NZ-COM

The Automatic Z-System

by

Joe Wright

User's Manual

by

Jay Sage and Bridger Mitchell

Alpha Systems
711 Chatsworth Place
San Jose, CA 95128

Copyright Notice

Disclaimer of Warranty

Acknowledgments

The NZ-COM system was tested and improved by the suggestions of Cam Cotrill, Howard Goldstein, Bridger Mitchell, and Jay Sage.

Z-System is an evolving operating system environment for Z80-compatible microcomputers to which many have contributed. Development of NZ-COM has especially benefitted from the work of:

Michal Carson	-	VLU
Richard Conn	-	ZCPR v 3.0, HELP, TCSELECT
C. B. Falconer	-	LT
Rob Friefeld	-	SALIAS
Steven Greenberg	-	CRUNCH, UNCRUNCH, LDIR
Al Hawley	-	EDITNDR, SAVENDR
Bridger Mitchell	-	Z3PLUS, JETLDR
Bruce Morgen	-	LPUT, LGET
Paul Pomereau	-	EASE
Jay Sage	-	ZCPR v 3.3, v 3.4, ARUNZ, ZFILER, ZSHOW
Joe Wright	-	NZCOM

Trademarks

Z-System, ZCPR3	Alpha Systems
Z3PLUS, DosDisk, DateStamper	Bridger Mitchell
CP/M, CP/M-Plus	Digital Research, Inc.
WordStar	MicroPro Corp.

PREFACE

Automatic, universal, and dynamic, the two new **Z-Systems** — **NZ-COM** for **CP/M-2.2** computers and **Z3PLUS** for **CP/M-Plus** computers — are the result of the extensive cooperative efforts of Joe Wright, Bridger Mitchell, and Jay Sage. All three authors worked together on many aspects of both products; each has been particularly responsible for one essential characteristic.

Joe Wright brought automatic operation. It was he who first conceived of and demonstrated with **Z-COM** what many deemed impossible - a version of **Z-System** that would install itself automatically on almost any **CP/M-2.2** computer. Yet, even after **Z-COM**'s success, it still appeared that **Z-System** could never run on a **CP/M-Plus** computer with its radically different command processor and banked memory operating system. **Z3PLUS** proves otherwise.

Bridger Mitchell made the systems universal. From his first exposure to **Z-System**, he decried the complex, laborious effort required to make each code module run with any particular implementation of the system. He developed a universal **ZRL** file format and loader that would allow a single file to adapt to any **Z-System**, no matter what its configuration in terms of module addresses and sizes.

Jay Sage added dynamics. He conceived an operating system that can change its size and character — right in the middle of a command line if necessary — to suit the needs of a particular task. No longer must the user live with a rigid compromise between operating system features and memory consumption. The trade-off can be reconsidered at any time.

These three authors played the primary role in the development of the new systems and are responsible for equally fundamental developments embedded in the new **Z-System** - the **ZCPR** version 3.4 command processor (Jay Sage), the **Z3PLUS** loader and command processor (Bridger Mitchell), and the **NZ-COM** loader (Joe Wright). But the full cast included many other players, in fact, a whole community of them. It is this shared participation in the ongoing development of **Z-System** in general that makes the effort so satisfying and rewarding for all of us.

Contents

1 What is NZ-COM? 1

1.1 The Benefits of NZ-COM 2

1.2 The New Command-Processing Features 3

1.3 Other Major Features 4

1.4 Using This Manual 4

2 Starting NZ-COM the First Time 7

2.1 Introduction 7

2.2 Setting Up the Files 8

2.3 Running NZ-COM 9

2.4 Organizing Your Files 10

3 Getting Acquainted with Z-System 13

3.1 Creating an Alias 13

3.2 The Available Commands 15

3.2.1 RCP Commands 15

3.2.2 FCP Commands 17

3.2.3 CPR Commands 20

3.2.4 Transient Commands 22

3.3 More About the System Configuration 23

4 Learning More About Z-System **25**

4.1 Resident NZ-COM Components 25

4.2 Automatic Commands 26

4.2.1 Extended Command Processor 26

4.2.2 Error Handler 27

4.2.3 Command Shells 28

4.3 Other Z-System Tools 32

4.3.1 ARUNZ 33

4.3.2 HELP 35

4.3.3 Library Tools 35

4.3.4 File Compression 36

4.3.5 Named Directory Tools 37

4.3.6 Other Tools 38

4.4 Command Hierarchy 38

4.4.1 Command Acquisition 39

4.4.2 Command Resolution 39

5 Getting More Out of NZ-COM **41**

5.1 Alternative NZ-COM Invocation Commands 41

5.2 Customizing Your NZ-COM System 42

5.2.1 Temporary Changes 42

5.2.2 NZ-COM Descriptor Files 43

5.2.3 Modifying the Descriptor Files 44

5.2.4 Creating New Definitions with MKZCM 47

5.3 Loading New Systems 49

5.4 The System Cloning Option 51

5.5 Changing Systems on the Fly 52

5.5.1 Loading Rules 52

5.5.2 Why Change Systems 53
5.5.3 Automating System Changes 54
5.5.4 Automatic Returns to CP/M 55

6 Technical Reference 57

6.1 Definition of File Types 57
6.2 Files Supplied with NZ-COM 57
6.2.1 NZ-COM System Files 57
6.2.2 Tools and Utilities 59
6.3 NZCOM Command Lines 61
6.3.1 Help Screens 61
6.3.2 Loading NZ-COM Systems 62
6.3.3 Removing NZ-COM 66
6.3.4 Examples and Tips 67
6.4 Patching NZCOM.COM 69
6.4.1 The Internal Search Path 69
6.4.2 The Default Options 70
6.4.3 Startup Command Line 70
6.5 The JetLDR Program 74
6.6 The MKZCM Command Line 74
6.7 Theory of Operation 74

7 Bibliography 77

7.1 The Z-Nodes 77
7.2 *The Computer Journal* 78
7.3 Other Published Information 79

Chapter 1

What is NZ-COM?

This manual has to address two audiences: those who are already running a version of what is known as the ZCPR3 operating system or Z-System and those to whom this is all new. Since from our own experience, there is always more to learn about Z-System, we have no qualms about addressing the manual to those in the second group. To the best of our abilities, we have tried not to assume on the part of the reader any prior knowledge about Z-System. Experienced Z-System users may want to skip or skim some sections. On the other hand, little tidbits or new insights may await you in any section.

NZ-COM — formally known as Z-COM version 2 — is one of the most exciting and remarkable developments in the history of microcomputer operating systems. Your computer's operating system is its master program, present from the moment you turn on the power. It interprets commands, loads and executes programs, and manages the disk files and connections to your terminal, printer, and modem.

Normally, a computer's operating system is a static entity. You "boot" up the computer, and there you have the operating system, fixed and immutable. Few computers offer more than one operating system. With those that do, the only way you can get a different operating system is to "reboot", which generally involves inserting a new boot diskette and pressing the reset button. And never do *you* get to define the characteristics of that operating system. You have to accept what the manufacturer offers you.

With NZ-COM the operating system becomes a flexible tool just like an application program. You can change the configuration of the operating system at any time, even right in the middle of an automated command sequence. You can do it manually, or operating system features can be employed to do it automatically as required.

You can change the whole operating system or just a part of it. Would you like a new command processor? No problem. With a simple command, **NZCOM** will load it — no assembly or configuration required. That new command processor will continue to run until you load another one. Want to have a different set of built-in (resident) commands? Again, no problem; **NZCOM** can load them in a jiffy. **NZ-COM** offers you a whole new world of flexibility and adaptability. It makes it easy for you to experiment and learn.

The basic operating system provided by **NZ-COM** is the widely popular ZCPR3 or **Z-System**, whose features will be described at some length throughout this manual. Until you try **NZ-COM**, it is almost impossible to imagine how easy it is to design and run the **Z-System** of your choice. Gone are the days of taking source code (no source code is needed), editing configuration files (you don't need an editor), assembling (you don't need an assembler), and patching (you don't have to know how to use the arcane SYSGEN and DDT programs). Simple relocatable object-code files for a particular module can be used by anyone on any system.

We hope that **NZ-COM** will open the world of **Z-System** to the general community, to those who have no interest in learning to assemble their own operating system or do not have the tools or skills. If you are at all intrigued by the **Z-System** (how could you not be?) and have a Z80-based computer running the **CP/M-2.2** operating system,[1] now is your chance to experiment.

1.1 The Benefits of NZ-COM

Here are some of the general benefits that you will derive from **NZ-COM**.

- The new command processing system is much more convenient, powerful and flexible. You can readily customize it to your own style and habits.

[1] If you have a **CP/M-Plus** computer, order Z3PLUS, the automatic **Z-System** for **CP/M-Plus** computers.

- Hundreds of excellent programs written for **Z-System** computers will now run on your computer.
- You can obtain public-domain and user-group **Z-System** programs from **Z-System** bulletin board systems (called **Z-Nodes**) and use them.
- New, high-quality user-group and commercial software is continually being written for **Z-System** computers, programs that you will finally be able to run.
- The **NZ-COM** system is ready-to-run. No assembly or technical installation is required.
- You can remove **NZ-COM** at any time and later restart it.

1.2 The New Command-Processing Features

The central function of the **NZ-COM** system is to enhance the control and processing of the commands that direct the **CP/M** system. It provides

- convenient editing of mistyped commands
- recall of previous commands for re-use, correction, or modification
- conditional execution of a sequence of commands, where execution can depend on:
  - the success/failure of programs previously executed
  - the availability and characteristics of necessary files
  - the user's privileges and other system characteristics
  - run-time input from the user
- named directories to conveniently segregate files and optionally protect directories with passwords
- command aliases, single commands that cause an entire command script to execute automatically

1.3 Other Major Features

In addition to command processing, **NZ-COM** provides

- a standardized method of using a terminal's full-screen capabilities, available to all applications wishing to use them
- a programming and user environment that fosters innovation and cooperation
- portability of programs among a wide variety of Z80-compatible computers

Taken together, these features mean that you can benefit from a very rich collection of programs, even if they were not written for your particular model of computer.

1.4 Using This Manual

There is a lot of material in this manual, and you will be glad to know that you do not have to read and absorb all of it before you can begin to make effective use of **NZ-COM**!

Chapter 2 tells you the mechanics of setting up the system and getting it running. Chapter 3 shows you the basics of what you can do with the system. It demonstrates the use of a number of the new commands and includes examples. These two chapters are essential reading.

Chapter 4 contains a lot more information about what Z-System is and about some of its most powerful features. In order to really take advantage of Z-System, you will certainly want to read this chapter before very long.

Chapter 5 describes some of the advanced capabilites of **NZ-COM**, particularly how to tailor the system to your personal needs and tastes. You probably will not be ready to make effective use of the material in this chapter until you have acquired some experience with Z-System. We suggest, though, that you skim this chapter just to see what is there, so that you have some inkling as to what advanced capabilities are available to you.

Chapter 6 is primarily a technical reference. It summarizes and collects material that appears elsewhere in the manual, such as the syntax for commands and the names and functions of files supplied with **NZ-COM**. It is the most technical of all the chapters. However, there is some material that appears only there, and at some point you should at least skim through the chapter.

Finally, Chapter 7, the Bibliography, contains references to other sources of useful and helpful information about **NZ-COM** and the **Z-System**. Every user should read this chapter.

As you venture forth with **NZ-COM**, we wish you an exciting and enjoyable experience. We hope that this manual and your experimentation with **NZ-COM** will help you understand better how computers actually work, and make your own computer easier to use, more productive, and more fun as well!

Chapter 2

Starting NZ-COM the First Time

2.1 Introduction

A key characteristic of the `Z-System` is its flexibility. Within a very broad range, you can customize it to your own needs and preferences. Of course, this flexibility means that there isn't a single "standard" setup. Getting started is very easy, but to take full advantage of the system you will have to take time to learn about its options and make some choices.

In fact, the `Z-System` is so flexible that veteran users are regularly discovering new capabilities that weren't envisioned in the original design. This dynamic, innovative quality is unusual for a microcomputer operating system, and it attracts both users and developers to continually make improvements.

To get you started, we've established a basic start-up configuration. It includes all available `Z-System` features and will allow you to become familiar with them. A full-up system like this one reduces your transient program area or `TPA` (the memory space available to application programs) by `5.5K`. Later (p. 44) you will learn how to create custom configurations so that you can make your own trade-offs between operating system features and memory consumption and can create `Z-Systems` that use as little as `1K` of `TPA`.

This manual is only an introduction to the **Z-System**. As you begin to make use of your new **NZ-COM** system, you will want to become familiar with the earlier, more extensive manuals and other written materials that cover the **Z-System** in greater detail. Several of these are listed in the Bibliography (Ch. 7).

2.2 Setting Up the Files

First, before you do anything else, make a complete, working copy of the NZ-COM distribution disk and put the original disk away in a safe place. Then put the working copy into one of your auxilliary disk drives (not drive A) and log into it. The file RELEASE.NOT, if present, contains additional information that was not available at the time this manual was prepared. You should examine it briefly before proceeding.

Next, use the TCSELECT utility to create a terminal-capability file that matches your terminal or computer. There is a suitable file for almost every reasonably popular terminal. To do this, type

```
    TCSELECT myterm<cr>
```

substituting for "**myterm**" an appropriate filename of your choice for your terminal.[1] TCSELECT will display menus of available terminals. Find your terminal or computer and select it. Once you have confirmed the choice, TCSELECT will write out a file called "**myterm.Z3T**".

[1] If you use the name "NZCOM", then it will later be loaded automatically, but it will spoil our plan to teach you how to do it with an alias (p. 14).

If for some reason your terminal or computer is not on the menu, you should first check the manual for your equipment to see if it uses the same screen codes as another, more popular terminal. If you are still unsuccessful, you should contact either the dealer from whom you purchased **NZ-COM** or one of the **Z-Helpers** named in the file **ZHELPERS.LST**. They will probably be able to assist you in creating the **Z3T** file. In the meantime, you can continue to use **NZ-COM** without defining a terminal, but you will not be able to use programs that require terminal information, such as **SALIAS**, **SHOW**, and **ZFILER**.

