -- Confirm (Y/N)? Y
File MYTERM  .Z3T Created
```

Now that we have created MYTERM.Z3T, let's load it into memory so we can test it.

```
B0:WORK>ldr myterm.z3t
ZCPR3 LDR, Version 1.4
 Loading MYTERM.Z3T
B0:WORK>;and we are done.
```

## The Peripheral Definition Buffer

The peripheral definition buffer, located in the environment descriptor, holds information on the attributes of two CRT's and four printers.  The CPSEL utility is used to select one of the two CRT's and one of the four printers.  Certain ZCPR3 utilities such as PAGE and PRINT read the definitions of the selected terminal and printer and govern their output accordingly.

We have provided the source code listing for that part of SYS.ENV containing the terminal and printer definitions stored in the peripheral definition buffer after SYS.ENV (or any other .ENV file) is loaded into memory by LDR.COM.

```
db   0            ; CRT selection (0=CRT 0, 1=CRT 1)
db   1            ; Printer selection (n=Printer n)

db   80           ; width of CRT 0
db   24           ; number of lines on CRT 0
db   22           ; number of lines of text on CRT 0

db   132          ; width of CRT 1
db   24           ; number of lines on CRT 1
db   22           ; number of lines of text on CRT 1

db   80           ; width of Printer 0
db   66           ; number of lines on Printer 0
db   58           ; number of lines of text on Printer 0
db   1            ; form feed flag (0=can't formfeed, 1=can)

db   102          ; width of Printer 1
db   66           ; number of lines on Printer 1
db   58           ; number of lines of text on Printer 1
db   1            ; form feed flag (0=can't formfeed, 1=can)

db   80           ; width of Printer 2
db   66           ; number of lines on Printer 2
db   58           ; number of lines of text on Printer 2
db   0            ; form feed flag (0=can't formfeed, 1=can)

db   132          ; width of Printer 3
db   88           ; number of lines on Printer 3
```

```
db   77          ; number of lines of text on Printer 3
db   1           ; form feed flag (0=can't formfeed, 1=can)
```

The assembler source for the peripheral definition buffer uses the "DB" assembler directive to Define Bytes. The first byte of the buffer, which is the 48th byte of the environment descriptor, holds the selection of your CRT and the next byte holds the selection of your printer. The remaining bytes define the characteristics of the two possible logical CRT's and the four possible logical printers. Note that the last entry defines printer 3 as having 88 lines, with 77 lines of actual text. One of us uses printer 3 when he is printing in condensed print at 8 lines per inch. The definition of printer 3 as having 88 lines per page simply tells the PRINT utility not to advance the printer to a new page until 88 lines of text have been typed. Once again, we mention this to emphasize the difference between logical and physical devices.

## Selecting Terminals And Printers

There are a number of ways of selecting the logical terminal and the logical printer to provide the desired information to the appropriate ZCPR3 utilities.

## CPSEL

CPSEL selects one of two console CRT's and one of four printers as those devices are defined in the peripheral definition buffer of the environment descriptor. It also provides information on the selected console CRT and the selected printer. Finally, CPSEL shows you the definition of the remaining, unselected CRT's and printers.

This is the "double-slash" help offered by CPSEL:

```
B0:WORK>cpsel //
CPSEL  Version 1.0
CPSEL cmd1 cmd2 cmd3,...
Commands:
 Cc, c=0 or 1  -- Select CRT 0 or 1
 Pp, p=0,1,2,3 -- Select Printer 0, 1, 2, or 3
 Dd, d=A (All), C (CRT), P (Printer)
       -- Display Selection Values
```

And this is the display when we tell CPSEL to show us all selection values:

