The Operating System Handbook

or, Fake Your Way Through Minis and Mainframes

by Bob DuCharme

MVS

Table of Contents

Chapter 22 MVS: An Introduction.....
22.1 Batch Jobs.....1
22.2 Interacting with MVS.....3
22.2.1 TSO.....3
22.2.2 ISPF.....3
22.2.3 CICS.....4
22.2.4 Other MVS Components.....4
22.3 History.....5
Chapter 23 Getting Started with MVS.....
23.1 Starting Up.....6
23.1.1 VTAM.....6
23.1.2 Logging On.....6
23.1.2.1 Reconnecting.....8
23.1.3 Entering Commands.....9
23.1.3.1 Aborting Screen Output.....10
23.1.3.2 Command Parameters.....11
23.1.3.3 Long Commands.....11
23.1.3.4 Case Sensitivity.....12
23.1.3.5 Command-Line Options.....12
23.1.4 Finishing Your MVS Session.....13
23.2 File Names.....13
23.2.1 Sequential and Partitioned Data Sets.....13
23.2.2 Line Numbers and Data Sets.....14
23.2.3 Naming Data Sets.....14
23.2.3.1 The Members of a Partitioned Data Set.....15
23.2.4 Wildcards.....16
23.3 How Files Are Organized.....16
23.4 Available On-line Help.....17
Chapter 24 Using Files in MVS.....
24.1 The Seven Most Important Commands.....20
24.1.1 Common Error Messages.....20
24.1.2 Listing Data Set Names.....22
24.1.2.1 Listing a Catalog's Data Sets.....22
24.1.2.2 Listing a Partitioned Data Set's Members.....24
24.1.3 Looking at Data Sets.....25
24.1.4 Copying Data Sets.....26
24.1.4.1 Copying and Partitioned Data Sets.....26
24.1.5 Renaming Data Sets.....27
24.1.6 Deleting Data Sets.....28
24.1.7 Allocating Data Sets.....29
24.1.7.1 Allocating Existing Data Sets.....29
24.1.7.2 Allocating New Data Sets.....30

24.1.7.3 Unallocating Data Sets....34
24.1.7.4 Finding Out a Data Set's Allocation Status....34
24.1.8 Adding a Data Set to a Catalog....36
Chapter 25 The MVS ISPF Text Editor....
25.1 The ISPF Text Editor....38
25.2 Entering the ISPF Editor....38
25.2.1 Customizing Your Editor's Environment....42
25.3 Line Commands....43
25.3.1 Adding New Lines....44
25.3.2 Moving Your Cursor Around....47
25.4 Inserting, Deleting, and Typing over Words and Characters....47
25.4.1 Duplicating Lines....48
25.4.2 Deleting Lines....49
25.4.3 Copying Lines....51
25.4.4 Moving Lines....52
25.5 Searching for Text....52
25.6 Saving Your Changes....55
25.7 Quitting the ISPF Editor....55
25.7.1 On-line Help in the Editor....55
25.8 TSO's EDIT Text Editor....56
25.8.1 Starting the Editor....57
25.8.2 Creating a New Data Set....57
25.8.3 Line Numbering and the EDIT Editor....58
25.8.4 Input Mode and Edit Mode....58
25.8.5 Displaying the Data Set's Current Contents....59
25.8.6 The Current Line....60
25.8.7 Adding New Lines....60
25.8.8 Editing Existing Lines....62
25.8.9 Deleting Lines....62
25.8.10 Copying Lines....63
25.8.11 Duplicating Lines....64
25.8.12 Moving Lines....64
25.8.13 Searching for Text....64
25.8.14 Saving Your Changes....65
25.8.15 Quitting the TSO Editor....65
25.8.16 On-line Help and the TSO Editor....65
Chapter 26 Using an MVS System....
26.1 Printing Data Sets....67
26.2 Command Files....67
26.2.1 The Automatic Logon Command File....69
26.3 Communicating with Other Users....69
26.3.1 Sending Files....70
26.3.2 Receiving Mail and Data Sets 72
26.4 ISPF....74
26.4.1 Allocating Data Sets....75

26.4.2 Copying Data Sets 78
26.4.3 Renaming Data Sets.....80
26.4.4 Deleting Data Sets 81
26.4.5 Displaying A Data Set's Contents 82
26.4.6 Printing a Data Set.....83
26.5 A Sample MVS Session 83

This Section, the Rest of the Book

This is one part of the book "Operating Systems Handbook (or, Fake Your Way Through Minis and Mainframes)," which was originally published by McGraw-Hill as a $49.60 hardcover. Once they reverted the rights to me after it went out of print, I converted the original XyWrite files to DocBook XML and then used Norm Walsh's stylesheets (see www.nwalsh.com) and the Apache FOP program (see xml.apache.org) to convert that to Acrobat files. The six parts of the book are all available at http://www.snee.com/bob/opsys.html:

- Part 1: Introduction. Note that this part includes a new section explaining why I didn't update the book. I strongly suggest that, no matter how much or how little of the book you can use, you glance through the whole Introduction as well. The "Comments and Suggestions" part is now obsolete; my home page is now http:/www.snee.com/bob and my e-mail address is bob@snee.com.
- Part 2: UNIX. Everything described here should apply to Linux and its relatives.
- Part 3: OpenVMS. Basically, VMS. DEC was calling it "OpenVMS" at the time.
- Part 4: OS/400. The operating system for IBM's AS/400.
- Part 5: VM/CMS. An IBM mainframe operating system.
- Part 6: MVS. Another IBM mainframe operating system.

See www.snee.com/bob for information on books I've written since. In reverse chronological order, they are:

- *XSLT Quickly* is a tutorial and users guide to XSLT designed to get you writing stylesheets as quickly as possible.
- *XML: The Annotated Specification* is a copy of the official W3C XML specification with examples, terminology, and explanations of any SGML and computer science concepts necessary for a complete understanding of the XML spec.
- *SGML CD* is a users guide to free SGML software, most of which can be used with XML as well. The chapter on Emacs and PSGML, which requires no previous knowledge of Emacs, is available on the web site in English, Russian, and Polish.

One more thing: either I couldn't figure out the XSL `keep-together` property or version 0.18.1 of FOP doesn't implement it yet. Either way I apologize that some screen shots get split across page breaks.

Entire book copyright 2001 Bob DuCharme all rights reserved

Chapter 22 MVS: An Introduction

MVS is the primary operating system on the IBM 370 series of mainframes. (You may hear people use various initials and four-digit numbers when referring to IBM mainframes, such as 3033, 3090 or ES9000, but they are all considered hardware models of the 370 series.) MVS is the eighteen-wheeler of operating systems. People don't use it for flash and speed; they use it to bear large, heavy loads steadily and dependably.

When we talk about the tremendous processing power of a mainframe running MVS, we're talking about a power different from that of supercomputers. Supercomputers do complicated calculations at very high speeds. Designing them for the best possible calculation speed often means sacrificing I/O (input/output) speed; the scientists who use them are more likely to give them a complex math problem and say "grind away at this equation all night" than they are to say "read in these 300,000 records of data, do 8 calculations on each, and then output 300,000 separate reports."

Reading and writing a tremendous amount of data and doing relatively simple calculations with it (for example, calculating interest payments, as opposed to calculating a boat hull's optimum shape) is the province of mainframes running MVS. An insurance company keeping track of its accounts, a chain of stores keeping track of its inventory, or any large company keeping track of its employees and payroll would use MVS. Because it's a multi-user operating system, MVS lets many different users use the same programs and data at once.

Personal computer users like to make fun of big computers running MVS, calling them "dinosaurs." While the interface may seem primitive, MVS has had many features since its introduction in 1974 that people are only now trying to shoehorn into the operating systems that control personal computer networks. MVS includes built-in recovery routines for dealing with faulty hardware like tape drives or even (in a multi-processor environment) faulty processors. A system running MVS can support thousands of users at once. The security of one user's data against tampering by others is an integral part of the system, designed into it from the ground up. (How often do you hear of a virus or a worm breaking into an MVS system?)

The primitive interface isn't the only thing that give people the wrong idea about MVS. A given MVS installation is highly customizable, and so is the way that each user uses it. Many different parameters can be set when doing virtually anything, and MVS doesn't always have the default settings that we take for granted on other systems. The most efficient settings are left to individual system administrators to figure out. Since many settings and details are site-specific, a new user on a particular system—no matter how much MVS experience he or she brings to that system—can't be expected to know the best way to approach that system. Don't be embarrassed to ask questions when faced with an unfamiliar MVS installation.

22.1 Batch Jobs

The name "MVS," which stands for "Multiple Virtual Storage," comes from the technique it uses

 copyright 2001 Bob DuCharme

to manage memory. It lets any user work with huge amounts of memory at once, making MVS ideal for batch processing.

Running a job in batch mode is the opposite of running it interactively. Instead of starting a program, typing in some input, waiting for the response, typing some more, and continuing this cycle, you specify what needs to be done at the outset. You tell the system the program to run, the data to use, and what to do with the output. Preparing a company's payroll or feeding in thousands of inventory transactions are typical of jobs that should be run as batch jobs. (A batch job is also known as a "background" job—you can instruct the computer to start running it and then do something else interactively while your batch job runs in the background.)

In the early days of computers, no one else could use the computer once it began a batch job. Modern multi-tasking computers let people run other programs while a batch job runs. This makes MVS ideal for doing huge jobs involving lots of I/O.

Two acronyms that come up when people discuss batch jobs are JES2 and JES3. These are two versions of the Job Entry Subsystem, the part of MVS that deals with the scheduling of batch jobs. You need not worry about JES, but you may hear people refer to it.

See the sidebar "JCL: Job Control Language" for an explanation of how we give the system instructions for running a batch job.

JCL: Job Control Language

JCL is a language for telling a mainframe system how to deal with a batch job. You use it to indicate the job's size, priority, output destination or destinations, and files, printers, processor time, and disk space to use.

Many mainframe users use JCL regularly without ever knowing anything about the language. How can they do this? JCL statements are often inserted at the beginning and end of a file holding the program or data to submit. While doing data entry on a CMS system years ago (although it comes up more often on MVS systems, the batch capability of VM/CMS means that JCL is used there as well) I knew of a file of JCL code that I had to add to the beginning of my file and another to add to the end of my file before I submitted that file as a job to run. I didn't know what any of the JCL code did; I only knew that I had to add the JCL files to my data file before I typed the command that told the computer to process the data file. This sort of arrangement is not uncommon. Often, a more

hardcore programmer writes a couple of chunks of reusable JCL for less sophisticated end users who know enough about the text editor to insert these chunks into the appropriate places in their files.

JCL code looks strange and intimidating because it uses so many abbreviations. Since part of its job is to specify the treatment of certain files, you will discover that several MVS TSO commands have JCL equivalents, especially ALLOCATE.

Once you have a good handle on the different parts of a mainframe system, you can learn how they are represented in JCL and then start writing (or at least modifying) JCL yourself. It's valuable job skill.

22.2 Interacting with MVS

MVS offers several ways to interact with it: TSO, ISPF, and CICS.

22.2.1 TSO

TSO stands for Time Sharing Option. It's the part of MVS that lets you use the system interactively; you type commands at a command line and see the response on your screen. In addition to the commands that all operating systems have to copy, rename, and delete files, TSO also has commands to submit batch jobs and to check on their progress. Most of the next couple chapters show you various TSO commands for dealing with MVS basics.

When IBM first introduced TSO in 1969, interactive computing was a hot new feature that required a huge portion of the computer's memory. This is where the "O" in its name comes from: when all jobs were batch jobs, IBM made this interactive component of MVS optional. Today, it is part of all MVS systems.

22.2.2 ISPF

ISPF stands for Interactive System Productive Facility. Some refer to it by its older name, SPF, or by its full name of ISPF/PDF. This alternative to TSO lets you carry out most basic functions by making menu selections and filling out forms called "panels" on your screen. The menus vary from system to system, but you will find the same important choices available at all MVS installations.

Since ISPF often makes things easier, it has become more popular than TSO as a way to get work done in MVS. TSO, however, offers more speed and flexibility; typing out a command that indicates what you want to do takes less time than going through a series of menus, and the command may have options that are unavailable to you when doing the same thing from an ISPF menu. Also, some terminals and hardware connections to an MVS system may not provide everything necessary to run ISPF, so you should still know the basic TSO commands. (For more information on terminal connections and the use of TSO versus ISPF, see section 23.1.1, "VTAM."

22.2.3 CICS

Developers use the Customer Information Control System to ease the development of end-user applications. Programmers write applications in a traditional mainframe programming language, usually COBOL, and include sections consisting of CICS commands. Before compiling the program, they run a program called a CICS preprocessor, which finds each CICS statement in the program and converts it into the appropriate series of commands for the programming language being used.

What do the CICS commands do? Usually, they accomplish tasks that are difficult in a language like COBOL. For example, if you want to write a series of COBOL commands that display a form to fill out on the end-user's screen, you must worry about the different makes and models of terminals that the user might have. If you let CICS worry about this for you, you can describe and display your input form with a minimum of commands and let the the CICS processor turn them into the appropriate COBOL commands. CICS commands also simplify the use of data files and the relationship between an end-user's activity and the contents of the files.

22.2.4 Other MVS Components

MVS is made up of many parts. Like TSO, many started as optional features, but eventually became standard issue at MVS installations. Some components became obsolete and are no longer offered; others, especially those that take advantage of the extra power of more recent versions of MVS (like MVS/ESA) are optional now.

Many popular MVS programs, useful enough to seem like part of the operating system instead of being a separate application used for a specific purpose, don't even come from IBM. The exis-

tence of these and IBM's optional MVS features mean that the list of software tools available at one MVS site may not be the same as those available at another. You can, however, take for granted that TSO and ISPF are always available, which is why this book covers them.