At this point it is assumed that you are running your computer with the standard **CP/M-2.2** operating system, possibly with a replacement command processor (such as **ZCPR1**). Although **NZ-COM** can run under a manually installed **Z-System**, this wastes memory and is not recommended.

Next, using **PIP** or another file-copying program, copy the following files to user area 0 of drive A:

```
NZCOM.COM      ARUNZ.COM      EASE.COM
NZCOM.LBR      ALIAS.CMD      ZFILER.CMD
MKZCM.COM      LX.COM         SHOW.COM
SALIAS.COM     IF.COM         myterm.Z3T
SDZ.COM        ZEX.COM        ZEX.RSX
LDIR.COM
```

Finally, you should copy one of the two versions of **ZFILER**. This program makes use of video highlighting on the screen. Because reverse-video and dim-video are so different, two separate versions of **ZFILER** are provided: **ZF-REV.COM** for reverse video and **ZF-DIM.COM** for dim video. You should choose the appropriate one for your type of terminal. For example, if your terminal uses dim-video highlighting, you would use a command line like the following:

```
PIP A:ZFILER.COM=ZF-DIM.COM<cr>
```

2.3 Running NZ-COM

Now you are ready for the quick two-step process of defining and loading the standard **NZ-COM** system for your computer. The definition step is performed by the program **MKZCM** (**MaKe ZCoM**). Just log into drive A and enter the command

```
    MKZCM<cr>
```

You will be presented with a menu screen that you will learn to use later (p. 47) to define custom **NZ-COM** systems. For now, we will just accept the default values. Press the "S" key for "**Save**", and, when prompted for a file name, enter "**NZCOM**". This is the name that **NZCOM** uses for its default definition. **MKZCM** will generate two files: **NZCOM.ZCM** and **NZCOM.ENV**. The significance of these files will be explained later (p. 43).

Now enter the command

```
    NZCOM<cr>
```

Your new **NZ-COM** system will be generated, loaded, and run automatically. The first time you do this **NZ-COM** will display

```
    STARTZCM?
```

You can ignore this message for now; it means that **NZ-COM** is unable to find the file **STARTZCM.COM**. We will take care of that on page 13.

The system is complete, except that it doesn't yet know what kind of terminal you have. We will now exhibit an additional capability of **NZCOM** by having it load the terminal descriptor. Type

```
    NZCOM myterm.Z3T<cr>
```

That's all you have to do to get **NZ-COM** running! You can now go on to the next chapter to learn more about the system.

2.4 Organizing Your Files

After you have learned about **NZ-COM** and **Z-System** and are ready to use them on a regular basis, you will want to organize the files on your disks.

If you have a hard disk or large-capacity floppy disk you will want to keep most or all of your **NZ-COM** and **Z-System** tools in directory **A0:** and make that your root directory (the last directory on your search path). Some **Z-System** users prefer to use directory **A15:** for these files, but **NZCOM**, as distributed, is designed to use **A0:** as the root of its path.

If you have smaller disks, put just the most-used files there and keep a separate **Z-Tools** disk at hand. Once **NZ-COM** is loaded, the **NZCOM.COM** and **NZCOM.LBR** files are only needed again if you want to change the system configuration, so they could be relegated to a different "boot disk".

Chapter 3

Getting Acquainted with Z-System

We suggest that you actually carry out the following steps to become familiar with a few of the commonly used features of the NZ-COM Z-System.

3.1 Creating an Alias

An alias is a single word (a command) that stands for a longer or compound command. To create an alias, run the SALIAS (Screen ALIAS) program. We'll create a startup alias, called STARTZCM, that will automatically install your terminal file in the NZ-COM system the next time you type NZCOM. (You've already seen that command name, STARTZCM; it appeared on the screen followed by a question mark when you ran NZCOM the first time.)

Type

```
SALIAS STARTZCM<cr>
```

to start the SALIAS program and to prepare the STARTZCM alias. You'll notice right away that this utility, like many Z-System tools, makes effective use of the display features of your terminal (and that's why it was important to select a terminal-capabilities file at the outset).

The SALIAS program is an alias editor, a kind of wordprocessor for alias scripts. You type in the text of the alias script and use control keys to edit your input. Most of the keys, including those for moving the cursor, are the ones used in WordStar. If you type Control-J, you will see a list of all of the command keys.

For this alias, the only command line you need to enter is

```
    NZCOM myterm.Z3T /Q<cr>
```

We added the "/Q" quiet option to suppress the extensive information that NZCOM can display when it loads something. Now, type Control-K followed by Control-X to exit and save the new file.

Try executing this alias now by typing

```
    STARTZCM<cr>
```

You'll see from the display that it indeed runs the NZCOM program and loads the Z3T file (again). So far this just saves us a little bit of typing. However, we selected this exact name for the alias because NZCOM itself will execute this alias automatically the next time you load it. Let's test things so far. Type

```
    NZCPM<cr>
```

Yes, the NZ-COM system is gone, and the standard CP/M-2.2 command processor is once again active. Now type

```
    NZCOM<cr>
```

to restart the NZ-COM system. This time it will automatically load your Z3T terminal file using the new STARTZCM alias.

The startup alias you just created is quite simple but nonetheless useful. An alias, however, can do much more. It can be a whole sequence of commands, with symbolic parameters for filenames, directories, and command line parameters. One of the things the startup alias can do, for example, is immediately set up a command search path that is different from the default path to which NZCOM.COM is configured. For a more elaborate example, see the alias created on page 19. For more information, consult the references in the Bibliography. And keep SALIAS in mind. It will come in handy for automating a wide variety of tasks on your computer.

3.2 The Available Commands

Now type

```
H<cr>
```

This will display all of the in-memory commands in the current **Z-System**. The screen will show a display like that in Table 3.1.

```
FCP
    IF        AND       OR        ELSE      FI
    IFQ       XIF       ZIF

CPR
    GET       GO        JUMP

RCP
    CLS       ECHO      ERA       H         NOTE
    P         POKE      PORT      R         REG
    SP        TYPE      WHL       WHLQ
```

Table 3.1: Resident commands displayed by the H command.

The commands that are displayed are the ones in memory at the time the H command is given. They will vary according to which FCP (flow command package), RCP (resident command package), and CPR (command processor) have been loaded. As you see, the H command (short for "HELP") is in the RCP package.

Later you will see that you can create versions of **NZ-COM** that use less memory by not including an RCP. Also, some RCP packages do not support an H command. In these cases, the **SHOW** command, described on page 23, can be used to show this (and much more) information.

3.2.1 RCP Commands

Let's look at the RCP (resident command package) commands first. We've just used the H command. Try another one by typing

```
ECHO This is a test.<cr>
```

Before we go on, we should point out that a semicolon (";") is used to separate multiple commands on a single line. Try

```
ECHO Test 1;ECHO Test 2;ECHO Test 3<cr>
```

`ECHO` is a simple but important command that is often employed to tell the user what is happening while sequences of commands are being processed by a script. We will see an example of this on page 19. Because command lines are always converted to upper case, your messages appeared in capital letters. The `ECHO` command will interpret the character sequence "%>" to mean "change to lower case" and the sequence "%<" to mean "change to upper case."

Now try the `SP` command, which tells you how much space remains on a disk:

```
SP<cr>
SP B:<cr>
```

The `P` command peeks at memory. Try

```
P 100<cr>
```

This is not too informative if you don't read computer code, but it is very useful and handy if you do. Now try the `POKE` command:

```
POKE 100 "This is a poke test<cr>
```

Be sure you typed the quotation mark before the text; it tells `POKE` that you are giving it text and not hexadecimal numbers. Now peek at the result:

```
P 100<cr>
```

Obviously, you want to reserve `POKE`s for special purposes, when you know you won't damage something (see p. 22). You can peek with abandon, however.

`ERA` should be familiar; it erases files. `TYPE` displays a file on the screen. `CLS` clears the screen (handy in scripts). `NOTE` is a do-nothing command used in scripts to include comments. PORT is a combination peek/poke for the computer's input/output ports; it reads a data byte from or writes a data byte to a specified I/O port. As with the poke command for memory, it should be used with great caution.

The R command resets the disk system; it is like a Control-C, but it can be included in a multiple command sequence. REG displays or modifies the Z-System software registers, which are used to communicate values between programs. The WHL and WHLQ commands are used with the Z-System security byte called the "wheel byte". You will find that many commands, especially risky ones like ERA, will not work when the wheel byte is off. WHLQ (WHeeL Query) reports the status of the wheel byte (try it!). WHL, when followed by the correct password, turns on the wheel byte; otherwise it turns it off. The default password is SYSTEM.[1]

3.2.2 FCP Commands

The Z-System can process entire sets of commands in an intelligent, automated way using the commands in the flow command package (FCP). Flow control takes some effort to understand, but it is so powerful and useful that we think you'll find it worth the effort to learn about it. In a nutshell, the purpose of flow control processing is to enable your commands to perform multiple levels of IF-ELSE-ENDIF testing (just as in high-level programming languages) in order to control which of a number of alternative commands are executed.

The main concept you need to grasp is that of "flow states". When Z-System first starts to run, no flow state is active (we call this the "null" flow state). Under control of the commands in the FCP module, up to 8 levels of flow state can become active. Each level can be either true or false (the null flow state acts like a "true" flow state). When the current flow state is true, commands execute in the normal fashion; when the current flow state is false, the command processor ignores all commands except those in the FCP. You will see how this works with some examples in a moment.

The IF command is issued with some type of test expression, which is evaluated as either "true" or "false". For either value, a new level of flow state is activated. If the previous flow state was false, then the new flow state is always made false, no matter what the result of the

[1] If you need to change it, the file ▮ZRCP.ZRL in ▮ZCOM.LBR can be patched. With your favorite patch utility, search for the string "SYSTEM" and change it as you like. If you need help doing this, consult your dealer or a Z-Helper.

conditional test was. On the other hand, if the previous flow state was true, then the new flow state will take on the value resulting from the test.

Here are some examples of conditional tests that can be performed by the `IF` command:

`IF ERROR`	tests whether a previous program set the program error flag
`IF REG 3 > 1`	tests value of a user register
`IF INPUT RUN WS?`	displays the prompt "`RUN WS?`" and waits for user response
`IF ~EXIST DIR:FN.FT`	tests for (non)existence of a file

Enter the command line

```
IF //<cr>
```

to get a condensed listing of the test conditions that can be used. As you can see, the number is quite large. Later you can experiment with them. Only the first two letters are actually used to determine the test condition (thus `EX` is the same as `EXIST`), and a tilde ("~") prefix reverses the sense of the test. Most of the tests are performed by the program `IF.COM`, which is loaded and run automatically by the `FCP`.

The `IFQ` (`IF Query`) command is included in the `FCP` to help you visualize the flow state and learn how to use flow control. We will use it now with some examples.

First try out the `IFQ` command. Unless you have already been experimenting on your own (in which case you should run the command "`ZIF`<*cr*>" to return the system to the null flow state), you should see the message "`IF None`", telling you that there is no active flow state (i.e., the "null" flow state). Now enter the command:

```
IF T<cr>
```

This test, like the `IFQ` command, is included largely as a learning aid. As you can probably guess, it returns a value of true. Now run `IFQ` again. The display will read "`IF T`", indicating that one flow state level is active and that it is true. Try entering a command, such as "`ECHO TESTING`<*cr*>" and note that it executes normally.

Now enter the command line:

```
IF F;IFQ<cr>
```

The display will show "IF FT". This tells us that two levels of flow state are active, with the current one false and the one under it true. Now try to execute the "ECHO TESTING<*cr*>" command. It is ignored.

The FI command, which is IF spelled backwards, terminates the currently active flow state and returns the system to the previous level (or to the null state). Enter the command line

```
FI;IFQ<cr>
```

several times and watch how the message changes. Once you are back in the null flow state, enter the command

```
IF T;IF F;IFQ<cr>
```

to get us back to where we were (with the current flow state false and one underlying flow state that is true).

Now we will learn about the ELSE command, which reverses the current flow state providing the underlying flow state is true. Enter

```
ELSE;IFQ<cr>
```

and note that the display has changed from "IF FT" to "IF TT". Commands will now run again. Enter it a few more times, and note how it toggles the flow state.

At this point you know enough about flow control and the FCP commands to experiment on your own. When you are finished, enter "ZIF<*cr*>" to zero out the flow state.

Before leaving this subject, we would like to take you through one last example, one that will give you a hint as to how flow control can be combined with aliases to make very powerful commands.

Use SALIAS to create an alias called TEST by entering the command

```
SALIAS TEST<cr>
```

When the input screen appears, enter the following sequence of commands:

```
IF EX $1
ECHO FILE $1 EXISTS
IF EQ $1 *.COM
ECHO COM %>FILE: %<YES
ELSE
ECHO COM %>FILE: %<NO
FI
ELSE
ECHO FILE $1 NOT FOUND
FI
```

Note that we are using the special character sequences that ECHO recognizes as signals to change between upper and lower case (see p. 16). When you are done typing in the commands, press Control-K Control-I. Note how SALIAS conveniently indents the commands to show the nesting of flow states. Now save the alias by entering Control-K Control-X.

The expression "$1" in the alias stands for the first item given on the command after TEST when the alias is used. Try entering the following commands and see what happens in each case:

```
TEST NZCOM.COM<cr>
TEST NZCOM.LBR<cr>
TEST NOFILE.JNK<cr>
```

3.2.3 CPR Commands

The standard version of the command processor has only three commands. Most of the code in the CPR is needed to perform sophisticated processing operations on *your* commands rather than to execute commands of its own. Most of the resident (in-memory) commands are provided by the RCP, which offers you the advantage of being able to change the commands by loading different RCP modules depending on your current needs.