```
B0:WORK>cpsel da
CPSEL  Version 1.0
 Current CRT Selection: 0
*CRT 0:  Width =  80  Actual/Text Lines =  24/ 22
 CRT 1:  Width = 132  Actual/Text Lines =  24/ 22
 Current Printer Selection: 1
 Prt 0:  Width =  80  Actual/Text Lines =  66/ 58  Form Feed?
Yes
*Prt 1:  Width = 102  Actual/Text Lines =  66/ 58  Form Feed?
Yes
 Prt 2:  Width =  80  Actual/Text Lines =  66/ 58  Form Feed?
No
 Prt 3:  Width = 132  Actual/Text Lines =  88/ 77  Form Feed?
Yes
```

Note the "*" indicates that particular logical device has been selected.

For those who want to flex their new Z muscles, how about using the resident POKE command? If your environment descriptor starts at F300H, you might try `POKE F32F 01` in

order to change your CRT selection from CRT 0 to CRT 1.  Take the time to use the PEEK command to inspect the peripheral definition buffer.  Knowledge of the addresses of the various definition bytes will permit you to construct alias scripts to poke different definitions into the buffer and select different console CRT's and printers without using CPSEL.  Sometimes the long way around winds up being the short way home.

## Examples Of Utilities That Read The Peripheral Definition Buffer

PAGE and PRINT both read the data contained in the peripheral definition buffer of the environment descriptor.  This abililty provides power and flexibility in the manipulation of terminals and printers.

## PAGE

PAGE prints text files on your screen.  It is a fullfeatured, elegant version of TYPE.  Most notable, for purposes of our exploration of the environment descriptor, PAGE reads the information about the system console CRT and performs screen wordwrap if line length exceeds the screen width as defined for the selected console CRT in the peripheral definition buffer.  Once again, the "slash-help" facility provides information on the power of PAGE.

```
B0:WORK>page //
PAGE,  Version 2.0
Syntax:
PAGE file1,file2,...,filen o...
Options:
0-9  Select Delay Constant
I    Inspect and Select Files First
L    Toggle Numbering of Each Line
P    Toggle Paging
SnnnnSkip to Specified Page before Printing
Examples:
     PAGE MYFILE.TXT,*.MAC LI
          -- Number Lines, Inspect Files
     PRINT MYFILE.* S25
          -- Skip to Page 25
Commands during printout:
^C  - abort PAGE          ^X  - skip to next file
^S  - suspend output       P  - toggle paging
0-9 - change speed
```

Note that PAGE permits you to view files sequentially and to skip any file in a series.

## PRINT

PRINT is the printer equivalent of PAGE.  It too reads logical device definitions in the environment descriptor and offers full page printing with selection of printer characteristics.  PRINT controls page headings, page numbering, and sequential file printing.  Once more, the ever-helpful double-slash provides you with sufficient information to get going.

```
B0:WORK>print //
PRINT III,  Version 2.1
     PRINT file1,file2,...,filen o...
Options:
 E   Exact Print (expand tabs, form feed, no line
          or page numbers, no heading)     default no
 F   Toggle File Name Display              default yes
H@head@ Specify Page Heading (@ is any printing char)
```

51