22.3 History

The history of MVS must begin with its lineage. It is descended from the forefather of modern operating systems: OS. IBM introduced OS, which stands for "Operating System," in 1964 when it announced the 360 series of computers. (For more on this, see the sidebar "IBM's 360 Series of Mainframes" in this book's introduction.)

The history of operating systems over the next few years is really the history of IBM's improvements to OS, as it introduced versions that allowed more users the ability to simultaneously run more programs and to let each of these programs use more memory. IBM called successive versions MFT (Multiprogramming with a Fixed number of Tasks), MVT (Multiprogramming with a Variable number of Tasks), SVS in 1972 (Single Virtual Storage) and MVS/XA, (Multiple Virtual Storage/Extended Architecture) in 1974.

In 1988, IBM introduced its ESA/370 mainframes, and a new version of MVS known as MVS/ESA (Extended System Architecture) to take advantage of the increased power of the new hardware. The version of MVS that you use has no effect on any of the basic commands explained in this book; improvements to the operating system usually involve additional power for users pushing the outer limits of the system.

Chapter 23 Getting Started with MVS

23.1 Starting Up

Logging on to an MVS system consists of two basic steps: entering your user ID and entering your password. Before you reach the point where you log on, however, you may need to go through an MVS component known as VTAM.

23.1.1 VTAM

VTAM (pronounced "vee-tam") stands for "Virtual Telecommunications Access Method." The phrase "access method" comes up often in MVS. An access method is a collection of programs that acts as an intermediary between you and something that would otherwise be difficult and complicated to gain access to.

VTAM is essentially a telecommunications program. When you connect to the MVS system, you might really be connecting to some hardware running VTAM, which then enables you to connect to the MVS system. If you see a screen that lists several possible commands to type and one of them is `DIAL`, then `DIAL VTAM` will probably connect you to the MVS system.

Sometimes you can choose between logging on to TSO directly or logging on to VTAM and using that to access the MVS TSO system. (For example, in addition to the `DIAL` command, the screen mentioned above might offer `TSO userid` as a choice.) Going through VTAM might offer better control of your terminal than a direct connection to TSO. This increased control might be reflected in the ability to run ISPF: displaying menus and forms on you screen to fill out requires a more sophisticated relationship between a computer and a terminal than the ability to just scroll lines of text from the bottom of the screen to the top. Trying to start up ISPF at the TSO prompt after connecting directly to TSO (and bypassing VTAM) might produce an error message similar to this:

```
INVALID TERMINAL ACCESS METHOD, ISPF VERSION 3 REQUIRES ACF/VTAM.
```

In other words, the connection made allows the primitive screen control necessary to let you enter TSO commands and watch the system's responses scroll up the screen, but not the screen control necessary to display menus and forms that you would fill out by moving your cursor from place to place.

23.1.2 Logging On

When you first connect to an MVS system, there are several ways to tell it that you want to log on. The simplest is to just type the following:

```
logon
```

The system responds by asking you for your user ID:

```
IKJ56700A ENTER USERID -
```

Type in the ID that you were assigned when you received the account. If the system does not recognize a user ID of "JOEUSER," it displays a message similar to this:

```
IKJ56420I Userid JOEUSER not authorized to use TSO
```

An alternate way to log on at some sites uses menus that offer you the choice of connecting to several different systems. This screen will let you know the syntax for logging on to the system you want. Joe User would use a command similar to this to log on to MVS's TSO component directly:

```
tso joeuser
```

Regardless of how you typed in your user ID, once the system knows it, it then prompts you for the password that goes with that ID:

```
ENTER CURRENT PASSWORD FOR JOEUSER-
```

When you type the characters of your password, they will probably not appear on your screen. This security feature prevents someone from learning your password by watching you log on.

Instead of prompting you for the password as above, the system may display a form to complete with the USERID field already filled out and the cursor waiting at the PASSWORD field for your response. Figure 23.1 shows an example.

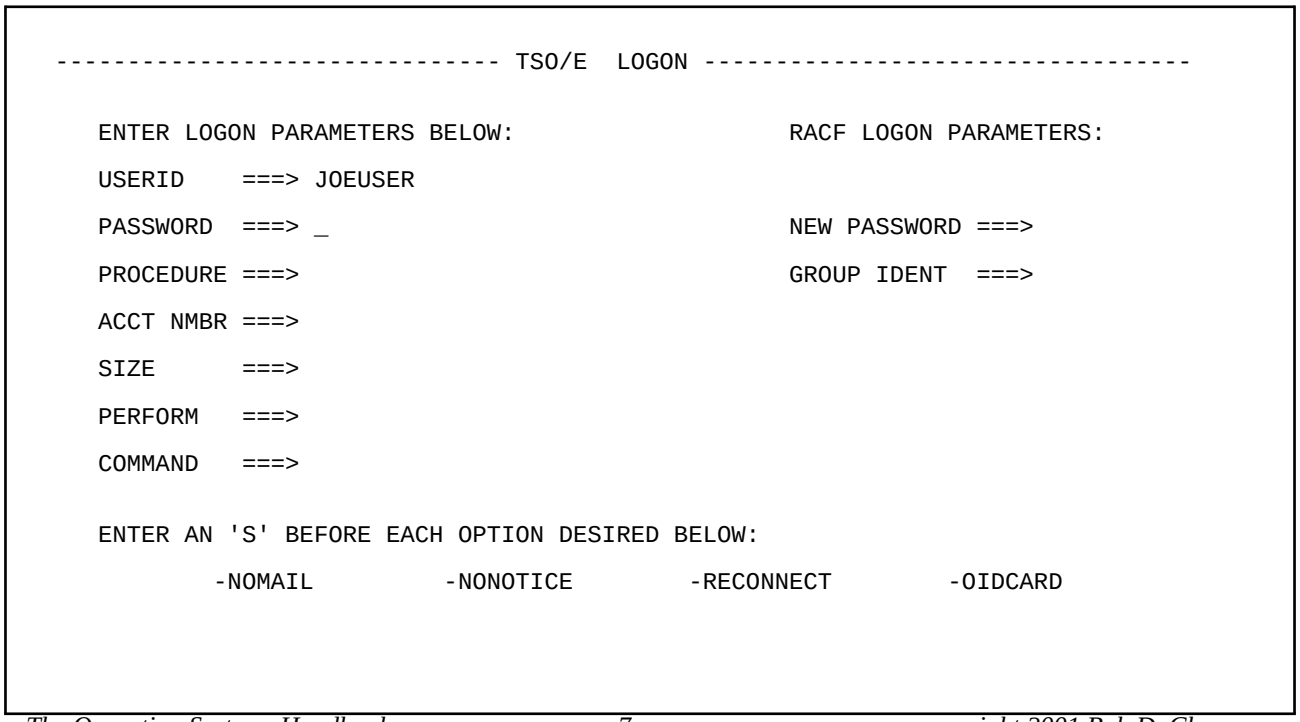

Figure 23.1 TSO logon form.

Don't worry about the other fields; type in your password (again, the characters do not display as you type) and press Enter.

Once you enter a valid ID and password, MVS displays some logon messages and the TSO READY prompt. Figure 23.2 shows a typical series of messages. (Because the system administrator can change the default prompt, it might be something other than the word "READY." Also, the system may automatically start up ISPF instead of displaying the TSO prompt; if not, we'll soon see how to start it up manually.)

```
ICH70001I JOEUSER  LAST ACCESS AT 18:31:55 ON MONDAY, JANUARY 11, 1994
JOEUSER LOGON IN PROGRESS AT 09:09:30 ON JANUARY 12, 1994
 ********************************************************************
 *                  O'Rourke Enterprises MVS/XA                     *
 ********************************************************************
 *    Use CLASS=T for all server batch JOBs or they may be cancelled.  *
 ********************************************************************
 *    Please report any problems to the Help Desk at 878-4531.        *
 ********************************************************************
 ***                         N  e  w  s  :                          ***
 ********************************************************************
 *    Due to a possible security breach, all users must change their   *
 *    passwords by Thursday, January 14.                              *
 ********************************************************************
You have no messages or data sets to receive.
**** NATIVE TSO READY ****
READY
```

Figure 23.2 Typical opening series of messages upon logon.

The messages appearing in upper case letters are from the system, telling you some statistics about this logon and the same user ID's last logon. The rest of the messages are from the system administrator, to keep you abreast of news about the system. The last line, READY, is the TSO command-line prompt. It means that the system is ready for you to type in TSO commands.

23.1.2.1 Reconnecting

Let's say you're accidentally disconnected from the system and you try to log on again. It may

tell you this

```
IKJ56425I LOGON REJECTED, USERID JOEUSER IN USE
IKJ56400A ENTER LOGON OR LOGOFF-
```

because you didn't log off properly. This isn't a problem; MVS provides the RECON option to the LOGON command, which means "reconnect me to my earlier session." If you respond to the above message with

```
logon joeuser recon
```

the system prompts you for your password and reconnects you, displaying the same screen you were viewing when you were disconnected.

If you log on by filling out the form displayed in Figure 23.1, note the line at the bottom that says, "ENTER AN 'S' BEFORE EACH OPTION DESIRED BELOW." (The "S" stands for "Select.") One of the options is "RECONNECT," so after you enter your password and before you press Enter, move your cursor to the RECONNECT option by repeatedly pressing the Tab key. If you type the letter "S" there and press Enter, the system reacts as if you had entered

```
logon joeuser recon
```

at the TSO prompt.

23.1.3 Entering Commands

When you start your session, and at any time throughout, the READY prompt shows that the system is ready for you to type in a TSO command.

When you type a command, it appears on the line under the READY prompt. After you press Enter, the command's output appears under the typed command. If there is too much to fit on the screen, asterisks appear on the last line to tell you that there is more to see. For example, the LISTCAT command lists file names (they're really called data sets; more on this later). Figure 23.3 shows the LISTCAT command entered when there isn't enough room for all of its output on the screen.

```
ICH70001I JOEUSER  LAST ACCESS AT 18:31:55 ON MONDAY, JANUARY 11, 1994
JOEUSER LOGON IN PROGRESS AT 09:09:30 ON JANUARY 12, 1994
 ********************************************************************
 *    Use CLASS=T for all server batch JOBs or they may be cancelled.  *
 ********************************************************************
 *    Please report any problems to the Help Desk at 878-4531.         *
 ********************************************************************
 ***                          N  e  w  s  :                          ***
 ********************************************************************
 *    Due to a recent security breach, all users must change their     *
 *    passwords by Thursday, January 14.                               *
 ********************************************************************
You have no messages or data sets to receive.
```

```
**** NATIVE TSO READY ****
READY
listcat
 IN CATALOG:SYS1.BGCCTLG
 JOEUSER.ACCNTING.CLIST
 JOEUSER.ACCNTING.COBOL
 JOEUSER.APL.ASM
 JOEUSER.CICS.CNTL
 JOEUSER.ERRORS.DATA
 JOEUSER.FMU.DATA
 JOEUSER.HLIPRINT.DATA
 JOEUSER.HLIPRINT.FOCUS
 JOEUSER.INVENTRY.DATA
 ***
```

Figure 23.3 Beginning of LISTCAT output when remainder doesn't fit on screen.

When you press Enter, more of the command's output appears. If the remaining output doesn't fit on the second screen, the asterisks will appear at the bottom again, waiting for you to press Enter. When you do reach the end, the READY prompt reappears, as shown in Figure 23.4.

```
 JOEUSER.INVENTRY.COBOL
 JOEUSER.ACCNTING.COBOL.BACKUP
 JOEUSER.ISPF.ISPPROF
 JOEUSER.JU.ASM
 JOEUSER.JU.CLIST
 JOEUSER.LOG.MISC
 JOEUSER.MASTER.DATA
 JOEUSER.OFFLINE.DATA
 JOEUSER.PROD.PROCLIB
 JOEUSER.RDBAPP.DATA
 JOEUSER.RDBAPP.OBJ
 JOEUSER.RDBAPP.REXX
 JOEUSER.TEST.CNTL
 JOEUSER.TEST.ASM
 READY
```

Figure 23.4 End of LISTCAT output.

23.1.3.1 Aborting Screen Output

If you find yourself repeatedly pressing Enter and regretting that you entered the command that produced so much output, you have an alternative to pressing Enter: the PA1 key. Pressing it puts you right back at the TSO `READY` prompt after it clears the screen, aborting the output of any command currently putting information on the screen. (When using a PC with a terminal emulation program instead of an actual mainframe terminal, check the emulation program's documentation to see which key acts as PA1.)

23.1.3.2 Command Parameters

Many commands need some information from you in order to do their job. For instance, when you type the `COPY` command, the system needs to know the name of the data set you want to copy and the name you want to assign to your new copy. As you'll see in the section on this command, you could type

```
copy old.dsname new.dsname
```

but if you just type

```
copy
```

by itself, MVS turns out to be fairly user-friendly about this abbreviated syntax—it prompts you for the information it needs:

```
copy
 ENTER 'FROM' DATA SET NAME -
jan13.memo
 ENTER 'TO' DATA SET NAME -
jan13.memo.backup
 READY
```

(VMS also does this, although MVS and VMS are completely different operating systems from completely different companies. IBM's VM and Wang's VS add to the name confusion—the fact that MVS, VS, VMS, and VM are four unrelated operating systems from three different companies is a big part of what makes the mini and mainframe world so confusing.)

23.1.3.3 Long Commands

Complicated commands may take up more than one whole line of your screen. If this happens, you can indicate that your command continues on the next line with the plus (+) or minus (-) sign. For example, listing the contents of the `TESTSEQ1.DATA` data set by entering

```
list +
testseq1.data
```

has the same effect as entering

```
list testseq1.data
```

Of course, there wouldn't be much point to breaking up a command this short over more than one line.