The commands in the CPR are ones that make use of code already there for other purposes. GET and GO/JUMP are basically the two halves of the normal load/execute process that is performed whenever you specify a command that refers to a COM file. All three commands can be dangerous; if used incorrectly they can result in a system crash that will require rebooting. Used correctly, they can be very powerful and handy.

The GET command performs only the loading process. It offers more flexibility than the normal command loader. First, *you* specify the memory address to which the file is loaded, and, secondly, any type of file (not just a COM file) can be loaded. There is nothing to stop you from loading a program to a wrong address (even an address where the operating system resides!) or from loading a file that is not a program.

Try entering the command line

```
GET 200 NZCOM.LBR;P 200<cr>
```

The library will be loaded into memory starting at 200H, and the peek command will show you the beginning of the file. If you have ever wondered what a library file really looks like, now you can see. GET and P together can make a handy learning tool.

Just as GET performs only the loading of a file, GO and JUMP perform only the execution of what is already in memory. GO automatically executes whatever is at address 100H. JUMP is similar but lets you specify the execution address (GO is equivalent to JUMP 100). For normal COM-file programs, the combination of GET and GO performs an ordinary execution. Thus the command sequence

```
GET 100 SDZ.COM;GO *.LBR<cr>
```

does exactly[2] the same thing as

```
SDZ *.LBR<cr>
```

Why, then, would one ever want to use these separate commands? We will give two examples. First, the GO command can be used to execute a program repeatedly without having to spend the time loading the file each time from disk. Try

```
SDZ *.COM<cr>
```

and then

```
GO *.LBR<cr>
```

[2] Well, almost exactly.

Caution: some programs are not designed to be reexecuted and could cause trouble if used this way. Most **Z-System** tools can be reexecuted, and those that cannot often exit gracefully at least. Other programs may be less forgiving, so you should experiment cautiously.

A second example of the use of **GET** and **GO** is the technique called "**GET**, **POKE**, and **GO**" or simply **POKE&GO**. The idea here is to load a file using the **GET** command, modify it using the **POKE** command, and then run it using the **GO** (or **JUMP**) command. For example, if your wordprocessor (let's say it is called **WP.COM**) keeps its right margin value at address **83BH** and normally uses a value of 76, you could make an alias named **WP60** with the following alias script to give you a version with a right margin value of 60:

```
GET 100 WP.COM;POKE 83B 3C;GO $*<cr>
```

Note that the poked value is given in hexadecimal form (**3CH=60**) and that the expression "**$***" in an alias script is replaced by whatever the user put on the command line after the name of the command. Thus the command

```
WP60 NEWFILE.DOC<cr>
```

would run **WP** with the file **NEWFILE.DOC** but with a right margin of 60. To do this without **Z-System**, you would have to have a second version of the entire wordprocessor. An **SALIAS** alias is only **1K** long; an **ARUNZ** alias (p. 33) for this function takes only about 40 bytes of disk space.

3.2.4 Transient Commands

"Transient command" is the name Digital Research (publisher of **CP/M**) gave to commands that do not reside in memory but are loaded from disk. These files all have a file type of **COM**.

In addition to the normal **COM** files supported by **CP/M**, there are several special types of **COM** file supported by **Z-System**. These files are designated as type-1, type-3, and type-4.[3]

Type-1 files are similar to standard **CP/M COM** files except that they have a special header at the beginning into which the command processor

[3] Yes, there are type-2 files also, but they are rarely used today.

inserts the address of what is called the Z-System environment descriptor. This is an area in memory that contains a complete description of the Z-System configuration. With that information, a program can make use of the special Z-System facilities.

Type-3 programs are similar to type-1 programs except that, unlike standard CP/M programs, they do not necessarily get loaded to and run at the standard address of 100H. Their execution address is included in the special header, and the command processor automatically loads them to the address specified there. For example, the program ARUNZ.COM included in the NZ-COM package is a type-3 program that runs at 8000H.

Type-4 programs are special Z-System programs derived from PRL (so-called Page ReLocatable) files. They contain relocation information to allow them to run at any address. The command processor automatically calculates the highest address in memory at which they can run with the existing configuration and then loads and runs them at that address.

The type-3 and type-4 programs were invented for transient programs that function as resident parts of the system (e.g., extended command processors, shells, and error handlers) or as transient versions of common resident commands (e.g., ERA.COM or REN.COM). Since the Z-System on its own often invokes these commands automatically, you cannot always predict when one of them will run, and it is handy if they do not interfere with the lower parts of memory where user programs run.

The special Z-System programs usually cannot run properly under CP/M. Because they were designed to run on computers with *permanently* installed Z-Systems, most of them do not include code to prevent attempts to run them under ordinary CP/M. Most of the ones supplied with NZ-COM, however, will terminate gracefully from such an attempt. Some of them will display a message; others will simply cause a warmboot.

3.3 More About the System Configuration

We mentioned earlier that under some circumstances you would not be able to use the H command to see what resident commands are available. The **Z-System** utility program **SHOW** can always be used to display this and a great deal of other information about your system. The best way to learn about the configuration of your **NZ-COM** system is to run **SHOW** and experiment with its menu selections. Some of the things you can do with **SHOW** are:

- display the memory map of the system
- show the commands supported by the **CPR**, **FCP**, and **RCP**, with an indication of which commands require the wheel byte to be on
- show the special features implemented in the **CPR**
- show the command search path and directory names
- display the information in the shell stack and message buffer

Chapter 4

Learning More About Z-System

The `Z-System` divides into

- resident (in-memory) components
- automatic tools and shells — the special programs that the command processor runs for you
- other tools — programs that are controlled by or make use of the resident components

The components included in the `NZ-COM` package and described below constitute a fairly basic set, and yet they are already rich in features. The following descriptions are included here for your reference, but you certainly don't need to absorb all of it at the first reading.

4.1 Resident NZ-COM Components

- The `ZCPR34` command processor: It interprets commands and loads programs.
- An `RCP` (Resident Command Package): It includes an extended set of commands (see p. 15) that can be quickly changed.

- An FCP (Flow Command Package): It tests logical conditions and manages the flow state of the system so that commands can be conditionally executed (see p. 17).
- An IOP (Input/Output Package): It allows I/O drivers to be loaded as needed to provide such facilities as keyboard macro definitions or redirection of screen or printer output to a disk file.[1]
- An NDR (Named Directory Register): It contains a table that associates directories (drive/user-number pairs) with names and optional passwords.
- A PATH: It indicates the sequence of directories to be searched to find a program.
- A Z3T terminal descriptor: It contains the control sequences needed by your terminal for positioning the cursor, clearing the screen, etc.
- Other components of the Z-System environment: These include the command-line buffer, a message buffer, a shell stack, the wheel byte, the external file control block, and a command processor stack.

4.2 Automatic Commands

Several Z-System tools are automatically caused to execute under specific conditions.

4.2.1 Extended Command Processor

When the command processor cannot process a command as either a memory-resident command or a disk-resident file, then it will pass the command to another program called the Extended Command Processor (ECP). The standard NZ-COM version of ZCPR34 always tries to load as the ECP a program with the name CMDRUN.COM in the root directory (the final directory in your search path). The distributed NZ-COM system includes two programs that are often used as ECPs:[2]

[1]This subject is not covered any further in this manual. Contact the dealer from whom you purchased NZ-COM for information about available IOPs.

[2]There is even a way to have both of them function together as the ECP.

- **ARUNZ** (**Alias-RUN-for-Z-system**), which performs a function very similar to that of the aliases created by **SALIAS**, except that
  - a large number of aliases can be defined in a single text file called **ALIAS.CMD**, and
  - much more extensive parameter processing is supported.

  ARUNZ can function as a very powerful command generator and translator.
- **LX** (**Library eXecutive**), which gets commands out of a library called **COMMAND.LBR**. Putting **COM** files into **COMMAND.LBR** saves both directory space and file space on the disk. **LX** is especially useful on computers with low capacity diskettes or diskettes whose format allows too few file names in a directory.

To make one of these programs function as your extended command processor, copy it to the filename **CMDRUN.COM**.[3] For example, to use **ARUNZ** as the **ECP**, type the command

```
PIP CMDRUN.COM=ARUNZ.COM<cr>
```

You should do that now, since in a little while (p. 33) we will be using **ARUNZ** for some examples. Later, if you want to try using **LX** instead, type the command:

```
PIP CMDRUN.COM=LX.COM<cr>
```

4.2.2 Error Handler

When the command processor cannot process a command because the extended command processor cannot be found or because the extended command processor also cannot process the command, then the **CPR** will invoke an error handler. Advanced **Z-System** error handlers can display information about what is wrong with the command and can allow you to edit the command line to correct any mistakes.

Any error handling command line can be loaded with the **ERRSET** utility (not supplied). However, most error handlers will install themselves

[3] It must be in the root directory (last element on the path). Unless you have changed it, this will be **A0**, but you can run the **PATH** command to see what the root is.

automatically if you invoke them manually as a command (they can tell whether they have been invoked by the user or by the **CPR**). The **NZ-COM** package includes the **EASE** error handler, which is described in the next section.

4.2.3 Command Shells

Shells are tools that are automatically executed whenever the command processor has completed all the commands it was given — including batch commands from **SUBMIT** and **ZEX** — and is ready for new command input. If no shell is loaded, then the normal prompt will appear on the screen, and the user will be asked to enter the next command. If a shell is active, then it will run instead. Most shells generate commands *for* the user and pass them on to the command processor. We have included several shells with the **NZ-COM** system.

The shell command line (sometimes along with other information used by the shell) is kept in a memory buffer called the "shell stack". The shell stack in the **NZ-COM** system can normally hold up to four shell command lines, each up to 32 bytes long, but these values can be changed using **MKZCM**. The command that was placed on the stack most recently is the active shell command. When that shell is terminated (usually by a special command response to the shell's input request), its command line is removed from the shell stack (the common term is "popped"), bringing the next most recently invoked shell to the top of the stack and making it active. This mechanism allows shells to be "nested".

The EASE Error Handler and Shell

EASE (**Error And Shell Editor**) is a perfect example of the shell concept. It is also particularly handy. When **EASE** becomes active, it puts up a prompt like the one normally presented by the **CPR** but with an extra ">" to remind you that **EASE** is running. There are three important differences, however.

1. When you enter a command, you have at your disposal a very powerful editor, what amounts to a wordprocessor for the command line. You can move forward and backward through the command line and can insert and delete characters.

2. When you issue your command by pressing the carriage return key, the command line is saved in a history file before it is passed to the command processor.

3. Commands entered in the past can be retrieved from the history file, edited as you wish, and then executed again.

After you see how convenient these features are, you may want to add EASE to the end of the list of commands in your STARTZCM startup alias.

EASE is not only a shell but an error handler as well. Combining these functions in a single program was a natural thing to do with EASE, since editing a new command line and correcting an old, mistaken one involve the same functions.

The version of EASE included with NZ-COM is a type-3 program (see p. 23) that executes at an address of 8000H (32K). As a result, unless the user's last program was longer than 32K, EASE will not interfere with rerunning it using the GO command.

EASE is installed as both a shell and an error handler by entering the simple command[4]

```
EASE<cr>
```

We recommend that you do this now and try some examples. Type in the following commands one at a time:

```
ECHO THIS IS COMMAND 1<cr>
SDZ *.COM<cr>
ECHO THIS IS COMMAND 3<cr>
LDIR NZCOM<cr>
```

When the next prompt appears, type Control-B (for Back). The previous command will appear. Try entering Control-B several more times. Then try Control-N (for Next); it will move you forward through the history. Continue moving through the history until the command "ECHO THIS IS COMMAND 3" is at the prompt.

Now we will edit the command. Most of the editing command keys are those from WordStar. Use Control-S and Control-A to move left one character or one word. Use Control-D and Control-F to move right.

[4] EASE can be installed as a shell only using "EASE S" or as an error handler only using "EASE E".

When you get to the "3", enter Control-G to delete (Gobble) it. Then enter "5" and a carriage return.

Now we will try a recall search. At the prompt, type in only the letters "ECHO" (no carriage return). Then press Control-O. Note that EASE has located for you the most recent command line that started with "ECHO". Press Control-O again. EASE will find the next previous occurrence. This feature is remarkably handy!

EASE will continue to record all commands (except very short ones) until you remove it (by typing Control-Q Control-_ [underscore]). The history is stored in the file EASE.VAR. This file will continue to grow, and from time to time you might want to erase it (but not while EASE is running) and start over. There is also a program called VARPACK that can pack the VAR file, keeping only a fixed number of recent commands.

You can generate a file that lists all of the control keys used by EASE by entering the command

```
    EASECMD EASE<cr>
```

This will produce the file EASE.CMD, which you can type or print.

The ZFILER Shell

ZFILER illustrates how a shell can change the user interface completely. Instead of a command line, it provides a "point-and-shoot" environment. To see ZFILER in action now, enter the command

```
    ZFILER<cr>
```

You do not have to remove EASE before running ZFILER. Shells can be "nested" under Z-System. If EASE was running when ZFILER was invoked, then EASE will become inactive while ZFILER is running but will return automatically upon exit from ZFILER.

You will see a full-screen display of the names of the files in the current directory and a pointer pointing to the first file. Use the WordStar-diamond control keys to move the pointer around (control keys E, X, S, and D to move up, down, left, and right, respectively). Position the pointer to a text file, such as EASE.CMD, ALIAS.CMD, or ZFILER.CMD. Press the "V" key to "View" the file. ZFILER has many such built-in functions, and you can see a listing of them by pressing the slash ("/") key for help. Press slash again to return to the file display.

Try tagging some files using the "T" key. Now experiment with a "Group" operation by pressing the "G" key. As you can see, a number of functions can be performed not only on single files but on all the tagged files. Try selecting "Copy" by pressing "C". When prompted for the destination directory, enter

```
A1<cr>
```

Note that the colon after directories is optional inside ZFILER.