```
I     Inspect and Select Files First              default no
L     Toggle Numbering of Each Line        default no
M     Toggle Multiple Runs (MR=No TOF Msg)   default yes
N     Toggle Numbering of Each Page        default yes
Onn   Offset Printout by nn Characters from Left default 0
Snnnn Skip to Specified Page before Printing default no
T     Toggle Time Display in header          default yes
Examples:
      PRINT MYFILE.TXT,*.MAC LH'SAMPLE'
           -- Number Lines, SAMPLE is Heading
      PRINT MYFILE.* S25E
           -- Skip to Page 25, Exact Print
At any time, ^C aborts PRINT and ^X skips to next file
```

As you begin to work with logical printer definitions you will find the PRINT utility to be one of the most frequently used of your ZCPR3 tools.

## Further Explorations

After you have become comfortable with the manipulation of the environment descriptor to tailor the interaction between your computer and its terminals and printers, you may wish to study DPROG, the control language interpreter used to program these physical devices. DPROG allows you to create programs or scripts with understandable English words for controlling your peripherals.  Many superb DPROG programs are available from Z-Nodes worldwide.  These programs have been created for Epson, Okidata, and other popular printers, as well as for a number of commonly used terminals.  One of the best ways to learn DPROG, or device programming, is to look at what others have done.

# THE Z-SYSTEM TOOLSET (Z Utilities)

This section describes briefly the major ZCPR3 utilities.  For more detailed explanations, check **ZCPR3: The Manual** or the appropriate ZCPR3 Help file.  Most ZCPR3 utilities will put a brief syntax help message on the screen if they are invoked with "//" as the command tail.

## ZCPR3 Utilities

**ALIAS**

The ALIAS utility creates alias files and displays their contents.  An alias is a script that can be passed as a series of commands when the COM file containing the script is run.

**CD**

The CD utility  stands for "change directory."  It permits moving from one directory to another.  Whenever it logs into a new directory it looks for a COM file called ST.COM.  If it finds ST.COM, it runs it.   ST.COM is usually an ALIAS that establishes the desired conditions in a directory area.  Thus, ST.COM may load new sets of RCP's, FCP's, and NDR's.  If you are logging into a directory ASM you may wish to change your RCP's, for example, to suit your work with assembler.  The interaction between CD and ST is another instance of the Z-System's emphasis on fine tuning the user's environment.

**CLEANDIR**

This utility "cleans" your physical disk directory.  CLEANDIR loads the directory of the disk, sorts it, and writes it out to disk, filling unused directory entries with E5's--resulting in speedier directory listings and increasing the likelihood of successful recovery of accidentally erased files.

**COMMENT**

Puts the Z-System computer in a mode similar to beginning all commands lines with a semi-colon: no commands are executed, but they are seen on the screen and may also be sent to the LST: device with a toggle command.

**CMD**

A special purpose tool designed to enhance the SHSET shell setting utility.  Allows direct access to the ZCPR3 command line, including SHCTRL program, which can terminate the shell.

**CMDRUN**

A simple demonstration program designed to illustrate the ZCPR3 CMDRUN function.  Usually replaced with functional extended command processor, such as ZEX.COM renamed to CMDRUN.COM.

**CPSEL**

CPSEL permits you to select one of two consoles and one of four printers as defined in the

environment descriptor. This is useful when you have directed your printer, for example, to print out 8 lines per inch and want the page to be filled. Thus, you would use CPSEL to select a logical printer defined as having more than the usual number of lines per page. Be aware that CPSEL's action can only be detected and used by a ZCPR3 utility.

## CRC

This utility is an enhanced ZCPR3 version of Keith Petersen's CRCK program. It runs a Cyclical Redundancy Check on specified files and optionally creates a disk file of their values. This utility is used to insure that two files are in fact identical.

## DEV

DEV manipulates the ZCPR3 redirectable I/O device drivers. It is used to display and select current output devices on Z-Systems implemented with I/O packages. It accepts instruction on the command line and thus is more useful than DEVICE (see below) when invoked from a command script.

## DEVICE

DEVICE assigns physical to logical devices, interactively displays the names of available physical device drivers, and, unlike DEV, runs only in the interactive mode. DEVICE requires that an appropriate I/O package be implemented and loaded into memory.