23.1.3.4 Case Sensitivity

TSO maps all entered commands to upper case, so it doesn't matter whether you type

```
LIST TESTSEQ1.DATA
```

or

```
LIST testseq1.data
```

or

```
list testseq1.data
```

at the READY prompt. They all list the contents of the data set called TESTSEQ1.DATA. In the TSO examples in this book, entered commands are shown in lower case to make it easier for you to distinguish between lines that the user types and lines that the system displays, which almost always appear in upper case.

23.1.3.5 Command-Line Options

IBM's on-line help and literature about TSO refer to command-line options as "operands." You don't need any special character to indicate that something you add to the command line is an operand for that command. For example, the LIST command's NONUM option indicates that you don't want line numbers listed with a data set's contents. You add it onto the command just as it is, with no "-" (as in UNIX) or "/" (as in VMS) or "(" (as in CMS) to indicate that it is a special option for the LIST command:

```
list testseq1.data nonum
```

Certain TSO commands have options that need extra information from you. This information is enclosed in parentheses right after the option's name. For example, the ALLOCATE command, which creates a new data set or indicates an existing data set for a program to use, sometimes takes the following form:

```
allocate dataset(dsname)
```

In this example, dsname represents the name of the data set to allocate. You would substitute a data set name between the parentheses.

Whether you enter an existing data set name or a new one depends on your purpose in entering the ALLOCATE command. We'll see more about this in section 24.1.7, "Allocating Data Sets." We'll also see that this command can take many more options than this—enough to give you practice at using the "+" character to spread your ALLOCATE command over two lines, as de-

scribed in section 23.1.3.3, "Long Commands."

23.1.4 Finishing Your MVS Session

Logging off is simple: just type

```
logoff
```

at the TSO `READY` prompt. The system then displays a message telling you the user ID being logged off, the time, and the date.

23.2 File Names

Files in MVS are known as data sets. Although most books on MVS tell you that "data set" is the term used for a file, keep in mind that the terms are not completely interchangeable. The word "file" has a specific meaning in MVS: when you use the `ALLOCATE` command to indicate a data set that a particular program will use, `FILE` is a synonym for `DDNAME`, or "data definition name." The program uses the ddname to refer to the data set. For example, if a database program saves report output in a data set instead of displaying it on the screen, it might expect the data set to have a `DDNAME` of `RPTOUT`. If you wanted the data saved in a data set called `JOEUSER.SUMRPT.TEXT`, you would allocate this data set with a `DDNAME` of `RPTOUT`. For more on this, see section 24.1.7, "Allocating Data Sets."

23.2.1 Sequential and Partitioned Data Sets

There are two kinds of data sets: sequential and partitioned. (Actually, there's a third called VSAM—pronounced "vee-sam"—used to store data for database applications, but that's a more advanced topic.) A sequential data set is like a regular file in other operating systems; a partitioned data set (also known as a PDS, or a "library") is like a group of files under one name. Each one of the files, or "members" of a PDS, is basically the same as a sequential data set. In fact, we'll see with the `COPY` command how easily you can take an existing sequential data set and make it a member of an existing partitioned data set.

Because a partitioned data set is a collection of "files," it's tempting to compare it to a directory or folder on another operating system. In practice, however, MVS users do not really use partitioned data sets this way. A single PDS usually holds a group of files that all serve the same purpose within their respective contexts and that all have the same characteristics (for example, record length, maximum size, and the units in which their size is defined). In VM/CMS, this would be like grouping together a collection of files with the same filetype; in other operating systems, it would be like grouping files with the same extension. For example, one PDS might hold several CLISTs (TSO command procedures). Another could hold the data files used by a certain database management program, and another could hold a collection of report specifications for using data from that program.

Section 23.3, "How Files Are Organized," describes the role of partitioned data sets in greater detail.

23.2.2 Line Numbers and Data Sets

You may have seen text editors on some operating systems that include line numbers next to the text. The two text editors used in MVS also do this, but there's an important difference when compared with other operating systems: MVS editors save the numbers with the data sets. If this ever presents a problem, most TSO commands offer an operand that lets you omit the line numbers—for example, when you print or copy a numbered data set.

23.2.3 Naming Data Sets

The rules for naming sequential and partitioned data sets are the same. (As we'll see, referring to a particular member of a PDS requires something extra added to the PDS name.)

A data set name is composed of pieces called qualifiers. A period separates each qualifier. For data sets that you create, MVS adds your user ID as the first, or "high-level" qualifier. For Joe User, the full name (or, in MVS parlance, the "fully qualified data set name") of a data set that he named `MYFILE.TEXT` would be `JOEUSER.MYFILE.TEXT`.

Some MVS users call this high-level qualifier the data set's "prefix." It's not always the user ID of the person who created it; it could be the name of the application that uses it or something else assigned by the system. The data sets that you create, however, nearly always have your user ID as their high-level qualifier.

When you refer to a data set, you rarely include the high-level qualifier in its name. The system assumes that your user ID is it. If you are including the high-level qualifier when specifying a data set name and don't want TSO to automatically add one, enclose the data set name in apostrophes. For a user ID of JOEUSER, this tells the system that you've added the "JOEUSER" yourself, like in the following:

```
'joeuser.myfile.text'
```

MVS users refer to the apostrophe as a "tick" and the period as a "dot," so if you were reading the above data set name out loud to someone you would say "tick joeuser dot myfile dot text tick."

You enclose a data set name in "ticks" to refer to a data set that is not your own, such as a data set that another user told you to use. Remember, all data sets have a high-level qualifier; if you enclose a data set name in apostrophes and leave out the high-level qualifier, or use an invalid one, MVS rejects it.

A fully qualified data set name can have as many parts (that is, qualifiers) as you want, as long as

the total number of characters (including the periods that separate the qualifiers) does not exceed 44 and no qualifier has more than eight characters. If Joe User's user ID is `JOEUSER`, this leaves him with 36 characters for the rest of his fully-qualified data set names, because MVS automatically adds his seven-character user ID and a period to the beginning of any data set he creates.

Qualifiers can contain letters and numbers, and must begin with a letter. The only other allowable characters are @, $, and #. (Note that the underscore, which is OK on most operating systems, cannot be used in MVS.) The case of alphabetical characters doesn't matter, since lower case letters will be converted to upper case.

Although your data set name could have over a dozen qualifiers, three is the most popular number. ISPF makes life easier for people who use data set names with three qualifiers; it even has a name for each of the three:

`project`	The user ID, or high-level qualifier.
`group`	The name that you make up to distinguish this data set from others with the same project and type.
`type`	The type of the data set—much like the concept of a file's type in VMS, its filetype in CMS, or its extension in DOS or UNIX.

For example, if Joe User had written three programs in the COBOL programming language, he might store them in data sets named `JOEUSER.INPUT.COBOL`, `JOEUSER.SUMRPT.COBOL`, and `JOEUSER.DETRPT.COBOL`. He could have named the last one `JOEUSER.DETAIL.REPORT.VER1.COBOL`, but the three-part name is more conventional.

Some MVS users call the type the "low-level qualifier." This applies not only to three-part data set names, but to all data set names with two or more parts—in other words, all data set names. The last part, the part that identifies what kind of data set it is, is always considered the low-level qualifier. (In the previous paragraph, all the data set names listed as examples have a low-level qualifier of `COBOL`—even `JOEUSER.DETAIL.REPORT.VER1.COBOL`.) We call the qualifiers in between intermediate qualifiers.

23.2.3.1 The Members of a Partitioned Data Set

The rules for naming a partitioned data set are the same as those for naming a sequential data set. Since a PDS is actually a collection of data sets, we also need a way to refer to a particular member of the PDS.

A member name goes right after the PDS name in parentheses. For example, if Joe User keeps his COBOL programs as members of a partitioned data set called `JOEUSER.SOURCE.COBOL`,

the member SUMRPT actually has the full name JOEUSER.SOURCE.COBOL(SUMRPT). Joe refers to it as SOURCE.COBOL(SUMRPT), knowing that the system adds the JOEUSER part for him. If he wants to refer to the fully-qualified name by enclosing it in apostrophes, he calls it 'JOEUSER.SOURCE.COBOL(SUMRPT)'.

Member names must follow the same rules as qualifier names: they can be up to eight characters long, they can contain letters, numbers and the @, $, and # symbols, and they must begin with a letter of the alphabet.

The limit of 44 characters on a data set name does not include the characters of a member name. A PDS with a 44-character name can still have members with eight-character names.

23.2.4 Wildcards

The only wildcard in TSO is the asterisk (*). You can use it to represent any single qualifier in a data set name. For example, the LISTCAT command can use the ENTRIES operand to list information about a single data set, as with the following:

```
listcat entries(dept.data)
```

Substituting an asterisk for one of the qualifiers means "perform this command on any data set that matches the other qualifiers, with anything in the asterisk's position." The following tells TSO to list the names of the data sets with "DEPT" as the second qualifier (remember, "JOEUSER" is the first) and anything as the third:

```
listcat entries(dept.*)
```

When you write out the data set name, you cannot make the asterisk the first part of the data set name. For example, to list all the data sets having "DATA" as their last qualifier, you could not enter this:

```
listcat entries(*.data)
```

To get around this, use fully-qualified data set names. Don't forget the apostrophes:

```
listcat entries('joeuser.*.data')
```

Remember, the use of the asterisk as a wildcard is not restricted to the LISTCAT command. Others, including COPY, RENAME, and DELETE can also use it.

23.3 How Files Are Organized

MVS keeps track of data sets in lists of their names and locations called catalogs. The system's master catalog stores a list of the names and locations of the user ID catalogs, and your ID's catalog is the list of your data sets. (Note that the command for listing data set names is LISTCAT; you're asking the system to list the contents of a catalog.)

The actual storage device that holds a particular data set is identified by a unique number called a Volume Serial Number, or VOLSER (pronounced "voll-sear"). Part of the purpose of the catalog system is to keep you from worrying about VOLSERs. On some ISPF screens where you enter a data set name, one line asks you to enter the "VOLUME SERIAL (IF NOT CATALOGUED)." In other words, it only needs to know this information if the data set is not in any catalog. (To see the Volume Serial Number of your data sets when you list their names, add the `VOLUME` operand to the `LISTCAT` command. See section 24.1.2, "Listing Data Set Names," for more on `LISTCAT`.)

A related term is VTOC, the Volume Table of Contents, a special data set that serves as the table of contents for a particular storage device. As with the VOLSER, advanced users sometimes use the VTOC to access their data more directly, but the use of catalogs relieves beginners from the need to worry about VTOCs. If it comes up in conversation, remember to pronounce it "vee-tok" and people will assume you know what you're talking about.

23.4 Available On-line Help

TSO's on-line is neither terrific nor terrible. Help information appears in all upper case letters, which is rather primitive, but it gives you syntax and a description for any command that you can name.

If you don't know command names, you can use the on-line help to find them. Typing `HELP` all by itself at the TSO `READY` prompt displays a list of TSO commands, with a brief description of each one's purpose. Figure 23.5 shows the first of these screens displayed.

```
 READY
help
 LANGUAGE PROCESSING COMMANDS:

 ASM         INVOKE ASSEMBLER PROMPTER AND ASSEMBLER F COMPILER.
 CALC        INVOKE ITF:PL/1 PROCESSOR FOR DESK CALCULATOR MODE.
 COBOL       INVOKE COBOL PROMPTER AND ANS COBOL COMPILER.
 FORT        INVOKE FORTRAN PROMPTER AND FORTRAN IV G1 COMPILER.

 PROGRAM CONTROL COMMANDS:

 CALL        LOAD AND EXECUTE THE SPECIFIED LOAD MODULE.
 LINK        INVOKE LINK PROMPTER AND LINKAGE EDITOR.
 LOADGO      LOAD AND EXECUTE PROGRAM.
 RUN         COMPILE, LOAD, AND EXECUTE PROGRAM.
 TEST        TEST USER PROGRAM.
 TESTAUTH    TEST APF AUTHORIZED PROGRAMS.

 DATA MANAGEMENT COMMANDS:

 ALLOCATE    ALLOCATE A DATA SET WITH OR WITHOUT AN ATTRIBUTE
             LIST OF DCB PARAMETERS.
 ALTLIB      DEFINE OPTIONAL, USER-LEVEL OR APPLICATION-LEVEL SETS OF
 ***
```

Figure 23.5 First TSO help screen.

For more detailed help about a specific command, enter the command's name as an operand to the HELP command. Figure 23.6 shows the HELP command being entered with LISTCAT as an operand, and the beginning of the HELP command's output.

```
help listcat

 FUNCTION -
   THE LISTCAT COMMAND LISTS ENTRIES FROM EITHER THE MASTER CATALOG OR
   A USER CATALOG.

 SYNTAX -
          LISTCAT   CATALOG('CATNAME/PASSWORD')
                    FILE('DNAME')
                    OUTFILE('DNAME')
                    LEVEL('LEVEL') | ENTRIES('ENTRYNAME/PASSWORD' ...)
                    CREATION('NNNN')
                    EXPIRATION('NNNN')
                    NOTUSABLE
                    CLUSTER  DATA  INDEX  ALIAS  SPACE  NONVSAM
                        USERCATALOG  GENERATIONDATAGROUP  PAGESPACE
                        ALTERNATEINDEX  PATH
                    ALL | NAME | HISTORY | VOLUME | ALLOCATION
   REQUIRED - NONE
   DEFAULTS - NAME
   ABBREVIATIONS -
       NOTE - IN ADDITION TO NORMAL TSO SHORT FORMS, THESE ARE ACCEPTED.
 ***
```

Figure 23.6 Command and output for help about LISTCAT.

Again, the asterisks at the bottom show that you're only looking at the first screen of a fairly extensive description of the command. The list of possible syntax variations alone take up almost the entire screen.