Let's verify that the files were copied. Log into the A1: directory using the "L" or "Log" key. Answer the prompt with

```
A1<cr>
```

Delete these duplicate files using the "D" key. Now let's return to directory A0:, but in a slightly different way. Directory A0: has the name COMMANDS. Press the "L" key and answer the prompt with

```
COMMANDS<cr>
```

As you see, named directories can be used just as well as drive-user designations. This is true throughout the Z-System.[5]

The built-in functions are not the only ones ZFILER can perform. A set of "macro" commands can be defined, along with a user information screen, in the file ZFILER.CMD. We have supplied a very simple one to illustrate what can be done. Move the file pointer to the file NZCOM.LBR. Now press the escape key. You will be prompted for a macro. Later, when you know the keys associated with the macro functions, you can enter the appropriate key immediately. Alternatively, you can press escape again (do that now), and the information screen will appear. You will see that the "L" macro will give you a directory of a library file. Press "L" now and watch what happens.

ZFILER generated a command line for you and passed it on to the command processor. After that command line had completed execution, the ZFILER shell was reinvoked automatically. In this case, it knew that it had just finished executing a command that put some information on the screen, so it waited for you to press any key before it restored the display of file names.

[5] That is, for the ZCPR34 command processor and for the Z-System-specific utility programs.

It is even possible to run macro commands on groups of tagged files. Tag a few files and then press "G" to start a group operaton. When asked for the operation, press the escape key to invoke macro processing. As before, you will be prompted for the macro key. You can again enter the key immediately or press escape to see the information screen. Press "E". ZFILER will now automatically generate a file called ZFILER.ZEX in directory A0: and then initiate a ZEX batch operation. The "E" macro, as you will see, is a harmless command line using ECHO to show how ZFILER can parse the name of the designated file (pointed to or tagged).

You can now exit from ZFILER by pressing the "X" or "eXit" key. That will terminate the shell. If EASE was running before you invoked ZFILER, then EASE will return. Otherwise, you will again see the prompt from the command processor itself.

Other Shells

There are quite a few other shells not provided with the standard NZ-COM package that offer a wide variety of functions. Some are in the same family as ZFILER. In contrast to ZFILER, which shows the files on the screen and displays the available macro commands only on request, the MENU shell shows only the command options. It supports multiple levels of nested command menus. The VMENU and FMANAGER shells are half way between MENU and ZFILER. They show the files in the upper half of the screen and the command options in lower half.

A different kind of shell is the patching shell ZPATCH. It enables you to edit the binary image of a file. For example, you might be customizing the configuration options of a program. From within ZPATCH you can execute the modified program to see the results of your handiwork and then return automatically to ZPATCH, positioned to the exact byte you were at when you left.

Another set of shells (SH, GETVAR, FOR-NEXT) allow one to create and use "shell variables", much like the "environment variables" in MS-DOS.

4.3 Other Z-System Tools

There are many, many other Z-System tools, far too many to begin to describe them all here. In fact there is probably no Z-System user, no matter how expert, who knows them all! To get you started, we will describe a few particularly important ones. We did not have you copy all of them from the NZ-COM working disk (p. 8) into your A0: directory. You can either copy them now or run them from the work disk.

4.3.1 ARUNZ

A Z-System alias or alias script is a sequence of one or more commands that can be invoked by a single name. As we saw in Section 3.1 (p. 13), stand-alone aliases are short COM files that can be created by SALIAS, the screen oriented alias editor.

ARUNZ provides an alternative way to define and use aliases. Instead of making each alias its own file, ARUNZ allows one to define numerous aliases in a single file called ALIAS.CMD. The ARUNZ program extracts a designated alias script from the ALIAS.CMD file, expands any symbolic parameters in the script, and passes the resulting command line to the command processor.

The ALIAS.CMD file is an ordinary text file that is created with any text editor or wordprocessor (in non-document mode). Since all the scripts are combined in a single file, they require very little disk space. As a result, you can easily have dozens or even hundreds of them, and they can greatly enhance and ease your computing life.

ARUNZ aliases are invoked by a command of the form:

```
ARUNZ ALIASNAME COMMAND-ARGUMENTS<cr>
```

ALIASNAME specifies which of the scripts in ALIAS.CMD to use, and COMMAND-ARGUMENTS supplies other information needed by the scripts, such as file names. The real power of ARUNZ comes when it is renamed to CMDRUN.COM and serves as the extended command processor (see p. 26). Then the command can be issued simply as:

```
ALIASNAME COMMAND-ARGUMENTS<cr>
```

An example might make this clearer. One line in the ALIAS.CMD file reads:

```
D=SD    sdz $td1$tu1:$tn1*.$tt1* $-1
```

This may look rather forbidding with all those dollar signs, but, when we take it apart piece by piece, it won't be so bad. First of all, the name of the alias is either "D" or "SD". **ARUNZ** allows multiple names to be assigned to a single script. The program run by this alias is SDZ, the super-directory program.

The interesting part of this script is the way the command arguments are handled. To illustrate what happens, let's assume that we entered the following command:

```
D S.C<cr>
```

The parameter expression "**$td1**" means the drive specified in the first token. In our example, the first token is "S.C". Since we did not specify a drive in that token, the current drive, A, is assumed. Similarly, "**$tu1**" means the user number specified in the first token. Again, none was given, so the current user number, 0, is assumed. Next we have a colon. So far the command line reads: "SDZ A0:".

The next parameter, "**$tn1**" indicates the file name part of the first token. In our example, this is "S". The next characters in the script are an asterisk and then a period. Then we have the parameter "**$tt1**", which stands for the file type in the first token, or "C". Then we have another asterisk in the script. So far the command line reads: "SDZ A0:S*.C*".

The final part of the script is the parameter expression "**$-1**". This means the entire command argument less the first token ("**$-2**" would omit the first two tokens). In the example, there are no other tokens, but one might have used an option to SDZ such as "/C" to get the file size as a record count instead of in kilobytes. Thus the command entered as

```
D S.C<cr>
```

has become, courtesy of **ARUNZ**,

```
SDZ A0:S*.C* <cr>
```

Try entering the alias command and see what happens. The script gives us a very convenient way of automatically making the file specification to the SDZ program into a wildcard specification, saving us the nuisance of having to type all the asterisks. In the example, we get all files whose name starts with "S" and whose type starts with "C", including SALIAS.COM, STARTZCM.COM, and SDZ.COM.

To learn more about how to write and use ARUNZ aliases, consult the references in the Bibliography. But note that the version of ARUNZ included with NZ-COM is more recent than the one described in the reference there. Many new parameters have been added, and a few old parameters have been changed. So read the update documentation provided with the NZ-COM package.

4.3.2 HELP

Most of the Z-Tools will provide a terse reminder of their command line syntax if you enter them with just a double slash, as in

```
NZCOM //<cr>
```

Much more complete on-line information can be obtained using the Z-System HELP utility. It provides an organized method (tree structured) for searching special text files with a file type of HLP and displaying the information in them. You run HELP as follows:

```
HELP<cr>
HELP filename<cr>
```

The first form uses a file with the name HELP.HLP; the second form uses one with the name "filename.HLP". For example, if you want help with SALIAS and there is a help file for it, enter the command

```
HELP SALIAS<cr>
```

Help files for a number of other Z-System programs can be found on the Z-Nodes. You can also write your own HLP files.

4.3.3 Library Tools

Z-System library tools allow you to put an entire set of files into a single library file and then access the individual members of the library as needed. This is effective for keeping short and related files organized and for saving directory and file space on a disk. Programs like NZCOM and JetLDR are able to access multiple members of a library more rapidly than individual files could be read. Libraries are also widely used on remote access CP/M systems (RASs or RCPMs). The essential NZ-COM system files, for example, are all kept in NZCOM.LBR.

The two multi-purpose library tools are NULU (new library utility) and VLU (visual library utility). These two programs can create a library, insert and extract files, type files to the screen, and a host of other functions. NULU is a generic CP/M program readily available from RASs; VLU is a Z-System-specific tool and works somewhat like the ZFILER shell. We have included it with the NZ-COM package.

Library tools that perform more limited functions are:

LPUT	put files into a library
LGET	get files out of a library
LDIR	display the directory of a library
LX	extract and execute a member
LT	display a member (library type)

4.3.4 File Compression

Text compression and decompression tools allow files to be "crunched" to a fraction of their original size for more compact storage and faster transmission by modem. Crunched files have a "Z" as the second letter of their file type (or a file type of ZZZ if the original file had no file type at all). CRUNCH and UNCR do the job. We have provided them with the NZ-COM package.

LT.COM is quite versatile. It was listed with the library tools because it can display text in library member file. It can also display individual files, and in both cases the file can be crunched as well as normal text. It even figures out automatically whether uncrunching is needed. To type a library member to the screen, use the following command form:

```
    LT LBRNAME FILENAME.TYP<cr>
```

To display an individual file, use the form:

```
    LT FILENAME.TYP<cr>
```

4.3.5 Named Directory Tools

The ability of **Z-System** to associate names with directories can make life easier, especially when the computer has a hard disk. **EDITNDR**[6] is a tool for editing the assignment of directory names. It can be used in either interactive mode or from the command line. Let's use it interactively first. Enter the command

```
EDITNDR<cr>
```

You will get a prompt inviting you to enter "?" for help. Why not accept the invitation! Now try entering a carriage return. **EDITNDR** will now show you the names that are currently assigned. Now enter the line

```
A1:TEXT<cr>
```

This will assign the name TEXT to user area 1 on drive A. Press <*cr*> to see the new listing. You may not notice the assignment for **A1:** because it is not with the other assignments for drive A. Enter the command "S" to sort the listing and then another <*cr*>. Now the names are in order. If you look at the built-in help screen, you will see that **EDITNDR** is very flexible in the syntax it will accept. When you are done experimenting, enter "X" to exit from EDITNDR.

The **ALIAS.CMD** file contains an alias to facilitate assigning names to directories. You just enter the command

```
NAME DU:DIRNAME<cr>
```

where DU is the drive/user to be given the name DIRNAME. If "DU:" is omitted, the current directory will be given the name. Try entering

```
NAME SYS<cr>
```

and see how the command prompt changes. You can look at the contents of **ALIAS.CMD** to see how this alias works.

The named directory assignments made by **EDITNDR** are temporary. You can use the **SAVENDR** utility to write the assignments out to a file.

[6] This was originally released as EDITND. We have changed its name to make it consistent with SAVENDR.

The file can be loaded by the **STARTZCM** alias to reinstall your names. For example, the command

```
SAVENDR SYS<cr>
```

will create a file called SYS.NDR in the current directory. This file can be loaded by **NZCOM** using the command

```
NZCOM SYS.NDR<cr>
```

NZCOM can load many files at once, so you can combine this with the loading of the Z3T file as follows:

```
NZCOM myterm.Z3T SYS.NDR<cr>
```

If you put those two files inside NZCOM.LBR using LPUT, you can then load them even faster using

```
NZCOM NZCOM myterm.Z3T SYS.NDR<cr>
```

For the ultimate, you can rename the two files to NZCOM.Z3T and NZCOM.NDR before putting them into NZCOM.LBR. Then **NZCOM** will load them automatically when it starts, and you will not have to include any command in the **STARTZCM** alias to do it.

4.3.6 Other Tools

A number of other utility programs may be included with your **NZ-COM** distribution diskette. You should experiment with them on your own. Enter the command with "//" after it to see any built-in help information. If there is a help (HLP) file with the same name as the command, you can use HELP to learn more about the command.

4.4 Command Hierarchy

The ZCPR34 command processor is highly sophisticated in the way it processes commands. Two steps are involved in command processing. First, a command must be acquired from some source of commands. Then, that command has to be processed in an appropriate way. We will now describe briefly these two aspects of command processing.

4.4.1 Command Acquisition

The ZPR34 command processor acquires its next command according to the following hierarchy:

1. the multiple command line buffer
2. a ZEX batch command
3. a SUBMIT file
4. a shell command
5. user input

Commands in the multiple command line buffer have the highest priority. If the memory-based batch processor ZEX is active, its command stream is treated as an extended multiple command line. Once the command line buffer and any ZEX stream are exhausted, the command processor will check to see if a SUBMIT job is running, in which case it will read its next command line from the SUBMIT file. This means that a SUBMIT command sequence acts, in effect, like a further extension of the multiple command line. When all of these sources of commands have been exhausted, the command processor will return to any shell that has been engaged. Only if there is no shell will the command processor turn, as a last resort, to the user!

4.4.2 Command Resolution

Once ZCPR34 has the next command, it resolves the command according to the following hierarchy:

1. scan the FCP commands
2. scan the RCP commands
3. scan the command processor's build-in commands
4. search for a COM file along the search path
5. invoke the extended command processor
6. invoke the error handler

Several factors can modify the resolution hierarchy. If the current flow state is false, only FCP and shell commands are scanned. RCP and CPR resident commands are ignored; transient commands and the extended command processor (ECP) are ignored. Shells will still run.

Several prefixes can modify the resolution hieracrchy. A slash or space prefix directs the command immediately to the extended command processor. When you know that your command is one that must be processed by the ECP, then you can save the time otherwise required to search the path for a transient command. The slash or space prefix can also be used when there are both transient and ECP versions of the same command, and you want the ECP version.

A prefix of just a colon or a period will tell the ZCPR34 command processor to skip resident commands.[7] It also adds the currently logged directory to the command search path for this command. If a transient command is not found, the ECP will still be invoked. These prefixes can be used when the path does not include the current directory ($$) and you know that the command is there. It can also be used to force the execution of a transient program with the same name as a command resident in the RCP or CPR.

An explicit directory prefix of the form D:, U:, DU:, or DIR: has the same effect as the colon or dot prefix except that the command is searched for *only* in the specified directory and the ECP is not invoked if the command is not found there.

If an error occurs, the command processor first attempts to invoke an error handler. If none has been installed or if the one installed cannot be found, it prints the entire remaining command line in the multiple command line buffer followed by a "?".

[7] This does not apply to resident commands in the FCP. Flow control commands are checked for with *every* command, no matter what prefix might be present (a colon, a slash, DU:, DIR:, etc.).