## DIFF

DIFF compares two files and tells you if and how they are different. Such comparison is useful when you are trying to determine whether two files are the same or to determine in what ways they differ.

## DIR

This utility displays current or requested directory files, gives file sizes, disk space used, and disk space remaining. DIR is the speediest of the ZCPR3 directory utilities, but lacks many of the advanced features of XD and XDIR.

## DPROG

DPROG is an interpreter used to program physical devices, such as terminals and printers. DPROG permits relatively simple, English commands to control physical devices.

## DU3

DU3 is the ZCPR3 version of DU, the disk utility that permits byte-level surgery to be performed on your disks. DU3 is far more powerful than previous DU variants.

## ERASE

ERASE is the transient counterpart of ERA and is much more powerful, offering the user full control of all erasure operations, with many options.

## ERROR1, ERROR2, ERROR3, AND ERROR4

These utilities are the ZCPR3 error handlers. Each error handler can be installed or uninstalled dynamically (remove any error handler with ERRORX), and each has a slightly different flavor. ERROR2 is screen oriented. Error handlers allow you to make graceful recoveries from erroneous commands, whether single or multiple.

## HELP

Provides full online documentation of the ZCPR3 system. **ZCPR3: The Manual**, available from Echelon, Inc., is the Bible for the ZCPR3 system and the HELP system contains a major subset of the information in the **MANUAL**, for interactive use use at the console. HELPCK and HELPPR are used to check the syntax of HLP files you create and print paper copies of HLP files.

## IF

Somewhat more powerful and flexible COM file alternative to the FCP-resident IF command.

**IFSTAT**

A simple diagnostic tool useful in writing flow-controlled command scripts. Displays the current IF level. IF state must be toggled to TRUE for IFSTAT to run.

**LDR**

The ZCPR3 system segment loader. Properly positions Environment Descriptors (*.ENV), Flow Command Packages (*.FCP), Resident COmmand Packages (*.RCP), Input/Output Packages (*.IOP) and ZCPR3 TCAP segments (*.Z3T) in memory.

**MCOPY**

This utility copies files from one directory to another. Unlike the resident CP command, MCOPY allows CRC verification and has other powerful options. Unlike CP, it does not allow copying to the same DU or DIR with a changed filename.

**MENU**

MENU runs the ZCPR3 menu subsystem. MNU menus are created with your text editor following the syntactical rules for the MENU subsystem described in the **Manual** and the MENU Help file. MENU is a ZCPR3 shell, a program that can optionally replace the standard ZCPR3 prompt as the interface between user and computer.

**MKDIR**

MKDIR creates named directories. Each directory may have a name up to eight characters. Optional passwording of each directory is available.

**MU**

The ZCPR3 screen-oriented memory editing utility. Available as both a COM file (MU3) and an RCP (DEBUGRCP). A more sophisticated implementation of the P (peek) and POKE RCP commands, with added features.

**PAGE**

PAGE prints text files on your screen. You may view files sequentially and skip any file in a series. PAGE will perform screen wordwrap if line length exceed "logical" screen width.

**PATH**

PATH displays and dynamically alters the command search path .

**PRINT**

PRINT offers full page printing of text files on printer. CPSEL selects printer characteristics. PRINT controls page headings, page numbering, and sequential file printing.

**PROTECT**

PROTECT sets file attributes--read-only, system, archive, or read-write. An alternative to the ZRDOS utility, SFA.

**PWD**

PWD prints to the CRT the names of the available named directories.

**QUIET**

QUIET reduces the amount of information displayed by ZCPR3 utilities programmed to respond to the system's "quiet byte."

**RECORD**

Special purpose tool to control the operation of the Echelon IOREC (input/output recorder) or any properly designed Input/Output Package (*.IOP) implementing the routing of console output to a disk file or peripheral device.

### REG

A utility most useful in sophisticated flow-controlled command scripts. Loads, arithmetically manipulates and reads a series of 10 (0-9) reserved one-byte buffers called the ZCPR3 Registers.

### RENAME

More powerful COM file equivalent to the RCP resident and ZCPR3 intrinsic REN command. Allows operation on lists of files and supports ambiguous file name references.

### SAK

SAK stands for "Strike Any Key" and is used in command lines to suspend execution for a specified time or to allow optional cancellation of the following operation(s). SAK can optionally alert the user that input is requested by sounding the console bell.

### SETFILE

This utility is used to declare up to four system file names for use with aliases and with shell programs such as VMENU. The SHOW utility displays active system files.