The separate lines show options that you may add after the LISTCAT command. When you see

several options listed on the same line, such as

```
ALL | NAME | HISTORY | VOLUME | ALLOCATION
```

in Figure 23.6, that means you can choose one of these, but not more than one.

```
OPERANDS -
  CATALOG('CATNAME/PASSWORD')
            - SPECIFIES THE NAME OF THE CATALOG CONTAINING THE ENTRIES
              TO BE LISTED.
  'CATNAME'
            - NAME OF THE CATALOG CONTAINING THE ENTRIES TO BE
              LISTED.
  'PASSWORD'
            - PASSWORD OF THE CATALOG CONTAINING THE ENTRIES TO BE
              LISTED.
  REQUIRED - 'CATNAME'
  FILE('DNAME')
            - IDENTIFIES THE VOLUMES THAT CONTAIN THE CATALOG
              ENTRIES TO BE LISTED.
  'DNAME'   - NAME OF THE DD STATEMENT THAT IDENTIFIES THE VOLUMES
              CONTAINING CATALOG ENTRIES TO BE LISTED.
  OUTFILE('DNAME')
            - IDENTIFIES THE ALTERNATE OUTPUT DATA SET.
  'DNAME'   - NAME OF THE JCL STATEMENT THAT IDENTIFIES THE
              ALTERNATE OUTPUT DATA SET.
  LEVEL('LEVEL')
            - SPECIFIES THE LEVEL OF ENTRY NAMES TO BE LISTED.
  'LEVEL'   - LEVEL OF ENTRY NAMES TO BE LISTED.
***
```

Figure 23.7 More help for LISTCAT.

So what do all these options mean? This is revealed after the syntax. For example, Figure 23.7 shows some later help material for the `LISTCAT` command; each option has at least one line describing it.

Chapter 24 Using Files in MVS

24.1 The Seven Most Important Commands

The seven most important TSO commands in MVS are:

`LISTCAT`	lists the data sets in a catalog.
`LISTDS`	lists the members in a partitioned data set.
`LIST`	displays the contents of data sets.
`COPY`	copies data sets and data set members.
`RENAME`	renames data sets and data set members.
`DELETE`	deletes data sets and data set members.
`ALLOCATE`	creates and provides access to data sets.

Remember, in addition to performing these functions with TSO commands, you can use ISPF. TSO, however, gives you more flexibility. In other words, if you see a lot of potential options when you enter "HELP" followed by the name of one of the commands in this section, ISPF won't necessarily have an equivalent to all of those options.

Nearly all of the commands described here require you to name the data set you wish to act on. The examples leave out the high-level qualifier, but they all work the same way if you include the high-level qualifier—as long as you remember to put it between apostrophes. For example, when logged on to Joe User's ID, there's no difference between entering

```
delete memos.text
```

and

```
delete 'joeuser.memos.text'
```

24.1.1 Common Error Messages

The simplest mistake to make at the TSO command line is typing in the name of a "command" that doesn't exist at the `READY` prompt. If you type in such a command, like

```
hlep listcat
```

when you meant to type `help listcat`, TSO tells you:

```
COMMAND HLEP NOT FOUND
```

In other words, TSO has no command called HLEP.

A classic mistake on many systems is entering a command that expects command-line parameters without entering as many parameters as it expects. (For example, entering the COPY command without saying what you want to copy and what to call the copy.) When using TSO, this is not a mistake; as we saw in section 23.1.3.2, "Command Parameters," TSO prompts you for the rest of the necessary information.

If you try to perform a command on a non-existent data set, you will find that TSO has a variety of ways to tell you that there's a problem. Trying to rename a non-existent data set called MYNOTES.TEXT gives you the following error message

```
DATA SET MYNOTES.TEXT NOT IN CATALOG OR AMOUNT OF DATASETS EXCEEDS WORKAREA FOR GENERIC RENAME
```

while trying to copy the same non-existent data set gives you a much terser error message:

```
COPY     ENDED DUE TO ERROR
```

The justification for this disparity in error message information goes something like this: real TSO commands, like RENAME, get reasonable error messages, but COPY is not a real TSO command. It's actually a TSO utility program, but it's so useful that it's installed wherever TSO is installed—sort of like ISPF.

Another error message is worth examining because of its use of MVS terminology. Try to delete a non-existent data set called MYNOTES.TXT and you'll see

```
  ERROR QUALIFYING JOEUSER.MYNOTES.TEXT
```

The system first adds the high-level qualifier (the user's ID) so that it has the complete data set name, and then it uses this name to find the data set. Obviously, it fails; there's no such data set in the JOEUSER catalog. (You'll get the same error message for a data set that seems to exist, but just isn't part of your catalog. See section 24.1.8, "Adding a Data Set to a Catalog," for more on this.)

If you try to do something to a data set and leave off the last qualifier, TSO displays the final qualifiers for data sets that begin with whatever you entered. For example, if you have data sets named APRIL.MEMOS.SENT and APRIL.MEMOS.RECEIVED and you enter the command

```
delete april.memos
```

TSO responds like this:

```
  QUALIFIERS FOR DATA SET APRIL.MEMOS ARE
  SENT      RECEIVED
  ENTER QUALIFIER-
```

You then enter the final qualifier of the data set that you want to delete and press Enter.

TSO reacts similarly when it understands a command name, but not one of its operands. For ex-

ample, section 24.1.2.2, "Listing a Partitioned Data Set's Members," shows how you can list a partitioned data set's members by adding the MEM operand to the LISTDS command after the data set's name. If you misspelled it like the following,

```
listds setup.clist mom
```

TSO responds with the following:

```
  INVALID KEYWORD, MOM
  REENTER THIS OPERAND -
```

This gives you the chance to enter the wrong part again. If you really thought that MOM was right and you're not sure what to enter now, enter a question mark and TSO responds with a description of the proper syntax for the command you entered:

```
 ?
  SYNTAX -
           LISTDS   'DSLIST'  STATUS  HISTORY  MEMBERS  LABEL
                    CATALOG('CAT.NAME') LEVEL
    REQUIRED - 'DSLIST'
    DEFAULTS - NONE
```

24.1.2 Listing Data Set Names

TSO users use two commands to list data set names: LISTCAT lists the names of data sets in a catalog, and LISTDS lists a partitioned data set's members.

24.1.2.1 Listing a Catalog's Data Sets

If you enter the LISTCAT command without any operands, it lists the names of data sets in your user ID's catalog, as shown in Figure 24.1.

```
listc
 IN CATALOG:SYS1.BGCCTLG
 JOEUSER.ACCNTING.CLIST
 JOEUSER.ACCNTING.COBOL
 JOEUSER.APL.ASM
 JOEUSER.CICS.CNTL
 JOEUSER.ERRORS.DATA
 JOEUSER.FMU.DATA
 JOEUSER.HLIPRINT.DATA
 JOEUSER.HLIPRINT.FOCUS
 JOEUSER.INVENTRY.DATA
 JOEUSER.INVENTRY.COBOL
 JOEUSER.INVENTRY.COBOL.BACKUP
 JOEUSER.ISPF.ISPPROF
 JOEUSER.JU.ASM
 JOEUSER.JU.CLIST
 JOEUSER.JU.CLIST1A
 JOEUSER.JU.CNTL
 JOEUSER.JU.COBOL
 JOEUSER.JU.DATA
```

```
  JOEUSER.JU.DS14
  JOEUSER.JU.LOAD
  ***
```

Figure 24.1 Sample output of the LISTCAT command.

(The three asterisks on the last line show that TSO is only displaying the first screenful of names, not the entire list.) Note how only the first five letters of "LISTCAT" were entered. This abbreviation is all you need.

Two operands to the `LISTCAT` command let you narrow down the list of data set names to display. The `ENTRIES` operand takes a data set name as an argument, in parentheses, and lists the name of the data set and the name of the catalog that holds information about the data set. The following command

```
listc entries(inventry.cobol)
```

produces the following output:

```
  NONVSAM ------- JOEUSER.INVENTRY.COBOL
       IN-CAT --- SYS1.BGCCTLG
```

The `ENTRIES` operand is more useful when you use an asterisk as a wildcard in the data set name included in the parentheses. Substituting an asterisk for one (and only one) of the data set name qualifiers causes the `LISTCAT` command to list all the data set names that have any qualifier in the asterisk's position, and whose other qualifiers match the corresponding ones in the data set name between the parentheses. See section 23.2.4, "Wildcards," for examples.

The `LEVEL` operand tells the `LISTCAT` command to list data set names that begin with the qualifier (or qualifiers) entered as a parameter. For example, entering

```
listc level(joeuser.inventry)
```

produces the following output:

```
  JOEUSER.INVENTRY.DATA
  JOEUSER.INVENTRY.COBOL
  JOEUSER.INVENTRY.COBOL.BACKUP
```

You can use the `LEVEL` operand to list the data set names in another user ID. For example, entering

```
listc level(mjones)
```

lists all the data set names in the MJONES catalog—if you have permission to list them. Either Mary Jones must authorize you to list them, or you must be in her group. MVS's high priority on data security means that your default access to the list of another user's data sets is no access. For an extra touch of heavy security, a warning message may be mailed to Mary telling her that you tried to list the names of her data sets.

24.1.2.2 Listing a Partitioned Data Set's Members

Use the LISTDS command to determine if a data set is sequential or partitioned, and if partitioned, the names of its members.

When you enter this command with one data set name as a parameter, like this,

```
listds errors.data
```

you'll see output similar to this:

```
  --RECFM-LRECL-BLKSIZE-DSORG
    FB    80    23200   PO
  --VOLUMES--
    USERMB
```

The most important part of this right now is the data set organization, designated by "DSORG." The "PO" means that INVENTRY.COBOL is a partitioned data set; if it was sequential, the DSORG value would be PS. Other values may show up here, for things like specific types of database data sets, but PO and PS are the DSORG values that you will see most often for data sets in your user ID.

To list the members of a partitioned data set, add the MEMBERS (or just MEM) operand to the LISTDS commands. Figure 24.2 shows an example.

```
 listds inventry.cobol mem
  JOEUSER.INVENTRY.COBOL
  --RECFM-LRECL-BLKSIZE-DSORG
    FB    80    23200   PO
  --VOLUMES--
    USERMB
  --MEMBERS--
    SETUP
    ADDNEW
    DELETE
    SUMRPT
```

Figure 24.2 Listing a data set's members by adding MEM to the LISTDS command.

Adding the MEM operand when using LISTDS with a sequential data set has no effect—the command lists information about the data set as if you hadn't included the MEM operand. Figure 24.3 shows an example.

```
listds autoexec.bat mem
 JOEUSER.AUTOEXEC.BAT
 --RECFM-LRECL-BLKSIZE-DSORG
   VB    84    6233    PS
 --VOLUMES--
   USERMC
```

Figure 24.3 Adding MEM to LISTDS with a sequential data set.

As with the ENTRIES operand of the LISTCAT command, you can substitute the asterisk wildcard for one of the qualifiers in the specified data set name. This lets you list information about more than one data set, but only if their names have several qualifiers in common.

24.1.3 Looking at Data Sets

Use the LIST command to display the contents of a data set. Like the COPY command, LIST is technically a utility and not part of TSO. It's so basic and useful, however, that you can almost take for granted that it is installed on the system you're using.

Using LIST is simple: just add the name of the data set you want to display. For example, typing

```
list schedule.text
```

displays the contents of the data set SCHEDULE.TEXT:

```
  SCHEDULE.TEXT
00010 APRIL 7
00020 10 AM meet Mary Ann in conference room about new office space
00030 1 PM Lunch with Jimmy
00040 3:30 meet Frank in his office.  Bring budget notes.
READY
```

The LIST command is a good example of how you can treat a partitioned data set's individual members like sequential data sets. To view the contents of the APRIL6 member of the MEMOS.TEXT partitioned data set, type the following:

```
list memos.text(april6)
```

The system displays the contents of that member:

```
MEMOS.TEXT(APRIL6)
     Joe -
     Can you come to my office tomorrow at 3:30 to discuss
     the proposed budget?  If not, let me know as soon as
     possible so that I can find another time that is
     convenient for Jack and Mary.
     - Frank
```

If you try to list a partitioned data set without specifying a member name, MVS looks for a member named TEMPNAME in that PDS and lists it if found. Otherwise, it tells you that it couldn't find it:

```
MEMBER TEMPNAME NOT IN DATA SET MEMOS.TEXT+
```

24.1.4 Copying Data Sets

In its simplest form, copying data sets is similar to copying files on most other operating systems. Entering

```
copy schedule.text schedule.text.backup
```

instructs the system to make a copy of SCHEDULE.TEXT called SCHEDULE.TEXT.BACKUP. If a data set named SCHEDULE.TEXT.BACKUP already exists, TSO copies SCHEDULE.TEXT right on top of it without displaying any warning message, assuming that the two data sets have a compatible data set organization (DSORG).

Some data sets include line numbers and some don't. The default action of COPY is to include the line numbers, and if the source data set has none, you might see the following error message:

```
COPY TERMINATED, RENUMBERING ERROR+
```

If this happens, enter the copy operation again, but include the operand that instructs TSO not to bother with any line numbers in the source data set:

```
copy schedule.text schedule.text.backup nonum
```

24.1.4.1 Copying and Partitioned Data Sets

Copying an entire PDS uses the same syntax as copying a sequential data set. To copy a specific member of a PDS, you can substitute the name of a PDS member for either or both of the data sets in the COPY syntax shown above. For example, the following makes a copy of the APRIL6 member of the MEMOS.TEXT PDS. The copy is a sequential data set called FRANK.TEXT:

```
copy memos.text(april6) frank.text
```

Similarly, you could add a copy of an existing sequential data set to a partitioned data set as a new member of that PDS with a command like this:

```
copy joan.text memos.text(april7)
```

When copying a member from one PDS to another, it's not enough to enter the name of the destination PDS. If you enter

```
copy memos.text(april6) memos1994.text
```

you'll get the following error message:

```
 DATA SET ORGANIZATIONS ARE NOT COMPATIBLE+
```

Your source data set here is the functional equivalent of a sequential data set, and your destination is a partitioned data set—that's why they're incompatible. To indicate that you want `APRIL6` added as a member of the PDS `MEMOS1994.TEXT`, enter the following:

```
copy memos.text(april6) memos1994.text(april6)
```

Of course, you can call this new member of `MEMOS1994.TEXT` anything you want; it doesn't have to have the same name as the original.