Chapter 5

Getting More Out of NZ-COM

In the previous chapters, you learned about the basic operation of the Z-System and of NZ-COM. What you have seen, however, is only the tip of an iceberg! NZ-COM offers a vast array of possibilities, and we will try to introduce them to you in this chapter. In fact, there are surely ways to use NZ-COM that even we have not thought of, so we certainly cannot claim to be offering an exhaustive treatment here.

5.1 Alternative Invocation Commands

So far, you have learned the simplest way to get the NZ-COM system running using the command

```
NZCOM<cr>
```

NZ-COM can also be loaded in the following ways:

- from a SUBMIT script
- with a special multiple command line

NZCOM command lines can be included in SUBMIT files that will be run from either CP/M or Z-System. Including the NZCOM command in a SUB file that is invoked as part of a CP/M cold-boot procedure can be useful for automatically loading NZ-COM whenever you turn your computer on. See page 55 for information on more elaborate ways to use SUBMIT with NZ-COM.

NZ-COM can also be started in a way that allows you to pass a Z-System multiple command line to it. This is done by including the command line at the end of the NZCOM command after a semicolon, as in the following example:

```
NZCOM nzcom-tail;command 1;command 2;...<cr>
```

where "nzcom-tail" represents any file names and options for the NZCOM command itself. This will start by performing the NZ-COM system load corresponding to the command

```
NZCOM nzcom-tail<cr>
```

However, the command line following the ";" will be executed instead of the usual STARTZCM command. For example,

```
NZCOM ;MYSTART<cr>
```

will load the standard NZ-COM configuration and then run the alternative startup command MYSTART.

The multiple-command-line loading technique and the SUBMIT loading technique can be combined.

5.2 Customizing Your NZ-COM System

There are many ways to customize your NZ-COM system and adapt it to your needs and tastes. We will consider two kinds of changes, those that you make more-or-less temporarily by using utility programs and those that you make more-or-less permanently by creating entirely new system configurations.

5.2.1 Temporary Changes

After you become more familiar with the basic NZ-COM system, you will undoubtedly want to change a number of system characteristics.

Some of these changes will be temporary ones. For example, you might change the command search path to include a directory with files that you do not use all the time — and thus do not usually include in the path — but which you need for the current task. For this you would use the **PATH** utility. When the task is finished, you would use the utility again to restore the path to its usual configuration.

Other parameters that are often changed temporarily are the characteristics of the printer or **CRT** screen. Many **Z-System** programs will adjust automatically to these "environment" values. For example, if you are printing with a spacing of 6 lines per inch, you might set the printer data to 66 lines per page with 58 lines of text. When printing with a spacing of 8 lines per inch, you would use values of 88 and 78 instead. The **CPSET** utility is used to specify these characteristics.

Sometimes you will want such changes to be in effect for an extended period of time, but perhaps not permanently. One example of this would be if you want to use a path that is different from the one created by **NZCOM** (see p. 69). You would then include a **PATH** command in your startup alias along with any other configuration commands. These changes will then take place automatically when you first load the **NZ-COM** system.

5.2.2 NZ-COM Descriptor Files

Although, as we just described, you can easily make temporary changes in some of the system characteristics, there will probably come a time when you will want to make some of those changes permanent. It is also likely that you will at some point want to change system characteristics that cannot be changed using utility programs, such as the sizes of the system buffers (**RCP**, **FCP**, and so on). For example, if you have a hard disk, the standard **NDR** with its capacity of only 21 names will not be large enough, and you will want to increase the capacity to 28, 42, or even more names. We are happy to report that *these changes can be made very easily.*

The characteristics of **NZ-COM** systems are defined by "descriptor" files in two alternative formats, one with a file type of **ZCM** and the other with a file type of **ENV**. Files of both types are created by the **MKZCM** utility (see p. 47). You can then work with whichever format you prefer; the other file can be erased if you wish.

The **ZCM** files are ordinary text files in the form of a symbol table that can be edited quite easily using any text editor or wordprocessor (in non-document mode). At this point we recommend that you examine the default descriptor file by entering the command

```
    TYPE NZCOM.ZCM<cr>
```

The **ENV** files are binary files in the form of **Z-System** "environment descriptors". You can examine the default **ENV** descriptor by entering the **ARUNZ** alias command

```
    LOOK NZCOM.ENV<cr>
```

The screen display will not mean much to you unless you are already familiar with **Z-System** programming internals. The only advantage of the **ENV** form is that the file is only one record long, whereas the **ZCM** file is five records long. There are a very few instances where an **ENV** file is needed to configure some other **Z-System** program. In general, we recommend working with the **ZCM** file.

5.2.3 Modifying the Descriptor Files

The **MKZCM** system definition utility can only make changes in the memory allocation map; many other characteristics can be changed only by editing the system descriptor file. Some of those changes should be made to make the system descriptor consistent with the characteristics of your hardware. It is a good idea to make those changes in the **NZCOM.ZCM** file at the outset. By doing so, you will generally not have to make any further changes except by using **MKZCM**.[1]

Some characteristics that you will almost surely want to modify so that they match your hardware are discussed below.

Drives and User Numbers

The drive vector (DRVEC) is a 16-bit word that tells the system which logical disk drives to recognize. The lowest order bit corresponds to drive A. Thus if you had floppy drives A and B, hard drive partitions F, G, H, and I, and RAM drive M, the drive vector in binary would be constructed as follows:

[1]This is because **MKZCM** always reads the values from the currently running system (if **Z-System** is running) and incorporates those values into new **ZCM** and **ENV** files.

```
Drive:    PONM LKJI HGFE DCBA
Bit:      0001 0000 1111 0011
```

In hexadecimal notation, as required for the ZCM file, this would be 10F3 for this example. The default value generated when MKZCM is run from CP/M is FFFF (all 16 CP/M drives allowed).

There is a second symbol that defines the highest logical drive to be recognized by the Z-System. MAXDRV is the number of the highest drive on the system, starting with A=1. In the above example, this symbol would be given the value 13 in decimal or 000D in hex. The default value generated when MKZCM is run from CP/M is 0010 (16 decimal, all drives recognized).[2]

The symbol MAXUSR specifies the highest user number to recognize under Z-System. Although only user numbers from 0 to 15 can be logged into, CP/M-2.2 actually assigns files to user numbers up to 31 in the disk directory. Thus, the default value generated when MKZCM is run from CP/M is 001F (31 decimal), and there is seldom reason to change this value.

Printer and CRT Characteristics

The Z-System environment defines several physical characteristics of the printer and CRT devices. Before ZCPR34, the environment stored two CRT definitions and four printer definitions. The active definitions were selected by the values of the symbols CRT and PRT. Now there is only one definition for each device (hence the symbols CRT and PRT should not be changed from their default values of 0000), but the utility CPSET can be used to create any number of definitions.

The default device characteristics can be set by the following symbols:

- For the CRT, COLS for the width of the screen (default 80 decimal, 0050 hex), ROWS for the total number of lines on the screen (default 24 decimal, 0018 hex), and LINS for the number of lines to use for text display (default 22 decimal, 0016 hex)

[2]Most utilities do not know about the drive vector and use MAXDRV to determine the drives to access. Also, there are situations in which one wants to make a distinction between drives that are allowed for named-directory references (the drive vector) and the highest drive that a user can access using the drive/user (DU:) form of reference (MAXDRV).

- For the printer, PCOL for the number of print columns (default 80 decimal, 0050 hex), PROW for the total number of lines on the page (default 66 decimal, 0042 hex), PLIN for the number of lines to use for printed text exclusive of margins (default 58 decimal, 003A hex), and FORM to indicate the ability of the printer to respond to a formfeed character by advancing to the next page (0001 if so [default] or 0000 if not)

System Operational Characteristics

Some Z-System software timing loops are based on the value of the symbol SPEED. Set it to the clock speed of the CPU in your computer to the nearest megahertz. If your machine runs at 2.5 MHz, you'll just have to decide whether you like to play things down (use 0002) or to exaggerate (use 0003); the definition of this symbol allows no way to be honest! MKZCM's default value is 0004.

A number of Z-System utilities display less or no screen information if the system "quiet flag" is set. The state of this flag when the system is loaded is determined by the symbol QUIET. Set it to 0000 for normal operation or 0001 for quiet operation.

Z-System utilities and the command processor can allow or reject directory specifications of the form DU:. If the symbol DUOK is set to 0001 (the default), then directories can be specified in drive/user format. If the value is 0000, then only named directories will be accepted. The latter choice is rarely used and is not recommended.

Public Directory Specification

You can specify the initial configuration of ZRDOS public directories using the symbols PUBDRV and PUBUSR. The eight low-order bits of each word are used. For PUBDRV the least significant bit (bit 0) represents drive A and the highest (bit 7), drive H. Set the bits for any drives that you want to be public when the system is loaded. For PUBUSR the least significant bit (bit 0) represents user number 1 and the highest (bit 7), user 8. Set the bits for any user areas you want to be public when the system is loaded. The default values are 0000 for both symbols so that no directories are public initially.

5.2.4 Creating New Definitions with MKZCM

The `MKZCM` utility provides a convenient way to make changes in the memory map of the system and to create entirely new system descriptors. It allows you to change the sizes of the major system modules, including a special "user" memory buffer that sets aside memory that can be handy for "above-`BIOS`" system extensions like DateStamper or `RAM`- or hard-disk drivers.

You can enter the command

```
    MKZCM //<cr>
```

as usual to get a help screen explaining the function and syntax of `MKZCM`. From the help screen you can either exit back to the operating system or enter the program. You can also use either of the following two forms to enter the program directly:

```
    MKZCM<cr>
    MKZCM name<cr>
```

In the second form, "`name`" is the name that will be assigned to the `ZCM` and `ENV` descriptor files for the newly defined system. In the first form, you will be prompted for a name from inside the program if one is needed.

When `MKZCM` runs, it presents a menu display that covers most of the screen and a command prompt at the bottom. You can respond in a number of ways. You can enter any of the numbers (or the letter "U") that appear at the left side of the screen to select the system module whose size you want to change. In that case, you will be prompted to enter a new value for the size of the module in units of records (128 decimal or 80 hex bytes).[3]

The range of values that will be accepted and the value that will be provided when a carriage return only is entered depend on the module being defined. For many of the modules, a simple carriage return selects a value of zero. Because of a constraint that the `BIOS` start on a page boundary (an even multiple of 100 hex), `MKZCM` will sometimes automatically assign an extra record to a module. You may find it most convenient, because of this automatic adjustment, to start at the bottom of the menu and work upward.

[3]The definition of the shell stack is an exception. Simply respond to the prompts.

Although **MKZCM** will allow you to specify **CPR** and **DOS** sizes that do not comform to **CP/M** standards, we recommend that average users not make any changes to the sizes of modules 1 through 3. Advanced users may find interesting and useful applications for systems with larger (or smaller) than standard command processors or disk operating systems.

The intent in the design of **NZ-COM** is that users will define several different systems and load the one that best suits the current tasks to be performed. The next section of the manual discusses how one can change from one system to another at any time. Before we come to that subject, we would like to suggest some configurations that you might want to consider.

First let's talk about the "User's memory area". As an example, we will describe how you would use this buffer to run DateStamper.[4] The version of DateStamper that runs with a generic DOS requires 10 records of memory. It can automatically configure and load itself underneath the command processor, but this locks the command processor in place and effectively takes away an additional 2K of memory. It is much more efficient to load DateStamper above the BIOS.

Locating it above the *real* BIOS requires moving the BIOS down to make room for DateStamper. Users may not have the skill to carry this out, and it also means that the memory assigned to DateStamper can never be recovered for use by programs. With **NZ-COM** you can have one system definition that includes a buffer for DateStamper and another definition that does not.

You should run **MKZCM** now and follow through the examples. Note the addresses of the various system modules. Now define a buffer for DateStamper by pressing "U" at the command prompt from **MKZCM** and then entering

```
10<cr>
```

Note how the addresses for the system modules change automatically to accommodate the buffer. At this point you would note the starting address of the user buffer. Then you would carry out the procedure for creating a special "above-BIOS" version of DateStamper.[5] You would then load DateStamper in the usual way. You should note, however, that every time you load a new **NZ-COM** system, the **DOS** and **BIOS**

[4]This is an operating system extension for CP/M-2.2 that maintains time/date stamps indicating when files were created, last accessed, and last modified.

[5]Consult the DateStamper manual for instructions on how to do this.

are reloaded. The hooks into DateStamper are thereby lost, and you must reload DateStamper. This can be done automatically by changing systems using alias scripts (look at the sample aliases in **ALIAS.CMD**).[6]

Some common choices for the other components of the system might be as follows.

One will probably want to define systems both with and without an IOP. If one usually uses an IOP, such as the excellent **NuKey** keyboard macro IOP, then the default system would include an IOP buffer, and the definition that omits it could be called something like **NOIOP**. If one does not use an IOP most of the time, then the definition with no IOP buffer would be given the default name of **NZCOM** (**ZCM** and **ENV**), and the version with the IOP could be called **WITHIOP**.

There will be times when one is really cramped for program memory. For these occasions, one might want a system, perhaps with the name **SMALL**, that drops both the IOP and the RCP, the two largest auxilliary modules. If the memory crunch gets even more severe, a system perhaps called **MIN** might omit even the FCP and/or NDR.

As you gain experience using **NZ-COM**, you will develop a sense of the system definitions that best suit your needs. And as your needs change, you can easily go back and redefine existing systems or create still additional systems. You can even create a temporary definition on the spot and load it in less than a minute!

5.3 Loading New Systems

So far we have been discussing how new systems are defined; we have not described how systems other than the default system with the name **NZCOM** are loaded. Recall that the default system is loaded with the command

```
    NZCOM<cr>
```

This command uses the system descriptor **NZCOM.ZCM** (only the **ZCM** version). It loads the code and other modules it needs either from the library **NZCOM.LBR** or from individual files located along the path.

The command processor (CPR), the DOS, and the **NZ-COM** special BIOS are loaded using the default files **NZCPR.ZRL**, **NZDOS.ZRL**, and

[6] See p. 71 for another possibility.