### SHOW

SHOW displays the status of the ZCPR3 environment--buffer address locations, command line length, system cpu speed, maximum permitted drive and user, and much more on eleven information screens. The command SHOW E installs SHOW as an error handler.

### SHCTRL

SHCTRL is the ZCPR3 shell manipulation utility. It allows the user to view the ZCPR3 Shell Stack, clear all shells or exit the currently active shell.

### SH, SHDEFINE, SHFILE, SHVAR

SH is the ZCPR3 Named Variable Shell, the remaining programs are utilities most useful in conjunction with SH. User applications for these programs are somewhat few and far between, but they do provide a very impressive demonstration of the ZCPR3 shell facility's power and versatility.

### SHSET

SHSET allows any executable program to be invoked as a ZCPR3 shell. Word processors, communications programs, and many other programs of an interactive nature are good shell candidates.

### SUB

The highly enhanced ZCPR3 version of CP/M's SUBMIT utility. A disk based command script facility, it is considerably slower to execute than an alias or the memory-based ZEX program and has been largely supplanted by them in the user community.

### UNERASE

This utility lists and recovers accidentally erased files.

### TCCHECK, TCMAKE, TCSELECT

These utilities support the ZCPR3 TCAP (Terminal CAPabilities) facility and are of little interest to users of standalone systems like the Kaypro. They are indispensible if you operate your system from an external terminal(s).

### VFILER

VFILER is a screen-oriented file manipulation utility similar to DISK7 or NSWP, but much more powerful in a ZCPR3 environment. It allows you to point an arrow to a file in your directory and perform a number of operations on it. VFILER is a ZCPR3 shell.

**VMENU**

Runs the VMENU (Video-oriented MENU) subsystem of ZCPR3. Employing user created *.VMN command files, VMENU offers full menu control, lists portions of the directory at the top of the screen and has point-to-file capability. VMENU is a ZCPR3 shell.

**WHEEL**

Permits changing the system status from secure to unsecure and back. If system is set to secure, passwords may be needed to access certain directories and protected files. Certain utilities, such as MKDIR and ALIAS, may not be used when the wheel byte is RESET.

**XD**

A directory utility, XD is one step up in power from DIR but somewhat slower. While not as powerful as XDIR, it is faster in operation.

**XDIR**

XDIR is the most powerful of the ZCPR3 directory utilities as well as the slowest.

**Z3INS**

The ZCPR3 utility installation tool. Z3INS must be used to install ZCPR3 programs obtained from Echelon or from Z-Nodes before running them on your system. Implants the environment descriptor address in the target COM file(s).

**Z3LOC**

Displays all major Z-System addresses. Faster than SHOW, but supplies less detailed information.

**ZEX**

ZEX is ZCPR3's very powerful, memory-based alternative to CP/M's SUBMIT and XSUB programs. Not only allows command scripts of virtually unlimited length, but allows operation of many programs "from the inside" with scripted input. Very fast and versatile, it interacts very closely with ZCPR3 via special message bytes.


# ZRDOS Utilities

These utilities operate only with the ZRDOS BDOS replacement.

**AC**

The archive copy utility, AC permits you to select for automatic copy those files that have not been marked as archived. It has many options, including erasure of source file upon copy completion, resetting of disk system before changed disks, and others.

**COMP**

COMP is a video-oriented file compare utility, more powerful than DIFF, that graphically displays the differences between two files. It also manipulates the ZCPR3 registers, a feature useful in conditional command scripts based on file comparison results.

**DFA**

Displays file attributes, which can be set by the ZRDOS SFA or ZCPR3 PROTECT utilities.

**DUMP**

Dumps file contents, in enhanced DDT-style hexadecimal and ASCII, to screen or to printer.

**PUBLIC**

This utility sets drives and user areas as public or private. Public areas are great places to put overlays for programs such as WordStar, permitting full program access from any user area.

**SFA**

Sets file attributes--wheel protected (you cannot write to a wheel protected file if the wheel byte is reset), system, read- only, and archive.

**VIEW**

A text file scanner.  Allows toggle of display of control characters and permits movement forward and backward in file.

**VTYPE**

File scanning utility permitting forward and reverse text viewing with controlled scrolling rate, string searches, quick goto bottom and top of file movements, etc.  Fully screen oriented with built-in in-context help system.