There a couple of points to keep in mind about copying data sets:

- If you enter the above command and the PDS `MEMOS1994.TEXT` doesn't exist, TSO creates it for you before adding `APRIL6` as its only member.
- If you specify a partitioned data set as the source of your copy and a nonexistent PDS as the destination, TSO makes a copy of the whole PDS.
- You will see in section 24.1.7, "Allocating Data Sets," that when data sets are created, one characteristic specified about them is whether its records (lines) are of fixed or variable length. You can copy a fixed-length member to a PDS with variable-length records, but you can't copy from a PDS with variable-length records to a destination with fixed-length records.

The "Allocating Data Sets" section also describes how creating a new data set means specifying its maximum size in directory blocks. If a PDS gets too big and runs out of directory blocks, copying it somewhere else is the first step toward fixing the problem. After you make a copy, you can reallocate it and then copy the temporary copy back onto the original.

24.1.5 Renaming Data Sets

Using the `RENAME` command is simple. Just indicate the data set (or PDS member) to rename and its new name. The following renames the data set `FRANK.TEXT`, giving it the new name `FRANK.AUG23.TEXT`:

```
rename frank.text frank.aug23.text
```

Renaming a member of a partitioned data set uses similar syntax:

```
rename memos.text(april6) memos.text(fbapril6)
```

A renamed member will still be in the same data set, so MVS doesn't allow you to change any of the qualifiers that make up the member's PDS name.

Renaming a whole PDS and leaving its members' names alone has the same syntax as renaming a sequential data set:

```
rename memos.text oldmemos.text
```

You can rename multiple data sets if their names are all the same except for one qualifier in the same position in each data set. For example, to rename `MEMOS.JAN.TEXT`, `MEMOS.FEB.TEXT`, and `MEMOS.MAR.TEXT` to the names `MEMOS.JAN.TEXTBAK`, `MEMOS.FEB.TEXTBAK`, and `MEMOS.MAR.TEXTBAK` enter the following:

```
rename memos.*.text memos.*.textbak
```

24.1.6 Deleting Data Sets

To delete a data set, just type the `DELETE` command followed by the name of the data set you want to delete:

```
delete memos.jan.text
```

After deleting the data set, TSO confirms the deletion:

```
 ENTRY (A) JOEUSER.MEMOS.JAN.TEXT DELETED
```

To delete a single member of a data set, just specify the member name with its data set:

```
delete memos.text(apr6)
```

As with the `RENAME` command, you can enter an asterisk instead of one of the qualifiers to delete multiple data sets at once. The following command deletes all data sets with a first qualifier of `MEMOS` (after the high-level qualifier of the user's ID), a low-level qualifier of `TEXT`, and any second qualifier:

```
delete memos.*.text
```

To delete multiple data sets with completely different names, you don't have to enter the `DELETE` command multiple times; just list the names of each data set between parentheses:

```
delete (apl.asm test.data memos.text)
```

TSO responds with messages about the deletion of each data set:

```
  ENTRY (A) JOEUSER.APL.ASM DELETED
  ENTRY (A) JOEUSER.TEST.DATA DELETED
```

```
ENTRY (A) JOEUSER.MEMOS.TEXT DELETED
```

24.1.7 Allocating Data Sets

There are two kinds of allocation:

- Allocation of an existing data set usually sets it up for use by a particular program.
- Allocation of a non-existent data set is how you create new data sets.

If you have used other operating systems but are new to MVS, data set allocation is the most difficult concept to get used to, because it has no direct equivalent in other operating systems. (VM/CMS, being another IBM mainframe operating system, has a related command called `FILEDEF`, but it's not as crucial to survival as `ALLOCATE` is to MVS users.) With most operating systems, telling the text editor to edit a non-existent file creates a file by that name; in MVS, however, you must explicitly create a new data set by allocating it before you can do anything with it.

Allocation requests access to a data set, whether it already exists or not. There are various levels of access, and if you try to allocate a data set so that you can modify its contents while someone else currently has it allocated for modification, MVS won't let you. This is part of the point: more than one user trying to modify the same data set could result in the loss of one user's work. This doesn't mean that multiple users can't edit the same data set; they just need to use software—for example, a database management system—that coordinates their actions so that no user's work destroys another's.

24.1.7.1 Allocating Existing Data Sets

There are several possible reasons to allocate an existing data set:

- Specifying the data sets that will be needed by an application program you're about to use.
- Telling the system where to find things like your command procedures (CLISTs), error list partitioned data sets, and startup procedures.
- Redirecting output from a program to a new destination, like the printer, a data set, or your terminal.

The first of these is the most important for now. An application program often refers to a data set that it uses by a data definition name, or "ddname." The documentation for that program tells you the ddnames it expects to find already assigned to the data sets it needs. Before starting the program, you assign these names with the following syntax:

```
allocate dataset(dataset.name) ddname(ddname) [old/shr/mod]
```

The order of these three operands does not matter. Also, you can abbreviate the keyword DATASET to DSNAME or even just DA, and DDNAME can be written as FILE, FI, or just F. This is why you must be careful about using the terms "data set" and "file" interchangeably: MVS users sometimes use the term "file" to mean "ddname," which is quite distinct from a data set name. Like a data set qualifier, a ddname can have no more than eight characters.

The final operand in the syntax above shows the status, or the kind of access to the data set that you want:

OLD	requests exclusive access to the data set. If you or the program that needs to use the data set will modify its contents, you need this type of access. This is the default.
SHR	requests sharable access to the data set. If you or the program will only read the data (for example, for running a report), you don't need to prevent others from using it by requesting OLD or MOD access.
MOD	requests access allowing the appending of data onto the end of the data set. Unlike OLD, which requests access to edit data currently existing in the data set, we use MOD to add new data—for example, to perform data entry with a database.

Again, when you need this command, the chances are slim that you will need to figure out all the syntax elements yourself. Part of the job of the documentation for MVS application programs that require specific data set allocations is to explain the details of these allocations. Just remember to recognize the various alternatives to the keywords DATASET (DSNAME, DA) and DDNAME (FILE, FI, or F) when it tells you the necessary allocations.

For example, the documentation for the UpRiteBase database system might tell you that it expects data files to have the ddname "urdata" associated with them, and that you would allocate the data for your inventory data file with the following command (note that ALLOCATE only needs its first five letters included):

```
alloc da(inventry.data) fi(urdata) old
```

This has exactly the same effect as entering this:

```
allocate dataset(inventry.data) ddname(urdata) old
```

Since you may need to allocate more than one data set, and a typo in the ddname could cause trouble, you'll want the ability to list out the allocations made in your session. See section 24.1.7.4, "Finding Out a Data Set's Allocation Status," for more on this.

24.1.7.2 Allocating New Data Sets

When creating a new data set, the system needs to know much more about it than just its name—whether it's sequential or partitioned, how much space to set aside for it, the length of its lines, and several other details that few users completely understand. This brings us back to the high degree of customization possible with MVS: allocating data sets with the best possible settings for your site boosts the efficiency of the system. (And there's the converse: doing it badly, on a large scale, can slow down the machine.)

Fortunately, there's a way to tell TSO to allocate a new data set with a particular name and to copy all of its other attributes from an existing data set: the `LIKE` operand. For example, to create a new data set called `PROJECT2.COBOL` and model its attributes after `PROJECT1.COBOL`, enter this:

```
allocate dataset(project2.cobol) like(project1.cobol)
```

By using a fully-qualified data set name for the model, you can easily copy the details from someone else's data set:

```
allocate dataset(project2.cobol) like('mjones.project1.cobol')
```

If you want to model your new data set after an existing one in all details except certain ones, you can specify those extra details after the `LIKE` operand. For example, if you know that Mary Jones' `PROJECT1.COBOL` data set is a partitioned data set with room for 18 members, but you want to allow more room in yours, include the `DIR` operand, which specifies how many directory blocks to set aside for a new PDS:

```
allocate dataset(project2.cobol) like('mjones.project1.cobol') dir(6)
```

Since each block holds about six members (although this varies from installation to installation—ask about the figure at your site), this leaves room for 36 members.

If you don't model a new data set after an existing one, a new data set requires you to at least specify the following:

- Its name.
- How much space to set aside for it.
- Whether it's a sequential or partitioned data set.

We've already seen that the `DATASET` operand specifies the name. We also saw earlier that the `LISTDS` command produces output in which one of the headings, DSORG, shows a data set's organization. It will usually be either sequential (PS) or partitioned (PO). Specifying this with the `ALLOCATE` command uses the same abbreviations: you add the `DSORG()` operand with either `PS` or `PO` between the parentheses. (Other values are possible, but most data sets will be one of these two.) If you do specify `PO` as the data set organization, don't forget to also include the `DIR` operand in order to indicate the maximum number of members that may be stored in the parti-

tioned data set.

When you specify the space to set aside for your new data set, you must first decide on the units you will use to indicate this space. You have a choice of TRACKS, CYLINDERS, or BLOCKS. Most typical data sets are just a handful of blocks or tracks. Only very large data sets are allocated in terms of cylinders.

To understand tracks and cylinders, picture a mainframe's storage device as a stack of 15 or so hard disks. A typical individual disk has 2,600 tracks arranged as concentric rings. Each track holds about 47 kilobytes of data. A cylinder is a collection of tracks represented by the same track on each disk in the pack; a typical size for a cylinder is 15 times 47K, or 705K.

A block is a unit of storage whose size you specify with the BLKSIZE or BLOCK operand. The best possible size depends on the particular model of storage device that your system uses; typical values are 2160, 3600, and 6160. Fortunately, we've already seen the command to check an existing data set's block size: the LISTDS command. Along with the data set organization (DSORG), which tells you whether a data set is partitioned or not, LISTDS displays BLKSIZE as one of the headings describing a data set. Investigating the block size of existing data sets gives you an idea of how to set this parameter if you choose to allocate your new data set in units of blocks instead of tracks or cylinders.

Once you decide on the units of storage, you must choose how many of these units to set aside with the SPACE operand. SPACE takes two arguments: the primary allocation, which indicates how much space you expect the data set to need, and the secondary allocation, a backup amount that the system provides if the data set grows beyond the primary amount. The figure for the secondary amount is usually half of the primary amount.

The following command allocates a sequential data set called MYTEST.TEXT and sets aside 10 tracks of space for it, with 5 tracks as the amount of secondary space to add on if 10 isn't enough:

```
allocate dataset(mytest.text) dsorg(ps) tracks space(10 5)
```

The next command creates a partitioned data set named INVENTRY.COBOL with room for 10 members. Its total space is 10 8000-byte blocks, with 5 blocks set aside as the secondary storage if necessary. Note that the command won't fit on one line; as section 23.1.3 ("Entering Commands") explains, we add a plus sign at the end of one line to show that the next typed line is a continuation of the first one.

```
allocate dataset(inventry.cobol) block(8000) space(10 5) +
dsorg(po) dir(10)
```

Another common operand shows the disposition of the data set, or what happens to it when it is unallocated (see section 24.1.7.3, "Unallocating Data Sets," for information on how this happens.) There are three possible operands for the data set's disposition:

CATALOG Add the new data set to the master catalog, because you plan to keep

this data set around. This is the default, so you don't need to add it to your ALLOCATE command if you choose it for your data set.

DELETE — Delete the data set as soon as it is unallocated. Use this for temporary data sets.

KEEP — Don't delete it, but don't add it to the catalog. Leave this option to the experts. Part of the point of the catalog is to make it unnecessary for you to know the physical details of the data set's storage; accessing an uncataloged data set requires you to know more than you want to about your data set.

The following command allocates a sequential data set that MVS deletes as soon as the data set is unallocated:

```
allocate dataset(temp.tmp) ddname(tempfile) dsorg(ps) tracks space(10 5) delete
```

The last two important ALLOCATE operands to be aware of are RECFM, for Record Format, and LRECL, or Logical Record Length.

A record format of V means that a data set's records have variable length. ("Records" here means "lines." The terms were once fairly synonymous, but as the science of dealing with databases advanced, the term "record" took on a much more specific meaning.) In other words, the lines aren't all the same length. A format of F means that they have fixed length. If any lines seem shorter than others, the system inserts spaces to make their lengths equal.

When you indicate the record format, you also indicate whether to use blocked records by including (or leaving out) a comma and the letter "B." For example, RECFM(F,B) tells MVS to make them fixed length, blocked records; RECFM(V) means variable-length, unblocked records. Generally, you'll want blocked records unless you're told specifically otherwise. For example, if a program's documentation tells you about data sets to create for that program's use, it tells you how it expects RECFM to be set for these data sets.

If the records have a fixed length, you must set this length—in other words, you must indicate how long the records are. For variable length records, you must indicate the longest possible length. Use the LRECL, or Logical Record Length operand, for this. A setting of LRECL(80) is pretty common.

The following command allocates a data set called HEYJOE.TEXT as a fixed-length, blocked data set with a record length of 80. Note how the BLOCK operand is an even multiple of 80 (1760/80 = 22). This way, when the system reads or writes a block of data, it never includes a partial record, so input and output will be more efficient.

```
allocate dataset(heyjoe.text) space(10 5) tracks dsorg(ps) +
recfm(f,b) lrecl(80) block(1760)
```

The ALLOCATE command actually has over fifty possible operands. If you're curious to see them all, enter HELP ALLOCATE to display further information. I guarantee that you'll be using that PA1 key ("Aborting Screen Output," section 23.1.3.1) before you've seen them all!