NZBIO.ZRL, respectively. In addition, if the corresponding modules are included in the system definition, the **RCP** (**NZRCP.ZRL**), **FCP** (**NZFCP.ZRL**), and **IOP** (**NZIOP.ZRL**) are loaded. The latter is just a dummy **IOP** that provides the basic character input and output functions. If named directories are supported, a file called **NZCOM.NDR** will be loaded, and, if present, the terminal descriptor file **NZCOM.Z3T** will also be loaded.

Systems other than the default definition can be loaded in a great many ways. First we will describe a shorthand method that is easy to use and generally does what you want. This method uses the command line:

```
NZCOM name<cr>
```

where "**name**" is the name of a **ZCM** or **ENV** file that you created using **MKZCM**.[7] The system descriptor file of that name will be used to define the system. Modules will be loaded exactly as described above.

The shorthand command cannot be used to load systems that define smaller, but non-zero-size, modules. For example, if you try to define a system with a small **RCP** (e.g., 10 records), then when **NZCOM** tries to load **NZRCP.ZRL**, an error will occur because the **RCP** created from that file cannot fit into the buffer allocated for the **RCP**. Thus the shorthand method should be used only to load systems from which buffers have been omitted entirely or made at least as large as the defaults.

The most general form of the **NZCOM** loading command allows enormous flexibility, much more than most users will ever need. See page 63 for a full description. Here we will simply note the command form that should be used to load a system that you want to use a module other than one of the defaults. The form of the command is

```
NZCOM name module(s)<cr>
```

where "**name**" is, as before, the name of the descriptor file and where one or more system modules can be named explicitly. For example,

```
NZCOM SMALL SMALLRCP.ZRL<cr>
```

would load a system described by **SMALL.ZCM** (or **SMALL.ENV**) and would use **SMALLRCP.ZRL** instead of **NZRCP.ZRL** for the **RCP** module.

[7] Or a **ZCI** file created by **NZCOM** as described in the next section.

5.4 The System Cloning Option

There is an additional method of loading NZ-COM systems that offers the advantage of greater loading speed at the expense of requiring larger files on the disk. NZCOM has a "clone" option that allows it to go through the normal system generation procedure (as if run from the CP/M prompt) but to write the resulting binary image of the system to a file instead of loading it into memory. The result is a file whose name is the name of the ZCM or ENV file (or NZCOM if none was specified explicitly) used to define the system and whose type is ZCI (for Z-Com Image). For example, the command

```
NZCOM /C<cr>
```

creates NZCOM.ZCI with the default system image. The command

```
NZCOM SMALL SMALLRCP.ZRL /C<cr>
```

creates SMALL.ZCI with the image of the small system, including its special RCP module.

Systems defined by ZCI files are loaded in the same way as those described by ZCM and ENV descriptors by including their name on the NZCOM command line. For example,

```
NZCOM SMALL.ZCI<cr>
```

will load the small system in a single operation, without having to locate and load all the individual files that were used when SMALL.ZCI was created. This permits the system to be loaded in much less time. It also means that modules required to create the system, such as SMALLRCP.ZRL, do not have to be kept on the disk.

When the shorthand form of the NZCOM loading command is used, as in

```
NZCOM SMALL<cr>
```

the file type assumed for SMALL is first ZCI, then ZCM, and finally ENV. The basic loading command

```
NZCOM<cr>
```

however, is a special case and is equivalent to

```
NZCOM NZCOM.ZCM<cr>
```

If you want to load NZCOM.ZCI, you should use one of the following command forms:

```
NZCOM NZCOM.ZCI<cr>

NZCOM NZCOM.LBR NZCOM<cr>
```

The first form will load only a ZCI file; the second form simply forces NZCOM to use its normal hierarchy of ZCI, then ZCM, and finally ENV.

5.5 Changing Systems on the Fly

The previous discussion may have given you the impression that you are expected to load a particular NZ-COM system and then to stick with it for the duration of the session. This is not at all the case. Quite the contrary, one of the principal design features of NZ-COM is its ability to load new versions of the operating system at any time, even right in the middle of a multiple command sequence.

5.5.1 Loading Rules

When NZCOM is loaded from the CP/M command prompt (a "cold" load), all of the values specified in the descriptor file take effect and all system modules are loaded as described earlier (p. 49). On the other hand, when NZCOM is invoked from an already running NZ-COM system (a "warm" load), only things that have to be changed are actually changed.

First, all system information with the exception of the addresses and sizes of the system modules is preserved whenever possible. Shells and error handlers continue to run in the new system; the command search path, values in the user registers, system file names, printer and CRT characteristics, etc. do not change. The contents of the multiple command line buffer are carried over to the new system. This means that new configurations can be loaded as part of a complex sequence of commands, even one that includes flow control operations.[8] The system doesn't miss a beat!

Second, the contents of system modules (**RCP**, **NDR**, etc.) are not changed unless the address of the module has changed or the size of the module has decreased. Even naming a module explicitly on the **NZCOM** command line will not force it to be loaded; it will be used only if a change in the module address or size requires its loading.[9]

5.5.2 Why Change Systems

Why would you want to change systems on the fly like this? The usual reason is memory. If there were infinite memory, you would load the biggest, most powerful version of **NZ-COM** you could get and leave it there all the time. In real life, especially on a CP/M system with its address space of only 64K, trade-offs have to be made.

Most people most of the time will probably want to have a rather powerful version of Z-System, one that includes all system facilities with the possible exception of an IOP. There will be occasions, however, when an application program will be run that needs more working memory than this configuration allows. Then one will want to load a smaller system. Conversely, if the standard system does not include an IOP, then when one does want to use an IOP, one will load a bigger system.

For example, suppose we have an application program called "**BIGPROG**" that requires a lot of memory (perhaps it is a database manager). We could then enter the command

```
    NZCOM SMALL;BIGPROG ...;NZCOM<cr>
```

This command line will first install the small **NZ-COM** system with its larger **TPA**. Then it will run **BIGPROG**. Finally, after **BIGPROG** has finished, the standard **NZ-COM** system will be reloaded. Apart from the time penalty associated with changing systems, there is no disadvantage to dropping **Z-System** features that are not needed by the application program being run.[10] In the above example, by the time the prompt

[8] Provided, of course, that no flow commands have to be processed while a system with no FCP is active.

[9] You *can* force a module to be loaded by invoking the **NZCOM** program in its module loading mode as described on page 63.

[10] Few application programs make use of **Z-System** features. WordStar Release 4

has returned asking the user for his next command, the powerful verson of **Z-System** is back in place. As far as the user can tell, it was always there.

5.5.3 Automating System Changes

The process of changing systems can be automated to various degrees by taking advantage of **Z-System** capabilities. The system loading commands described in Section 5.3 can all be made into aliases or included in **ALIAS.CMD**. Then changing to a new system, such as **SMALL**, can be made using the simple command line

```
SMALL<cr>
```

The automation can be extended to situations where particular programs require particular versions of the **NZ-COM** system. Suppose, for example, that BIGPROG.COM is in directory BIGDIR: and that we rename it to BGPRG.COM. If we put the following script definition into our ALIAS.CMD file

```
BIGPROG    nzcom small;bigdir:bgprg $*;nzcom
```

then we can just enter our command as we used to:

```
BIGPROG FILENAME ...<cr>
```

The **ARUNZ** extended command processor will automatically handle the system switching for us. Fancier scripts can even test to see if the **TPA** is already large enough for a program and only change systems when necessary. There is no limit to what ingenuity can accomplish with the flexibility offered by **NZ-COM**!

There are two fine points that we should mention here. First, **NZCOM** is smart enough not to perform any loading operation when it recognizes that the requested system is no different from the existing system. Second, if **NZCOM** encounters an error when attempting to load a new system, it does not simply give up. It displays an error message on the screen reporting the nature of the problem and *then it invokes the error*

is the exception among applications from major software houses; it can use named directories. In this case you might want to create an **NZ-COM** system that includes only an **NDR** buffer.

handler. In this way, you have a chance to correct any errors before the command sequence plows ahead, possibly leading to a catastrophe (or at least a great inconvenience).

5.5.4 Automatic Returns to CP/M

There may be some exceptional circumstances under which you will need to run a program under normal CP/M. For example, you might need every bit of TPA memory that you can get your hands on. Or you might have to run a configuration program for your computer that performs direct BIOS modifications using addresses offset from the warm boot vector. Since under NZ-COM the warm boot vector points to the NZ-COM virtual BIOS, these utilities will not operate properly.

You can use the SUBMIT facility to run a batch job that exits from Z-System entirely, runs an application under plain CP/M, and then returns to Z-System! You do have to observe some precautions, however. For example, you obviously have to make sure that all command lines in the batch file that will execute while Z-System is not in effect are valid CP/M commands. This means only one command on a line and no type-3 or type-4 programs. Once the batch script has reloaded Z-System, it can resume using appropriate Z-System commands, including multiple commands on a line.

Another factor to bear in mind is that NZCPM returns you to CP/M in drive A user 0 no matter where you were when it was invoked. Since ZCPR3 (starting with version 3.3) writes its SUBMIT file to directory A0: rather than to the current directory, there is no problem with continuing operation of the batch file under CP/M. However, when you reload NZ-COM (it will be a cold load, including execution of STARTZCM), you will not automatically be back in your original directory. You will have to include an explicit command to get back.

You must also make sure that you use only versions of ZCPR34 that support the standard type of SUBMIT operation. The "LONGSUB" form of SUBMIT processing is incompatible with CP/M SUBMIT processing.

Here is a sequence of command lines that might be included in a file called CONFIG.SUB to automate the running of a utility originally called CONFIG.COM under CP/M.

```
    A0:NZCPM
    CONFIGO
```

```
NZCOM
$1:
```

The first command line removes **NZ-COM** from operation and returns the system to **CP/M** in user area **A0:**. The next two command lines run under **CP/M**. The first one runs the configuration utility, which we have renamed to CONFIG0.COM for reasons that we will explain in a moment. The second **CP/M** command reloads the standard configuration of **NZ-COM**. When that command is finished, **Z-System** is again running, but in directory **A0:**. The final line runs under **Z-System** and is designed to log us back into the directory we were in when this whole process was initiated.

The "**$1**" expression in the last command is a SUBMIT parameter. We have assumed that this SUBMIT script will be invoked by an alias named CONFIG in the ALIAS.CMD file. The alias script would have the form:

```
CONFIG    sub config $hb
```

The **ARUNZ** parameter "**$hb**" returns the default directory at the time the alias is invoked in the format "DU". This is passed as a parameter to the SUBMIT job.

Chapter 6

Technical Reference

6.1 Definition of File Types

The **NZ-COM** operating system identifies the functions of certain special files by their file types. These are listed in Table 6.1 on page 58. **ZRL** files are special relocatable files that can be adapted to any **NZ-COM** system configuration. They may be in the formats supported by either SLR Systems or Microsoft. The modules must be named as indicated in Table 6.1 by including a line in the source code using the **NAME** pseudo-op. There are no restrictions on the names used for the files themselves. At load time, **NZCOM** determines the kind of module coded by the file by examining the module name embedded in the code.

6.2 Files Supplied with NZ-COM

Many files are provided with the **NZ-COM** system. Most of them are listed in this section along with explanations of their functions. See the file **RELEASE.NOT** for information about any changes.

6.2.1 NZ-COM System Files

The files that comprise the **NZ-COM** product are listed in Table 6.2 on page 59. We want to remind you that these files are copyrighted and

FILE TYPE	CONTENTS	
LBR	library file	
ZCM	NZ-COM system descriptor	
ENV	Z-System environment and NZ-COM system descriptor	
ZCI	NZ-COM system binary image	
Z3T	Z3 terminal capability descriptor (TCAP)	
NDR	named directory register file	
ZRL	named-common Z-system ReLocatable file with a REL module name of:	
	CCPxxx	NZ-COM command processor
	DOSxxx	NZ-COM disk operating system module
	BIOxxx	NZ-COM BIOS module
	RCPxxx	Resident Command Package
	FCPxxx	Flow Command Package
	IOPxxx	Input/Output Package
	where "x" is any character	

Table 6.1: Table of file types used with the NZ-COM system.

are licensed for a single user only. It is illegal to copy or distribute these files to any other person. See the copyright notice page at the beginning of the manual for details.

The alternative RCP is smaller than the standard RCP and offers a very different set of commands, chosen with operating speed in mind. With the exception of the R command, which requires almost no code in the RCP, it includes only commands that operate entirely in memory and without reference to disks. Here the speed improvement offered by a resident command can be fully appreciated. Commands that operate on the disks or on disk files, such as SP, TYPE, and ERA, are left to transient commands.

The alternative FCP omits the IFQ command and thereby is one record smaller. Once you have learned how to use Z-System flow control, you may choose to use this smaller FCP. If so, you could rename NZFCP1.ZRL to NZFCP.ZRL (and the standard NZFCP.ZRL to, perhaps, NZFCP2.ZRL) so that it will load as the default. The same renaming can be done for the RCP described above. You can rename the library members using the NULU library utility. If you do not have NULU, you can extract all the files from the library (LGET), rename the ones you want to change,

NZCOM	.COM	NZCOM system loader
NZCOM	.LBR	library of NZCOM system modules
NZCPR	.ZRL	default command processor
NZDOS	.ZRL	default disk operating system
NZBIO	.ZRL	default virtual BIOS
NZRCP	.ZRL	default resident command package
NZRCP1	.ZRL	alternative resident command package
NZFCP	.ZRL	default flow command package
NZFCP1	.ZRL	alternative flow command package
NZCOM	.NDR	default named directory definition
MKZCM	.COM	NZ-COM system defining utility
ZCPR34	.LBR	alternative command processor modules
JETLDR	.COM	Z-System package loader

Table 6.2: List of the NZ-COM system files.

and rebuild the library (LPUT). If you are adept at patching files, you could change the names directly in the LBR file that way as well.