24.1.7.3 Unallocating Data Sets

When you log off, the system automatically unallocates any allocated data sets. You may want to unallocate one or more data sets before logging off if someone else needs to use one that you've been using and you're not ready to log off yet. You also may want to unallocate a data set if you've allocated it with some of the wrong details, like its disposition, and you want to redo it. MVS won't let you allocate a data set that's already been allocated, so you must unallocate it first.

The FREE command makes this possible. The only operand it really needs is the name of the data set to unallocate. For example, to free up the TEMP.TMP data set mentioned above, enter

```
free dataset(temp.tmp)
```

Because the allocation of TEMP.TMP in section 24.1.7.2 included a disposition of DELETE, unallocating it deletes it.

You also have the option of unallocating it by specifying the ddname used to allocate it. Since each allocated data set must have a unique ddname, MVS knows which data set you want to unallocate. Given the allocation of TEMP.TMP shown earlier, the following command has the same effect as the previous one:

```
free dataset(tempfile)
```

Both the DATASET and DDNAME operands of the FREE command can take multiple arguments. For example, if you had allocated the data sets MAYMEMOS.TEXT, JUNMEMOS.TEXT, and JULMEMOS.TXT with the ddnames MAY, JUNE, and JULY you could free all three with either the command

```
free dataset(maymemos.txt junmemos.txt julmemos.txt)
```

or with this:

```
free ddname(may june july)
```

You can unallocate all the data sets you've allocated with the following command:

```
free all
```

Note that this frees up all the data sets that you personally allocated, not all the allocated ones you may use. The system automatically allocates some, as you'll see in the next section, and it takes care of unallocating them.

24.1.7.4 Finding Out a Data Set's Allocation Status

Two commands show you a data set's allocation status. You already know one; the `LISTDS` command, described in section 24.1.2.2 ("Listing a Partitioned Data Set's Members"), can take an additional operand that shows a data set's ddname and disposition along with its block size and data set organization: `STATUS`. The command

```
listds dept.data status
```

produces the following output:

```
--RECFM-LRECL-BLKSIZE-DSORG--DDNAME---DISP
  FB    80    3120    PO     SYS00002 KEEP
--VOLUMES--
  USERME
```

It's a partitioned data set, the ddname looks like something the system assigned, and it's a keeper.

Instead of listing a particular data set's allocation status, you might want to list the names of all of the allocated data sets. Use the `LISTALC` command for this. Figure 24.4 shows some sample output from typing `LISTA` (you only need the first five letters) with no operands.

```
DSN220.DSNLOAD
SYS1.DFQLLIB
EDC.V2R1M0.SEDCCOMP
EDC.V2R1M0.SEDCLINK
SYS1.V2R3M0.SIBMLINK
IPO1.CMDPROC
EDC.V2R1M0.SEDCLIST
ISR.V3R2M0.ISRCLIB
UBU0.CLIST
UBU1.CLIST
DSN220.DSNCLIST
SYS1.SBLSCLI0
ISR.V3R2M0.ISRCLIB
ISP.V3R2M0.ISPEXEC
SYS1.SBLSCLI0
DSN220.DSNSPFP
EDC.V2R1M0.SEDCPNLS
ISP.V3R2M0.ISPPENU
ISR.V3R2M0.ISRPENU
ISF.V1R3M1.ISFPLIB
SYS1.SBLSPNL0
SYS1.DFQPLIB
ISP.V3R2M0.ISPMENU
***
```

Figure 24.4 Listing allocated data sets with LISTALC (or LISTA).

The asterisks at the bottom show that this is only the first screen of names. You don't have to allocate over 23 data sets to see output like Figure 24.4; the system automatically allocates quite a few data sets on its own, and their names are mixed in.

As with the `LISTDS` command, you can add the `STATUS` operand to the `LISTALC` command to find out the ddname and status of the allocated data sets. Figure 24.5 shows a sample.

```
--DDNAME---DISP--
DSN220.DSNLOAD
  STEPLIB  KEEP
SYS1.DFQLLIB
  ISPLLIB  KEEP
EDC.V2R1M0.SEDCCOMP
           KEEP
EDC.V2R1M0.SEDCLINK
           KEEP
SYS1.V2R3M0.SIBMLINK
           KEEP
IPO1.CMDPROC
  SYSPROC  KEEP
EDC.V2R1M0.SEDCLIST
           KEEP
ISR.V3R2M0.ISRCLIB
           KEEP
UBU0.CLIST
           KEEP
UBU1.CLIST
           KEEP
DSN220.DSNCLIST
           KEEP
SYS1.SBLSCLI0
***
```

Figure 24.5 Listing the ddname and status of allocated data sets by adding STATUS to the LISTALC command.

The first one listed, `DSN220.DSNLOAD`, has a ddname of `STEPLIB` and a status of `KEEP`. Most others on the list don't have ddnames. Ours is not to question why; it's the system's job to worry about these things.

24.1.8 Adding a Data Set to a Catalog

There's no specific command for adding a data set to a catalog, but as I mentioned in section 24.1.7.3, "Unallocating Data Sets," the ability to unallocate lets you re-allocate a data set with different operands. This makes it possible to take an existing data set that doesn't belong to a cat-

alog and add it to one.

The following series of commands and responses shows an attempt to take a data set called MEMOS.TXT and add it to the user's catalog. At first, the system won't allow this because it's already allocated and not part of a catalog. After the data set is freed up, the same allocation command works the second time and adds the data set to the catalog.

```
allocate dataset(memos.text) tracks space(4 2) catalog
 DATA SET JOEUSER.MEMOS.TEXT NOT ALLOCATED, REQUESTED AS NEW BUT CURRENTLY ALLOCATED
 READY
free dataset(memos.text)
 READY
allocate dataset(memos.text) tracks space(4 2) catalog
 READY
```

The lack of an error message after the second attempt shows that it worked.

Chapter 25 The MVS ISPF Text Editor

25.1 The ISPF Text Editor

Editing data sets is one of the few places in MVS where ISPF gives you far more power and flexibility than TSO. Given the choice between the ISPF text editor and the TSO `EDIT` text editor, virtually everyone chooses the former. If the lack of a proper system connection limits you to using TSO without access to ISPF, see section 25.8, "TSO's EDIT Text Editor," for the basics of editing data sets with the more primitive editor. (In fairness, I should mention one advantage of EDIT over the ISPF editor: you can get in and out of it much faster. The lack of menus and panels to fill out make it more efficient for quickly creating small data sets or performing simple edits to short data sets.)

25.2 Entering the ISPF Editor

To get into the ISPF editor, you must first enter ISPF by typing

```
ispf
```

at the TSO command prompt. This displays the ISPF main (or "primary") menu screen shown in Figure 25.1. (An MVS system administrator can customize the ISPF menu screens, so your main menu may not look exactly the same.)

```
------------------------  ISPF/PDF PRIMARY OPTION MENU  ------------------------
OPTION  ===> _
                                                          USERID   -JOEUSER
   0  ISPF PARMS  - Specify terminal and user parameters    TIME     - 09:24
   1  BROWSE      - Display source data or output listings  TERMINAL - 3278
   2  EDIT        - Create or change source data            PF KEYS  - 24
   3  UTILITIES   - Perform utility functions
   4  FOREGROUND  - Invoke language processors in foreground
   5  BATCH       - Submit job for language processing
   6  COMMAND     - Enter TSO Command, CLIST, or REXX exec
   7  DIALOG TEST - Perform dialog testing
   8  LM UTILITIES- Perform library administrator utility functions
   9  IBM PRODUCTS- Additional IBM program development products
  10  SCLM        - Software Configuration and Library Manager
   D  DB2         - DB2 Facilities
   H  HSM         - DFHSM Facilities
   C  CHANGES     - Display summary of changes for this release
   S  SDSF        - System Display and Search Facility
  SO  DFSORT      - DFSORT Facility
   T  TUTORIAL    - Display information about ISPF/PDF
   X  EXIT        - Terminate ISPF using log and list defaults
 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

Figure 25.1 The ISPF primary menu.

Your cursor appears next to the `OPTION ===>` prompt, waiting for you to enter the number of one of the menu choices. On the menu shown in Figure 25.1, entering `2` brings you to the entry panel shown in Figure 25.2, where you enter the name of the data set that you will edit. (Here's a nice shortcut: if you enter `ISPF 2` at the TSO prompt, you jump right to this screen.)

```
--------------------------  EDIT - ENTRY PANEL  ----------------------------
COMMAND ===>

ISPF LIBRARY:
   PROJECT ===>
   GROUP   ===>           ===>           ===>           ===>
   TYPE    ===>
   MEMBER  ===>                 (Blank or pattern for member selection list)

OTHER PARTITIONED OR SEQUENTIAL DATA SET:
   DATA SET NAME  ===>
   VOLUME SERIAL  ===>          (If not cataloged)

DATA SET PASSWORD ===>          (If password protected)

PROFILE NAME      ===>          (Blank defaults to data set type)

INITIAL MACRO     ===>          LMF LOCK   ===> YES    (YES, NO or NEVER)

FORMAT NAME       ===>          MIXED MODE ===> NO     (YES or NO)


 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

Figure 25.2 Entry panel for naming the data set to edit.

Each arrow (`===>`) points at a field of the entry panel to fill out. To move your cursor forward from field to field, press your Tab key. To move your cursor in the other direction, press Backtab. (On most PC keyboards, this means pressing the Shift key and the Tab key together; on a mainframe keyboard, Backtab has its own key.)

The only really crucial fields on this entry panel are the ones that you use to enter the name of the data set that you want to edit. (The other fields on the Edit entry panel are used for more advanced features like password protection and stored collections of settings known as edit pro-

files. If you don't know what to do with any of these other fields, leave them alone. Blank is a good default in ISPF.) As explained in section 23.2.3, "Naming Data Sets," fully qualified data set names often have a total of three qualifiers, and ISPF refers to the three as the data set's project, group, and type. Figure 25.3 shows how Joe User fills out the ISPF LIBRARY section of the entry panel to edit the data set `INVENTRY.COBOL`.

```
ISPF LIBRARY:
   PROJECT ===> JOEUSER
   GROUP   ===> INVENTRY  ===>           ===>           ===>
   TYPE    ===> COBOL
   MEMBER  ===>                 (Blank or pattern for member selection list)

OTHER PARTITIONED OR SEQUENTIAL DATA SET:
   DATA SET NAME  ===>
   VOLUME SERIAL  ===>          (If not cataloged)
```

Figure 25.3 Sample entries in the ISPF LIBRARY section of the EDIT entry panel to edit INVENTRY.COBOL.

To edit a member of a partitioned data set, enter the three parts of the PDS name in the PROJECT, GROUP, and TYPE fields, and then the member name in the MEMBER field.

If your full data set name does not have three qualifiers, enter its name at the DATA SET NAME field under the line "OTHER PARTITIONED OR SEQUENTIAL DATA SET." You don't need the high-level qualifier; ISPF assumes that it's your user ID. If you don't want it to automatically add this, do the same thing that you do when entering the data set name as part of a command on the TSO command line: enclose it in apostrophes. (You really only need the first one.)

Figure 25.4 shows how Joe fills out the DATA SET NAME field to enter the same data set name in the panel.

```
ISPF LIBRARY:
   PROJECT ===>
   GROUP   ===>           ===>           ===>           ===>
   TYPE    ===>
   MEMBER  ===>                 (Blank or pattern for member selection list)

OTHER PARTITIONED OR SEQUENTIAL DATA SET:
   DATA SET NAME  ===> INVENTRY.COBOL
   VOLUME SERIAL  ===>          (If not cataloged)
```

Figure 25.4 Sample entries in the DATA SET NAME section of the EDIT entry panel to edit INVENTRY.COBOL.

You don't always have to fill out the data set name; once you've entered a data set's qualifiers in the PROJECT, GROUP, and TYPE fields, the ISPF editor redisplays the qualifier names you typed there the next time you use this entry panel. To edit that data set again, just call up that panel and press the Enter key. To edit a data set with a similar name, you only need to change the appropriate fields. (However, if you entered the name at the DATA SET NAME field, ISPF will not remember what you typed there.)

One way or another, some existing data set's name must be entered. With many text editors included with other operating systems, telling the editor to edit a data set that doesn't exist causes it to create a file with that name. The ISPF editor, however, expects the data set to already exist, even if it's empty. If you want to create a new data set, you must allocate it first with the ALLOCATE command. (For more on this, see section 24.1.7.2, "Allocating New Data Sets.")

Once you tell ISPF the data set that you want to edit press Enter. ISPF displays the editing screen, as shown in Figure 25.5.

```
EDIT ---- JOEUSER.INVENTRY.COBOL ----------------------------- COLUMNS 001 072
COMMAND ===>                                                  SCROLL ===> PAGE
****** ***************************** TOP OF DATA ****************************
==MSG> -WARNING- THE UNDO COMMAND IS NOT AVAILABLE UNTIL YOU CHANGE
==MSG>           YOUR EDIT PROFILE USING THE COMMAND "RECOVERY ON".
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
''''''
****** ************************** BOTTOM OF DATA ****************************
 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

Figure 25.5 ISPF editor screen for an empty new data set.

Besides showing that you're using the ISPF EDIT program, the top line of the screen shows the name of the data set being edited and which columns of this data set are being displayed. The second line has the prompt where you enter editing commands and, to the right of where it says `SCROLL ===>`, a message telling you how far you will scroll when you press the PF7 or PF8 keys to move up or down in your data set.

The left side of the screen shows the numbers automatically added to the lines of your data set. Where no lines of data exist, the editor shows six apostrophes (`''''''`) instead.

The main blank space on the screen is where you enter your text. When editing an existing data set, this is where it appears. As Figure 25.5 shows, other kinds of information can also appear there. The column on the left identifies the type of this information. For example, `==MSG>` means that the line shows a message from the editor to you. The message shown in the example tells the user that the `UNDO` command is unavailable until a certain change is made to the edit profile. (For more on edit profiles, see the following section.)

25.2.1 Customizing Your Editor's Environment

There are several commands that you can type at the editor's command line to customize the editor's behavior. (If your cursor is not already at the command line, press the Home key to move it up there.)

Because some programming languages expect everything to be in upper case, the ISPF editor sometimes translates everything you type to upper case. This is annoying when you type a paragraph, press Enter, and the whole paragraph gets converted. If you want the editor to respect a mix of upper and lower case letters, enter

```
caps off
```

at the command line. When writing a program in a language like FORTRAN, which requires upper case letters, entering the command

```
caps on
```

will translate every new line of text to all upper case letters each time you press Enter.

Another setting that can cause confusing behavior is NULLS. You set it on by typing

```
nulls on
```

at the editor's command line and set it off by typing `OFF` instead of `ON` in the same command.

When set to OFF, the editor automatically puts blanks everywhere on a line where you didn't type anything else (between each word and throughout the remainder of the line after the last word) as if you had moved to each of these places and pressed the space bar to insert the blank character. This fills up each line, and causes problems when you want to insert new characters on

a line. As with any text editor, inserted characters move existing characters on the right of the cursor further to the right; if the blank character fills all the space that looks empty, then the editor has nowhere to move the existing characters when you insert new characters. Trying to do so locks the keyboard, and you must press the Reset key to continue. (For more detailed information on inserting new characters, see section 25.4, "Inserting, Deleting, and Typing over Words and Characters.")

If setting NULLS to ON avoids this problem, there must be a disadvantage to setting it ON. Otherwise, the editor wouldn't offer you two different settings, right? Many books describing the ISPF editor describe the following "problem": sometimes you want a space inserted where you didn't actually press the space bar, and with NULLS set to ON, pressing Enter resets the typed line, squeezing out all the places where nothing was typed. If you type "how are you" on a line and use your cursor-right key instead of the space bar for the spaces between the words, pressing Enter turns the phrase into "howareyou."

When editing a simple text data set, it makes more sense to set NULLS to ON. Setting it to OFF only provides an advantage when the ISPF editor is used to fill out a data entry form, in which the screen displays named fields of fixed lengths. In this case, use the Erase EOF (erase to end of field) key to clear any characters in the field from the cursor's position to the end of the field.

To sum up: if you have problems inserting, set NULLS to ON so the editor will not automatically fill in the remainder of a line after the last word with blanks.

Once you've set various parameters to customize your editing environment, many editors provide a way to save these settings for future use in a collection called an edit profile. The ISPF editor does this for you automatically, and it goes a step further than many other editors: it saves different collections of settings based on the TYPE of the data set. This way, if you set CAPS to ON when writing a program in FORTRAN (for example, a data set called `MAINMENU.FORT`), then CAPS will always be ON for any data set you edit with a type of FORT, whether it's called `RPTMENU.FORT`, `SUMRPT.FORT`, or `MAINMENU.FORT`. If you set CAPS to OFF when editing a data set called `MAY6MEMO.TEXT`, it remains set to OFF every time you edit a data set with that type. If you alternately edit TEXT and FORT data sets, you won't have to reset CAPS every time, because the editor remembers how you set CAPS and all the other settings for each data set type.

25.3 Line Commands

In addition to the kind of commands that you enter at the command line at the top of your screen (described in section 25.2.1, "Customizing Your Editor's Environment,") the ISPF editor has another category of commands called "line commands." (These will be familiar to users of the AS/400's SEU editor. Also, CMS users who find that the screen resembles the XEDIT text editor's screen will find that line commands are the ISPF editor's counterpart to XEDIT prefix commands.) You can add, delete, copy, and move lines by entering one- or two-character commands

in the line command area and pressing the Enter key. You can enter line command commands in the column to the left of the `TOP OF DATA` and `BOTTOM OF DATA` lines as well; if you couldn't, it would be pretty tough to add your first line.

You can enter a line command command anywhere on the line command area. For example, if you enter `d2`, the command to delete two lines, on the line command area like this

```
d2000
```

or like this

```
000d2
```

it still works the same. If you enter a command with a number in it, make sure that your cursor is right after the number when you press Enter, because the system treats every character up to the character preceding the cursor as part of the number you meant to type. For example, when you enter your `2d` on the line command area for line number 11400 like this

```
d2400
```

and press Enter with your cursor at the 4, the editor deletes two lines. If your cursor was on the second zero when you pressed Enter, the system would think that you wanted to delete 240 lines!

25.3.1 Adding New Lines

Use the line command `i` (for "insert") to add new lines on which to type. Figure 25.6 shows an example.

```
COMMAND ===>                                                  SCROLL ===> PAGE
***i** ****************************** TOP OF DATA ******************************
****** ***************************** BOTTOM OF DATA *****************************
```

Figure 25.6 Adding the i line command to insert a new blank line.

When you press Enter, the editor adds a new blank line after the line where you entered the command and positions your cursor at the beginning of the new line, waiting for you to type in text, as shown in Figure 25.7.

```
COMMAND ===>                                                  SCROLL ===> PAGE
```

```
****** ***************************** TOP OF DATA ******************************
''''''
****** *****************************_BOTTOM OF DATA ******************************
```

Figure 25.7 The effect of the i command entered in Figure 25.6.

The new line isn't really a part of your data set until you add something there. After you type some text and press Enter, the editor assigns a number to the line, creates a new blank line under it, and positions your cursor at the beginning of the new one. Figure 25.8 shows an example.

```
COMMAND ===>                                                  SCROLL ===> PAGE
****** ***************************** TOP OF DATA ******************************
000100 The shadowy daughter of Urthona stood before red Orc
''''''
****** *****************************_BOTTOM OF DATA ******************************
```

Figure 25.8 New line added by ISPF after you enter text and press Enter.

If the editor converts your entered text to all upper case, but you didn't want it to, you'll need the `CAPS OFF` command to change this. See section 25.2.1, "Customizing Your Editor's Environment," for more information.

Adding a number after the `i` command tells the editor to add that many lines. In Figure 25.9, 3 lines are about to be added; Figure 25.10 shows the effect of this command.

```
COMMAND ===>                                                  SCROLL ===> PAGE
****** ***************************** TOP OF DATA ******************************
0001i3 The shadowy daughter of Urthona stood before red Orc
****** ***************************** BOTTOM OF DATA ******************************
```

Figure 25.9 Adding 3 lines with the i command.

```
COMMAND ===>                                                 SCROLL ===> PAGE
****** ***************************** TOP OF DATA ******************************
000100 The shadowy daughter of Urthona stood before red Orc
'''''' _
'''''' 
'''''' 
****** **************************** BOTTOM OF DATA ****************************
```

Figure 25.10 The effect of the i line command entered shown in Figure 25.9.

If any of these new lines have no text when you press Enter, the editor removes them and only assigns numbers to the new lines with text. Figure 25.11 shows three new blank lines with text entered at only one of them, and Figure 25.12 shows how this looks after pressing Enter.

```
COMMAND ===>                                                 SCROLL ===> PAGE
****** ***************************** TOP OF DATA ******************************
000100 The shadowy daughter of Urthona stood before red Orc
'''''' 
'''''' When fourteen suns had faintly journey'd o'er his dark abode;
'''''' 
****** **************************** BOTTOM OF DATA ****************************
```

Figure 25.11 Three new lines, but with text entered at only one of them.

```
COMMAND ===>                                                 SCROLL ===> PAGE
****** ***************************** TOP OF DATA ******************************
000100 The shadowy daughter of Urthona stood before red Orc
000200 When fourteen suns had faintly journey'd o'er his dark abode;
****** **************************** BOTTOM OF DATA ****************************
```

Figure 25.12 The effect of pressing Enter when viewing Figure 25.11.

If you do want a blank line to remain in your data set, use your space bar to type at least one space there.

25.3.2 Moving Your Cursor Around

Your up, down, left, and right cursor keys move the cursor in the direction in which they point.

The Tab key moves your cursor around more quickly. To move your cursor to the beginning of the previous line, use the Backtab key (with a PC that is emulating a mainframe terminal, press the Shift key and the Tab key simultaneously). If your cursor is on a line of text, Tab and Back-tab jump your cursor to the line command area; if your cursor is on the line command area, pressing either key jumps your cursor to the beginning of the appropriate line.

Some mainframe terminals have a Return key that is separate from the Enter key. It might have an arrow pointing down and then left, or it might say "New Line" on it. When using the editor, pressing this key jumps your cursor to the beginning of the next line.

Use the commands `UP, DOWN, LEFT,` and `RIGHT` at the command line to scroll the text in one of those four directions. For example,

```
down 10
```

scrolls the text down ten lines, and

```
right 20
```

scrolls to the right 20 positions.

25.4 Inserting, Deleting, and Typing over Words and Characters

To delete an individual character, move your cursor there and press your Delete key. On a 3270 terminal, the delete key has a lower case "a" with a proofreader's symbol for deletion: a line through it that forms a loop. When emulating a 3270 terminal, your emulation software probably has your PC's Delete key doing this job.

To type over existing text, just move your cursor where you want the new text and type.

To insert text, move your cursor to the place where you want to insert it and press the Insert key. On a 3270 terminal, this key has the letter "a" with a carat symbol (^) over it. When you press it, a carat symbol should appear at the bottom of your screen to indicate that you are in insert mode. (When emulating a 3270, your cursor may change shape.) Text that you type in moves any text currently on the right of the cursor further to the right.

If the last character gets moved too far to the right, the ISPF editor beeps at you to indicate that

you can't insert more characters on that line. When you first allocate a data set, the LRECL operand (described in section 24.1.7.2, "Allocating New Data Sets") sets its width and the maximum length of any lines.

To return to overstrike mode while using a 3270 terminal, press the key marked "Reset." The carat symbol disappears, and newly typed text will then take the place of the characters at the cursor. (When your keyboard "locks up," or refuses to accept input, the Reset key is also useful for freeing up the keyboard.) On most PCs emulating a 3270, the Insert key does the job of the 3270 keyboard's Insert key and the Escape key serves as the Reset key. Check you emulation program's documentation to make sure.

25.4.1 Duplicating Lines

When duplicating lines, think of it as "repeating" lines. This will make it will be easier to remember the line command: r. Enter r by itself in the line command area and press Enter to make a single copy of a line. For example, if you enter r on the third line in the text shown in Figure 25.13, and then press Enter, you get the result shown in Figure 25.14.

```
COMMAND ===>                                                 SCROLL ===> PAGE
****** ***************************** TOP OF DATA ******************************
000100 The shadowy daughter of Urthona stood before red Orc
000200 When fourteen suns had faintly journey'd o'er his dark abode;
0r0300 His food she brought in iron baskets, his drink in cups of iron
000400 Crown'd with a helmet & dark hair the nameless female stood.
000500 A quiver with its burning stores, a bow like that of night
****** **************************** BOTTOM OF DATA ******************************
```

Figure 25.13 Entering the r line command to repeat a line.

```
COMMAND ===>                                                 SCROLL ===> PAGE
****** ***************************** TOP OF DATA ******************************
000100 The shadowy daughter of Urthona stood before red Orc
000200 When fourteen suns had faintly journey'd o'er his dark abode;
000300 His food she brought in iron baskets, his drink in cups of iron
000400 His food she brought in iron baskets, his drink in cups of iron
000500 Crown'd with a helmet & dark hair the nameless female stood.
000600 A quiver with its burning stores, a bow like that of night
****** **************************** BOTTOM OF DATA ******************************
```

Figure 25.14 The effect of pressing Enter after entering the r line command shown in Figure 25.13.

Notice how the editor renumbered the line (or if applicable, lines) after the new one.

Following the `r` command with a number tells the editor to add that many repetitions of the line with the command. Pressing Enter after entering the command shown on line 200 in Figure 25.15 repeats that line four times, as shown in Figure 25.16.

```
COMMAND ===>                                                  SCROLL ===> PAGE
****** ***************************** TOP OF DATA ****************************
000100 The shadowy daughter of Urthona stood before red Orc
r40200 When fourteen suns had faintly journey'd o'er his dark abode;
000300 His food she brought in iron baskets, his drink in cups of iron
000400 Crown'd with a helmet & dark hair the nameless female stood.
000500 A quiver with its burning stores, a bow like that of night
****** **************************** BOTTOM OF DATA ****************************
```

Figure 25.15 Entering the r line command with a 4 to repeat the line 4 times.

```
COMMAND ===>                                                  SCROLL ===> PAGE
****** ***************************** TOP OF DATA ****************************
000100 The shadowy daughter of Urthona stood before red Orc
000200 When fourteen suns had faintly journey'd o'er his dark abode;
000300 When fourteen suns had faintly journey'd o'er his dark abode;
000400 When fourteen suns had faintly journey'd o'er his dark abode;
000500 When fourteen suns had faintly journey'd o'er his dark abode;
000600 When fourteen suns had faintly journey'd o'er his dark abode;
000700 His food she brought in iron baskets, his drink in cups of iron
000800 Crown'd with a helmet & dark hair the nameless female stood.
000900 A quiver with its burning stores, a bow like that of night
****** **************************** BOTTOM OF DATA ****************************
```

Figure 25.16 The effect of pressing Enter after entering the r4 line command shown in Figure 25.15.

25.4.2 Deleting Lines

Delete lines with the d line command. Entering d by itself in the line command area and then pressing Enter deletes the line where you entered the command; adding a number after the d command deletes a total of that many lines, beginning with the line where you entered the command.

You can also delete multiple lines by indicating the beginning and end of a block to delete. Enter dd at the first and last line of the block to delete and the editor will delete all of those lines, including the ones where you entered the dd commands, when you press Enter. For example, after entering dd at lines 200 and 700 in Figure 25.17, pressing Enter deletes those lines and the lines between them, as shown in Figure 25.18.