6.2.2 Tools and Utilities

Many files that are not a part of the proprietary NZ-COM system are included with the distribution package *as a convenience to users.*[1] Some of these files are public-domain. Others are copyrighted by their authors or by ZSIG, the Z-System Interest Group. With all of them, however, the authors have granted permission for them to be copied and distributed free of charge to other users for *non-commercial* use.

The following files support the Z-System TCAP facility:

TCSELECT	.COM	select a terminal descriptor
Z3TCAP	.TCP	database of terminal descriptors

The following files support Z-System aliases:

SALIAS	.COM	standalone alias generator
ARUNZ	.COM	alias command processor

[1]Therefore, we cannot take responsibility to support these programs.

ALIAS	.CMD	alias script file for ARUNZ

The following file is used by the FCP to process extended conditional tests:

IF	.COM	extended flow condition tester

The following utilities define or display various Z-System environment variables and system capabilities:

CPSET	.COM	displays/defines CRT/PRT characteristics
PATH	.COM	set/display command search path[2]
SHOW	.COM	display Z-System configuration information
EDITNDR	.COM	edit named directory register in memory[3]
SAVENDR	.COM	save named directory register to file

The following programs are utilities of general interest:

FF	.COM	file finder
CRUNCH	.COM	file compression tool
UNCR	.COM	file decompression tool

The following programs support library files:

LDIR	.COM	library directory program
LGET	.COM	library member extractor
LPUT	.COM	library file inserter
VLU	.COM	video-oriented library file utility
LX	.COM	library file executive

The following files are related to shells and error handlers:

ZF-REV	.COM	ZFILER shell for reverse-video terminals
ZF-DIM	.COM	ZFILER shell for dim-video terminals
ZFILER	.CMD	macro script file for ZFILER
EASE	.COM	command history shell and error handler
EASECMD	.COM	generates file with EASE command keys
VARPACK	.COM	compresses EASE history file

[2]name changed from the original SETPATH.COM
[3]name changed from the original EDITND.COM

The following files support the **Z-System** help facility:

HELP	.COM	displays menu-driven help files
xxx	.HLP	various help files

The following programs are type-4 transient programs (see p. 23) that run at the very top of the available **TPA** and leave low memory undisturbed. They are especially useful in a "minimum" system, where transient programs are used for functions otherwise performed by RCP commands. You will want to rename them to omit the leading **TY4**.

TY4SP	.COM	disk space
TY4SAVE	.COM	save memory to file
TY4REN	.COM	rename file
TY4ERA	.COM	erase file

There are several ways to tap into the rich lode of ever-expanding **Z-System** user-group files. The telephone numbers of the **Z-Node** remote access systems are listed in:

ZNODES .LST

A number of individuals have volunteered to help others install and use **Z-System**. Their names, addresses, and phone numbers are listed in the file

ZHELPERS .LST

6.3 NZCOM Command Lines

This section describes the various forms of command line that can be used with the **NZCOM.COM** program.

6.3.1 Help Screens

A built-in help screen designed to remind you of the syntax required for **NZCOM** commands is displayed by typing either of the following commands:

```
NZCOM //<cr>
NZCOM ?<cr>
```

This screen will also indicate how the current version has been configured by patching the file (see p. 69).

6.3.2 Loading NZ-COM Systems

The general form of the **NZCOM** command line is

```
NZCOM [library] [descriptor] [filelist]
   [/options] [;commands]<cr>
```

All elements on the command line are optional. If multiple items are present, they may be separated by spaces (as shown), commas, or combinations of the two. Different separators can be used on different parts of the command line.

General Rules

Before describing the individual command-line items in detail, we want to define two general rules. First, a prefix indicated as "**dir:**" in a syntax expression represents any appropriate kind of directory specification. When **NZCOM** is invoked from **CP/M**, then the directory prefix may have any of the following drive/user formats: colon only for current directory, D:, U:, or DU:. If Z-System is already running, then named directory references may also be used.

Second, files specified on the **NZCOM** command line are looked for according to the following procedure. If the file has a specific directory prefix, then only that directory is searched. If no explicit directory prefix is present, then the currently selected library (more on that in a moment) is searched first. If the file is not found in the library, then the "path" is scanned. If Z-System is already running, then "path" means the current Z-System path including the current directory. If **CP/M** is running, then "path" means an internal path configured into the **NZCOM.COM** program (see p. 69). You can see what this path is by invoking the **NZCOM** help screen.

The Library Term

The first item on the command line is an optional library file specification. It has the form

```
[dir:]lbrname.LBR
```

The file type **LBR** is required. If no library is named explicitly, then a term of "**NZCOM.LBR**" is assumed for this command-line item. A directory prefix is optional.

Libraries can also be specified, as we shall soon see, as items in the file list. In all cases, **NZCOM** looks for library files just as it does for any other file, except that, naturally, it does not look for it as a member of any other library. The most recently named (or implied) library is the one used in the search hierarchy for other files. A library named at the beginning of the command tail applies to the search for any descriptor file.

The Descriptor Term

The second item is an optional **NZ-COM** system descriptor file. It, too, can optionally include a directory prefix. It is identified by having either no file type at all or one of the three system descriptor file types **ZCI**, **ZCM**, or **ENV**. If no file type is named explicitly, then the file search procedure is applied sequentially to each of the three file types in the order just listed. If a system descriptor does appear on the command line, then **NZCOM** enters *system-building mode.* Otherwise, it enters *module-loading mode.*

If no file list (see below) is present, then it is generally as if a term of "**NZCOM**" were present as the system descriptor. An exception occurs when there are no files at all named in the command tail. Then the file-specification part of the command line is taken to be equivalent to

```
NZCOM NZCOM.LBR NZCOM.ZCM
```

where the **ZCM** file type is used instead of **ZCI** as required by the general rule given above.

The File List Term

The third item is an optional list of one or more module or library files having the form

```
filename [filename [...]]
```

Each file name has the form

```
[dir:]name.typ
```

where the allowed file types are LBR, ZRL, REL, Z3T, NDR, FCP, RCP, or IOP.

When a library file is named, it becomes the currently active library and is used in the search for module files named subsequently.

The ZRL file is a special form of relocatable image that can be configured for use in a **Z-System** when the file is to be used (as opposed to when it was created). It is the only kind of file that can be used to load command processor, disk operating system, or virtual BIOS modules. The ZRL file is a very convenient way to supply other code modules, such as FCP and RCP modules, since a single ZRL file can be used in any **Z-System** that has memory allocated for it. We encourage developers and users to give ZRL-type files the file extension ZRL. However, files of the ZRL type with an extension of REL can be loaded.

NZCOM can load several forms of non-relocatable files as well. Z3T terminal capability descriptor files and NDR named directory register files never pose a problem, since these modules contain no code; they contain only data. **NZCOM** can also load absolute binary images of FCP, RCP, and IOP code modules. This is inherently a risky procedure in a dynamic system like **NZ-COM**, where the addresses of system components can change. For that reason, we strongly encourage the use of ZRL files. **NZCOM** does make some effort to check these absolute modules for compatibility with the **NZ-COM** system running or being built, but there is no test that can guarantee compatibility.

NZCOM operates on the list of files in two different ways depending on which mode it is in. If it is in module-loading mode, then any module files named in the list are loaded directly into the currently running operating system. If it is in system-building mode, then it builds the image of a new operating system.

In system-building mode, if an **NZ-COM** system is already running, modules named in the list are loaded into the image only if the newly defined **NZ-COM** system differs from the currently running **NZ-COM** system in a way that requires loading of the new module, namely, if the address of the buffer to which the module would be loaded has changed or if its size has decreased. If any modules are needed that were not specified in the list, then **NZCOM** uses its standard default module names, looking for the files as if they were specified explicitly at the end of the file list. These modules are as follows:

- `NZCPR.ZRL` for the command processor
- `NZDOS.ZRL` for the disk operating system
- `NZBIO.ZRL` for the **NZ-COM** virtual `BIOS`
- `NZFCP.ZRL` for the flow command package
- `NZRCP.ZRL` for the resident command package
- `NZIOP.ZRL` for the input/output package
- `NZCOM.NDR` for the named directory register
- `NZCOM.Z3T` for the terminal capabilities descriptor

If the "`C`" option flag is specified (see below), then **NZCOM** pays no attention to whether `Z-System` is currently running. It acts as though `CP/M` is running and loads all module files into the image.

The Options Term

The next item in the **NZCOM** command line is an optional slash character followed by one or more flag characters.

Flags of "`Q`" or "`V`" select "quiet" or "verbose" mode. In verbose mode, **NZCOM** displays information about the modules that are being loaded, including the name of the file used and, where appropriate, the address to which the result will be loaded. In quiet mode this display is suppressed, and only a series of dots is displayed. This lets you know that something is happening and that progress is being made. The **NZCOM** help screen indicates the default mode for this flag. Quiet mode is the default in the distributed version of **NZCOM.COM** but can be changed by the user (see p. 69).

Another option flag is "C" (for "clone"). When this flag is specified, NZCOM acts as though it was invoked from the CP/M command line and builds a complete system. However, it does not load this system. Instead, it writes it out to a file whose name is that of the NZ-COM descriptor file named on the command line (or the default NZCOM). Naturally, this option can be used only in system-building mode.

A third option is one of the pair "Z" and "R". These determine whether NZCOM will recognize a file extension of ZRL or REL for ZRL-type files that it loads as *default* files (see the list of default files above) when it needs a module that you have not specified explicitly. We recommend that this option not be used and that you rename the files to an extension of ZRL so that it is clear that they are files of the ZRL type.

The Command Line Term

The final optional item on the NZCOM command line is a semicolon followed by Z-System-type multiple command line expression. Since a semicolon terminates a command line under Z-System, this option has no meaning if Z-System is already running.[4] It is intended as a way to allow the user to include an initial multiple command line when NZCOM is invoked from CP/M.

6.3.3 Removing NZ-COM

When NZCOM is invoked from CP/M, it looks to see if a file with the name NZCPM.COM already exists in the root directory of the internal path configured into NZCOM.COM (see p. 69). If not, it captures an image of the CP/M system and saves it in a file together with a loader that can restore it. This file is called NZCPM.COM. The NZ-COM operating system can be removed from the system and the CP/M operating system restored at any time by issuing the command[5]

```
NZCPM<cr>
```

[4] Whenever NZCOM loads a new version of Z-System, any pending commands in the command line buffer of the running system are put into the command line buffer of the new system. As a result, the option described here appears to work just the same from a running Z-System.

[5] This assumes that NZCPM is in a directory along the search path. If not, you have to include an explicit directory prefix with the command.

6.3.4 Examples and Tips

We will now present a few examples of NZCOM command lines to help make clear how the syntax works. As we discuss these examples, we will point out some tricks that can simplify or speed up the operation of NZCOM.

Loading a Special NZ-COM Configuration

The following command line could be used to load a special configuration defined by SMALL.ENV. It is assumed that the definition and the modules SMALLFCP.ZRL and SMALLRCP.ZRL that it uses are included in the library SPECIAL.LBR which resides in directory A15:. Other than those specific modules, all other modules are the default files kept as usual in NZCOM.LBR in A0:. Here is the command line that loads this system:

```
NZCOM A15:SPECIAL.LBR SMALL.ENV SMALLFCP.ZRL
   SMALLRCP.ZRL A0:NZCOM.LBR /V<cr>
```

We have used explicit drive specifications for both libraries so that NZCOM will not have to waste any time searching for them along the path.

By including an explicit file type in the term SMALL.ENV we gain two advantages. First we make sure that files with the names SMALL.ZCI or SMALL.ZCM are not loaded (against our intentions here). Second, in case the file is accidentally not in the library, we prevent NZCOM from performing a rather lengthy search for files with all three descriptor file types over the entire search route before the error is detected and reported.

The inclusion of the term A0:NZCOM.LBR *at the end of the file list is very important.* Without it, the current library at the end of the list would have been A15:SMALL.LBR. The default files required for the remaining modules would not have been found, since NZCOM would not have looked automatically in NZCOM.LBR.

The option "/V" at the end makes sure that NZCOM displays a detailed load map showing how the system was generated.

If one were going to load the system configuration defined by this command more than just one or two times, it would make sense to add this command line to the **ALIAS.CMD** file under an alias name like **SMALL**. This would save a lot of typing and prevent a lot of errors, such as forgetting to include the **NZCOM.LBR** term or typing one of the file names incorrectly.

Loading Special Modules

The following command is used to load a new group of command modules. It assumes that we develop our **RCP**s in a directory with the name **RCP** and that we keep the ones we have completed in a library called **RCPS.LBR**. It assumes that we only have a few versions of **FCP** and that we prefer to leave them as individual files in our main directory **A0:**.

```
NZCOM RCP:RCPS.LBR RCP-D.ZRL A0:FCP-1.ZRL<cr>
```

By including an explicit **A0:** in front of the **FCP-1.ZRL** term we gain speed. **NZCOM** would normally look for the file first in **RCPS.LBR** and only then search the path. By including the correct directory prefix, we get **NZCOM** to locate the file immediately.

Loading a New Command Processor

Users of **Z-System** are accustomed to the idea of loading new **RCP** and **FCP** modules. It may come as something of a surprise that **NZCOM** can just as easily load a new command processor. The **NZ-COM** distribution package includes a library called **ZCPR34.LBR** that contains several alternative command processor configurations. One of them supports a nonstandard form of **SUBMIT** processing that allows arbitrarily long **SUBMIT** files.[6] To use the special command processor file named **Z34LONG.ZRL** we would use the command line

```
NZCOM A0:ZCPR34.LBR Z34LONG.ZRL<cr>
```

It would probably be useful in the long run to define an alias in **ALIAS.CMD** under a name like **LSUBCPR** that would generate this command line for us.

[6] Ordinary **CP/M SUBMIT** files may not contain more than 128 command lines, the number that can be included in a single file "extent".

6.4 Patching NZCOM.COM

There are several options in NZCOM.COM that are controlled by configuration bytes in the code. You can install changes in several ways. The most convenient method uses a patching utility like ZPATCH. You can also use facilities already provided by Z-System. You can load NZCOM into memory using the GET command, make the changes using the POKE command, and then save the modified image using the SAVE utility. Before you do this, however, you have to know how large the file NZCOM.COM is in records. The SDZ directory program with the "/C" option will give you this information.

We will now describe the current configuration areas in detail. This information is correct for the initial release version of NZ-COM, and it is quite possible that future versions will have additional or different configuration options. You should always consult the file RELEASE.NOT, if one is present, for more recent information.

6.4.1 The Internal Search Path

The "internal" search path, which is used when NZCOM operates under CP/M, which becomes the initial Z-System path, and which determines where the CCP image file NZCOM.CCP and the NZ-COM system unloader NZCPM.COM are put, is stored at address 0680.[7] Each element in the path is expressed by a pair of bytes. The first byte of the pair is the drive, with a value of 1 for drive A. The second byte of the pair is the user number and must have a value in the range 00 to 1F (31 decimal).

There are two special symbols that can be used in a path element expression. A dollar sign (24 hex) for a drive takes on the value of the currently logged drive at the time the path expression is used. Similarly, a dollar sign in the user position represents the currently logged user area.

There can be at most five elements in the path. The path is terminated by a single null byte (value 00). The end of the entire path sequence in the NZCOM code is marked by a byte of FF. The path in the distributed version of NZCOM.COM contains the sequence A0 $$ A0. The remaining two possible path elements are filled with nulls.

You might wonder why directory A0 is included twice in the path. The

[7] All addresses are given in hexadecimal form and assume that the program begins at an address of 0100.

final drive/user pair in the path is called the root directory.[8] Many **Z-System** operations automatically refer to this directory. For example, the **ZCPR34** command processor is usually configured to look in the root directory for the extended command processor, and **ARUNZ** and **ZFILER** look there for their **CMD** files.[9] **NZCOM** uses the root of this internal path as the place to keep the files **NZCOM.CCP** and **NZCPM.COM**.

On the other hand, the directory set up as the root often contains the most frequently used programs, and one generally wants the command search to look there early if not first. Thus **NZCOM** is set up to look in **A0** first *and* to use it as the root. Path expressions involving dollar signs can also give rise to duplicated directories in the path. In the standard configuration, the path will have *three* entries of **A0** when the user is logged into **A0**. If the first search of a directory does not turn up a sought-after file, looking a second or third time surely will not help. Both **NZCOM** and the **ZCPR34** command processor are smart enough to eliminate duplicate searches by using what is called a "minpath". The minpath is the search path after all duplicates have been removed.

6.4.2 The Default Options

NZCOM can be set up to assume either **Q** or **V** and either **Z** or **R** as the default option for those two pairs of options. The verbose option flag VOPT is stored at address 068D. A value of 00 selects quiet mode as the default; a value of **FF** selects verbose mode.

The default file extension used for the **ZRL**-type files by **NZCOM** is set by the **ZOPT** flag at address 068E. A value of 00 selects the **R** option; a value of **FF** selects the **Z** option.

6.4.3 Startup Command Line

The startup command line run whenever the **NZ-COM** system is loaded from **CP/M** is stored at address 0690. It consists of the characters that comprise the command line, followed by a null (00 byte) to terminate

[8]Don't confuse this root directory with a directory that happens to have the name **ROOT**. The latter is simply a name and confers no special significance on the directory.

[9]The versions of **ARUNZ** and **ZFILER** supplied with **NZ-COM** have been configured to look for their **CMD** files in the root directory. Those programs can, if desired, be configured to use a different directory.

the command string, and then a byte of FF to terminate the entire string.

This command line is normally set up to contain the string **STARTZCM**. This program will then be an alias that can contain many other commands. This approach makes it easier to change the startup commands than if a complex set of commands were all hard coded into **NZCOM.COM**. There is just enough room in **NZCOM.COM** — without overwriting the copyright notice — for an 8-character command name. Some amount of the copyright notice can be overwritten by the command line if you simply insist on including a longer command line (or take perverse pleasure in defacing copyright notices).

Configuration Program Patch

NZCOM.COM has a special configuration program patch area into which advanced users can load code to perform special operations in addition to those normally carried out by **NZCOM**. This patch area begins at address 0280H and extends for 1K bytes through address 067FH. Users will undoubtedly come up with many creative applications for this facility.

We envision using it to provide special patching when "above-BIOS" system extensions, such as DateStamper and/or BYE, are running in an **NZ-COM** system. With those programs it is necessary to extract information about how they are hooked into the currently running system and then to reestablish those hooks after a new system has been loaded (but before it starts to run). We will not try to explain how to do these things here; we will try to give you enough information about this patch area that you can make use of it if you have the necessary programming skills.

NZCOM issues CALL instructions to the code at address 0280H at several points in its operation, using the Z80 registers to pass information to the configuration routine. In particular, the value passed in the **A** register indicates from which the point in the sequence of tasks performed by **NZCOM** the call originated. The following calls are defined at present:

A=0	**NZCOM** has just started to run and has determined that an **NZ-COM** system *is not* currently running. No files have been loaded at this point.

A=1	**NZCOM** has just started to run and has determined that an **NZ-COM** system *is* currently running. No files have been loaded at this point.
A=2	All the designated or required default modules have been loaded into a working buffer and are ready for loading to their run-time addresses. No modifications have been made yet to the running operating system.
A=3	The modules have been copied as needed from the working buffer to their run-time locations, and **NZCOM** is ready to initiate a cold boot of the new system.
A=0FFH	**NZCOM** has determined that no new system is to be loaded. This call will occur if the new system specified does not differ from the currently running system in any way that requires anything to be loaded. It will also occur if an error is detected that requires aborting the operation of **NZCOM**.

The first two calls are provided in case some code in the configuration patch needs to be initialized before other calls occur or when other operations of the configuration routine need to know what kind of system is currently running.

Additional information is provided in other Z80 registers. The **HL** register pair points to the working buffer **WRTBUF** where **NZCOM** is composing the new system. The environment descriptor for the new system is in the block of code from 0100H to 017FH. The information contained there together with the address of **WRTBUF** can be used to calculate where each module is located in the working buffer.

The **DE** register pair points to a string of bytes with information about the load status. At present the string contains two bytes. The first is called **MODLST** (module list). Each of its eight bits is used to indicate whether changes in the system configuration require the loading of a corresponding module. The function of each bit is defined in Table 6.3. The second byte is called **SEGLST** (segment list). Each bit indicates whether a change in one of the system segments listed in Table 6.4 requires some action by the **NZCOM** loader.

Except when a call is made with a value of 3 in the **A** register, the old operating system is still in place and functional. The configuration code can, therefore, make use of operating system calls to perform its

0	CCP
1	DOS
2	BIOS
3	IOP
4	RCP
5	FCP
6	NDR
7	Z3T

Table 6.3: System segment corresponding to each bit in MODLST byte.

0	shell stack
1	message buffer
2	external FCB
3	command search path
4	wheel byte
5	command line buffer address
6	reserved
7	reserved

Table 6.4: System segment corresponding to each bit in SEGLST byte.

tasks. If it determines, for example, that the NZCOM operation should be aborted, it can do so by jumping to address 0000H to initiate a warm boot.

One should be very careful with any code that is performed on a call to the configuration routine with A=3. At this point the new system has been loaded into its run-time location, but *it has not been initialized and must not be used in any way.*[10]

[10] If you are clever, you can figure out the location of the real BIOS (as opposed to the NZ-COM virtual BIOS) and can make appropriate calls to it. When NZCOM is running under an existing NZ-COM system, the word at address 0101H is the address of the CONST routine of the real BIOS.

6.5 The JetLDR Program

Included with the NZ-COM package is a special program called JetLDR. This program is an extremely powerful general-purpose module loading program. All of the usual module loading functions required for the NZ-COM system can be performed by NZCOM.COM. However, the JetLDR program is extensible using special configuration or CFG files. These are modules which JetLDR loads into itself and which control the way it loads other modules. This facility can give JetLDR the ability to load special modules, such as resident system extensions (RSXs).

JetLDR has a built-in help screen that can be invoked using the standard command

```
JETLDR //<cr>
```

6.6 The MKZCM Command Line

Fortunately, there is not very much that has to be said about the MKZCM command syntax because it is so simple. The command line format is

```
MKZCM [name]<cr>
```

The optional token "name" — to be used as the name for the ZCM and ENV descriptor files created by MKZCM — can be included on the command line if you know in advance what you want to call the new system descriptor. However, there is little advantage to using this option. If you do not give the name on the command line, then MKZCM will prompt you for it when you tell it to save the system description. If you decide to abort the system definition process and not save the information, then you have saved yourself some typing! More importantly, should you, during the definition process, change your mind about the name you want to use, your options are still open. If you included a name on the command line, you are stuck with it!

6.7 Theory of Operation

NZ-COM is a special form of Resident System Extension (RSX) that is loaded into high memory just below the CP/M BIOS. It includes the following modules:

- command processor
- disk operating system
- warm-boot intercept
- **Z-System** resident segments:
  - external environment description
  - terminal capabilities buffer
  - message buffer
  - path
  - wheel byte
  - external file control block
  - multiple command line
  - shell stack
  - external command processor stack
- optional **Z-System** segment buffers
  - named-directory register
  - resident command package
  - flow command package
  - input/output command package

The warm boot intercept code initializes the disk system in the usual way but reloads the command processor not from the system tracks but from a file called **NZCOM.CCP**. When the system is removed by **NZCPM.COM**, the original DOS module and the original warm boot vector are restored. The host CP/M system then takes over and reloads its standard command processor from the system tracks.

When a new **NZ-COM** system is loaded while another system is already running, **NZCOM** saves the state of the current **Z-System** environment, determines what changes are required (e.g., a different-sized buffer), relocates and installs new packages, and restores the unchanged components of the environment.

Chapter 7

Bibliography

In an evolving, improving environment the documentation always lags behind practice. Here we recommend several sources of additional information about **Z-System**.

7.1 The Z-Nodes

The most current information about **Z-System** tools and standards and the latest versions of the freely distributed programs are to be found on the **Z-System** remote access systems (**RASs**), called **Z-Nodes**. These "bulletin boards" are also good places to find friendly help from fellow users. The crunched file **ZNODES45.LZT** contains a list of the 73 **Z-Nodes** operating as of January, 1988.

Of these many **Z-Nodes** we want to call special attention to four (all of which are accessible using Telenet's PC-Pursuit service). **Z-Node #1** in San Jose, California, is the granddaddy of them all. It is known as Z-Node-Central and is operated by sysop Ron Bardarson. Almost all active **Z-System** programmers call in there on a fairly regular basis. It is also the one closest to and most frequently used by **NZ-COM**-author Joe Wright (in fact, soon it will be relocated to Joe Wright's house!). Its current phone number is (408)-432-0821, but this might have to change when the system is moved.

Z-Node #2 in Los Angeles is the nearest node to manual co-author Bridger Mitchell and the one on which he can be reached most quickly and directly. The phone number there is (213)-670-9465.

Manual-co-author Jay Sage, also one of the architects of **NZ-COM** and author of **ZCPR34**), is the sysop of **Z-Node #3** in the Boston area. It's phone number is (617)-965-7259. Unlike most remote access systems, it is an open system, with no individual user registration or passwords and only public messages between users. There is, however, a general system password, "DDT", designed to allow access only to users of **Z-System**-compatible computers.

Finally, we want to mention Chicago's Lillipute Z-Node, a misnomer if ever there was one. This node has two computers, each with a very large hard disk, and two phone lines. It is probably the largest and most active of the **Z-Nodes**. The sysop, Richard Jacobson, is very aggressive in making sure that he always has the very latest files available. It is a subscription system, and well worth the price. The phone numbers in area code 312 are 649-1730 and 664-1730.

7.2 *The Computer Journal*

An excellent ongoing source of **Z-System** material is *The Computer Journal* (TCJ), perhaps the last of the major hobbiest computer magazines with significant coverage of 8-bit systems. It has regular columns and special articles by **Z-System** experts, including both Jay Sage and Bridger Mitchell. A subscription is highly recommended! Contact the publisher at P.O. Box 1697, Kalispell, MT 59903.

Included with the **NZ-COM** package are disk files containing a number of Jay Sage's columns from TCJ. The files are all crunched to save space; see page 36 for a discussion of how to handle crunched files. Here is a list of those files with an indication of the information in each one that may be of particular interest:

TCJ26.MZG	optimizing a floppy-disk-based system
TCJ27.MZG	aliases and shells
TCJ28.MZG	recursive aliases
TCJ29.MZG	the ZCPR33 command processor
TCJ30.MZG	SALIAS and VLU
TCJ31.MZG	ARUNZ documentation
TCJ32.MZG	NZ-COM/Z3PLUS/ZCPR34 information

7.3 Other Published Information

For other printed information about **Z-System**, we recommend the following books.

Read first:

> *The Z-System User's Guide* (Bruce Morgen, Richard Jacobson). This is an introduction to the **Z-System** that tries to be comprehensible to the less technical user of **Z-System**.

Then read:

> *The ZCPR 3.3 User's Guide* (Jay Sage). This is the manual that accompanied the **ZCPR** version 3.3 command processor. It includes many examples of how the features of **Z-System** can be used to advantage. Extended command processing and security features, in particular, are covered. Almost all of this information applies to **ZCPR** version 3.4.

An older reference, with information that is no longer always current, is:

> *ZCPR3: The Manual* (Richard Conn). This was the bible for **ZCPR3**, but much of the material is now out of date. The treatments of the **Z-System HELP** facility, the menu shells, and **TCAPs** (terminal capability descriptors, including the utilities **TCSELECT**, **TCMAKE**, and **TCCHECK**) are still very useful.

For the technically inclined who want to write their own **Z-System** programs, the following book, along with the relocatable subroutine libraries code, will be extremely useful:

> *ZCPR3: The Libraries* (Richard Conn). This book provides complete documentation on the hundreds of pre-written (and debugged) subroutines that make writing **Z-System** programs in assembly language almost as easy as writing in a high-level language.