```
COMMAND ===>                                                 SCROLL ===> PAGE
****** ***************************** TOP OF DATA ******************************
000100 The shadowy daughter of Urthona stood before red Orc
dd0200 When fourteen suns had faintly journey'd o'er his dark abode;
000300 When fourteen suns had faintly journey'd o'er his dark abode;
000400 When fourteen suns had faintly journey'd o'er his dark abode;
000500 When fourteen suns had faintly journey'd o'er his dark abode;
000600 When fourteen suns had faintly journey'd o'er his dark abode;
0dd700 His food she brought in iron baskets, his drink in cups of iron
000800 Crown'd with a helmet & dark hair the nameless female stood.
000900 A quiver with its burning stores, a bow like that of night
****** **************************** BOTTOM OF DATA ****************************
```

Figure 25.17 Entering the dd line commands to delete 6 lines.

```
COMMAND ===>                                                 SCROLL ===> PAGE
****** ***************************** TOP OF DATA ******************************
000100 The shadowy daughter of Urthona stood before red Orc
000200 Crown'd with a helmet & dark hair the nameless female stood.
000300 A quiver with its burning stores, a bow like that of night
****** **************************** BOTTOM OF DATA ****************************
```

Figure 25.18 The effect of pressing Enter after entering the dd line commands shown in Figure 25.17.

You will find this particularly handy when you want to delete a block that doesn't begin and end on the same screen, because the alternative (counting the number of lines so that you can put a number after a single d) is a lot of trouble.

25.4.3 Copying Lines

Copying is similar to deletion except that you use the letter c to indicate the line or lines to copy and you must indicate a destination for the copied text. The ISPF editor gives you three options for indicating the text to copy:

- Enter a single c in a line's command area if you only need to copy that one line.
- Enter a single c followed by a number to indicate how many lines to copy.
- Enter cc at the first and last lines of the block to copy.

In addition to indicating the line or lines to copy, you must indicate where to copy them. Two line commands make this possible:

b	When Enter is pressed, copy the block to the line before this one.
a	When Enter is pressed, copy the block to the line after this one.

In Figure 25.19, Joe User is about to copy the second, third, and fourth lines above the first line, to the beginning of the data set. Figure 25.20 shows the effect of pressing Enter after he enters these commands.

```
COMMAND ===>                                                 SCROLL ===> PAGE
****** ***************************** TOP OF DATA ******************************
0b0100 The shadowy daughter of Urthona stood before red Orc
cc0200 When fourteen suns had faintly journey'd o'er his dark abode;
000300 His food she brought in iron baskets, his drink in cups of iron
0cc400 Crown'd with a helmet & dark hair the nameless female stood.
000500 A quiver with its burning stores, a bow like that of night
****** *************************** BOTTOM OF DATA ***************************
```

Figure 25.19 Entering the cc line commands to copy 3 lines above the first line.

```
COMMAND ===>                                                  SCROLL ===> PAGE
****** ***************************** TOP OF DATA ****************************
000100 When fourteen suns had faintly journey'd o'er his dark abode;
000200 His food she brought in iron baskets, his drink in cups of iron
000300 Crown'd with a helmet & dark hair the nameless female stood.
000400 The shadowy daughter of Urthona stood before red Orc
000500 When fourteen suns had faintly journey'd o'er his dark abode;
000600 His food she brought in iron baskets, his drink in cups of iron
000700 Crown'd with a helmet & dark hair the nameless female stood.
000800 A quiver with its burning stores, a bow like that of night
****** **************************** BOTTOM OF DATA ****************************
```

Figure 25.20 The effect of pressing Enter after entering the cc line commands shown in Figure 25.19.

25.4.4 Moving Lines

Moving is similar to copying, except that after pressing Enter, the original lines are no longer there—they're at their new location. As with copying, there are three ways to specify the block to move, but these use the letter `m`:

- Enter a single `m` in a line's command area if you only need to move that one line.
- Enter a single `m` followed by a number to indicate how many lines to move.
- Enter `mm` at the first and last lines of the block to move.

To specify the destination of the block to move, use the letters `b` or `a` the same way you do to specify the destination of a block to copy.

25.5 Searching for Text

If you think of searching for text in the ISPF editor as "finding" it, you'll remember the command: the letter "F" entered at the command line. Figure 25.21 shows this command entered to find the word "his," and Figure 25.22 shows the result.

```
EDIT ---- JOEUSER.BLAKE.TEXT --------------------------------- COLUMNS 001 072
COMMAND ===> f his                                            SCROLL ===> PAGE
```

```
****** ***************************** TOP OF DATA ****************************
000001 The shadowy daughter of Urthona stood before red Orc
000002 When fourteen suns had faintly journey'd o'er his dark abode;
000003 His food she brought in iron baskets, his drink in cups of iron;
000004 Crown'd with a helmet & dark hair the nameless female stood.
000005 A quiver with its burning stores, a bow like that of night
****** **************************** BOTTOM OF DATA ****************************
```

Figure 25.21 Using the find command to look for the word "his."

The cursor jumps to the first occurrence, and a message in the upper-right of the screen shows the characters that it found.

If your search target has a blank in it, (for example, if you want to search for the string "his drink") enclose the string in apostrophes, like this:

```
f 'his drink'
```

Note at the bottom of the screen in Figure 25.22 how the F5 key means RFIND, or "repeat find." Press it, and the cursor jumps to the next occurrence of the string "his." Figure 25.23 shows the effect of pressing F5 when viewing the screen shown in Figure 25.22.

```
EDIT ---- JOEUSER.BLAKE.TEXT ------------------------------ CHARS 'his' FOUND
COMMAND ===>                                                  SCROLL ===> PAGE
****** ***************************** TOP OF DATA ****************************
000001 The shadowy daughter of Urthona stood before red Orc
000002 When fourteen suns had faintly journey'd o'er his dark abode;
000003 His food she brought in iron baskets, his drink in cups of iron;
000004 Crown'd with a helmet & dark hair the nameless female stood.
000005 A quiver with its burning stores, a bow like that of night
****** **************************** BOTTOM OF DATA ****************************









 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

Figure 25.22 The effect of pressing Enter after entering the find command shown in Figure 25.21.

```
EDIT ---- JOEUSER.BLAKE.TEXT ------------------------------ CHARS 'His' FOUND
COMMAND ===>                                                 SCROLL ===> PAGE
****** ***************************** TOP OF DATA ****************************
000001 The shadowy daughter of Urthona stood before red Orc
000002 When fourteen suns had faintly journey'd o'er his dark abode;
000003 His food she brought in iron baskets, his drink in cups of iron;
000004 Crown'd with a helmet & dark hair the nameless female stood.
000005 A quiver with its burning stores, a bow like that of night
****** **************************** BOTTOM OF DATA ****************************
```

Figure 25.23 Finding "his" a second time by pressing the F5 key.

What happens if the search target isn't found? Figure 25.24 shows what happens after pressing F5 a third time, when there are no more occurrences. As the message in the upper-right of Figure 25.24 shows, it searched from the cursor location to the bottom of the data, and didn't find the string.

```
EDIT ---- JOEUSER.BLAKE.TEXT ------------------------ *BOTTOM OF DATA REACHED*
COMMAND ===>                                                 SCROLL ===> PAGE
****** ***************************** TOP OF DATA ****************************
000001 The shadowy daughter of Urthona stood before red Orc
000002 When fourteen suns had faintly journey'd o'er his dark abode;
000003 His food she brought in iron baskets, his drink in cups of iron;
000004 Crown'd with a helmet & dark hair the nameless female stood.
000005 A quiver with its burning stores, a bow like that of night
****** **************************** BOTTOM OF DATA ****************************
```

Figure 25.24 The result of an unsuccessful search for a string.

Searching is not case-sensitive; although we were searching for "his," it also found "His." To do an exact-case search, put the letter "C" just before your search target. (Whether the target has a

blank in it or not, you need to enclose it in apostrophes to separate it from the "C.") The following shows an example:

```
f c'his'
```

The `f` command always searches from the cursor location, unless you tell it otherwise with the word "first." Typing the following

```
f first his
```

tells the system, "search for the word 'his' starting at the first line of the data set."

Combining these features is no problem; entering

```
f first c'his'
```

tells the editor to do an exact-case search for the string "his" starting at the beginning of the data set.

25.6 Saving Your Changes

To save your edits, enter SAVE at the command line and press Enter. You can then continue editing. To abort any edits made since the last time you saved your work, enter CANCEL at the command line. The editor returns you to the Edit Entry Panel where you first entered the name of the data set to edit.

The next section shows a shortcut for saving your work and quitting all at once.

25.7 Quitting the ISPF Editor

As the bottom of the screen shows, F3 is the END key. Pressing this saves your work and returns you to the Edit Entry Panel, where you can enter the name of another data set to edit or quit back to the main ISPF menu.

If you are done editing your data set and know that you don't want to edit any other data sets, press F4, the RETURN key (not to be confused with the Return key on the right side of your keyboard). This saves your data set, quits out of the editor, and skips the Edit Entry Panel, returning you directly to the ISPF main menu. To exit this menu and return to the TSO prompt, select the EXIT menu choice.

25.7.1 On-line Help in the Editor

While using the editor, you can access on-line help in two ways:

- Typing HELP at the command line.
- Pressing F1.

Either way, the system displays the main help menu for the editor, as shown in Figure 25.25.

```
TUTORIAL ------------------------- EDIT ---------------------------- TUTORIAL
OPTION  ===>

                     ------------------------------------
                     |              EDIT                |
                     ------------------------------------

  Edit allows you to create or change source data.

The following topics are presented in sequence, or may be selected by number:
  0 - General introduction              8 - Display modes (CAPS/HEX/NULLS)
  1 - Types of data sets                9 - Tabbing (hardware/software/logical)
  2 - Edit entry panel                 10 - Automatic recovery
  3 - SCLM edit entry panel            11 - Edit profiles
  4 - Member selection list            12 - Edit line commands
  5 - Display screen format            13 - Edit primary commands
  6 - Scrolling data                   14 - Labels and line ranges
  7 - Sequence numbering               15 - Ending an edit session

The following topics will be presented only if selected by number:
 16 - Interaction between LMF and SCLM
 17 - Edit models
 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

Figure 25.25 Main help menu for ISPF editor.

As with any other help screen, pressing F1 again gives you help about the available help—in other words, it tells you how to use the help system. Using it is pretty simple: at the OPTION ===> prompt, enter the number of the menu choice about which you want to learn more. Help about a complicated topic may lead to another menu that lets you choose a subtopic. When you are done, press F3 or F4 to return to the editing screen.

25.8 TSO's EDIT Text Editor

If you can't use the ISPF editor, you can still use the TSO EDIT text editor. Some PC users who used DOS before release 5.0 endured a similar situation: when they needed a text editor but lacked a decent one, they had to use the dreaded EDLIN, which came with the operating system.

As a matter of fact, anyone with experience with EDLIN will find the TSO EDIT editor very familiar. It has a command mode and an input mode, and everything you enter and everything that the system displays scrolls up the screen a line a time.

25.8.1 Starting the Editor

In its simplest form, you can tell the editor that you want to edit a particular data set by entering `EDIT` followed by the data set name. However, as you will see in section 25.8.16, "On-line Help and the TSO Editor," typing `HELP EDIT` at the TSO prompt reveals quite a few possible operands that you can add. Many of these tell the editor what kind of data set you are editing—for example, whether it's a text data set or source code for a FORTRAN or COBOL program—if the editor can't figure it out from the data set name.

For example, if you enter

```
edit blake.text
```

it knows that you are editing a data set consisting of simple text. If, on the other hand, you enter

```
edit jan3bc.memo
```

then TSO asks you to

```
ENTER DATA SET TYPE-
```

and you enter "text" in response to this. If the data set type is not obvious from the low-level qualifier, you can add it as an operand, like this,

```
edit jan3bc.memo text
```

and the editor doesn't prompt you to enter the type of the data set.

25.8.2 Creating a New Data Set

The TSO editor does have one significant advantage over the ISPF editor: if you tell it to edit a nonexistent data set, it allocates and creates the data set for you instead of just telling you that it doesn't exist. For example, if you enter

```
edit test.text
```

the TSO editor responds with

```
DATA SET OR MEMBER NOT FOUND, ASSUMED TO BE NEW
```

Because the details of the allocation vary from installation to installation, try creating a data set this way and then using the `LISTDS` command to learn the allocation parameters used. (Section 24.1.7.4, "Finding Out a Data Set's Allocation Status," explains more about the `LISTDS` com-

mand.)

After creating a new data set, the editor displays the line number for the data set's first line and puts you in input mode. The following section explains more about using input mode.

25.8.3 Line Numbering and the EDIT Editor

Data sets may be numbered or unnumbered, which affects how you use the EDIT editor. Like the programs that enable you to write programs in BASIC on older computers, you can use these numbers to indicate which lines should be affected by each command. Also, when editing a numbered data set, you can tell more easily when you are in input mode, because EDIT displays the number of the line being added.

25.8.4 Input Mode and Edit Mode

The EDIT program is always waiting for you to type something in. While in input mode, (see section 25.8.7, "Adding New Lines," for more on input mode) each line you type becomes a new line of the data set. In edit mode, the editor interprets each line you type as a command ("command mode" would have been a better name than "edit mode") and executes it, if possible, or gives you an error message if not. The two classic mistakes are the following:

- Entering a command when you're really in input mode, which adds a line to your data set consisting of the command you thought you were entering.
- Entering what you think is a new line of text when it's really in edit mode, so that the editor tries to interpret the text as a command and tells you something along the lines of "'Dear Mom' is an unrecognized editor command."

When editing a numbered data set, this won't happen too easily, because the editor displays the number of the line you are entering when you are in input mode. In the following example, after Joe User entered each line and pressed Enter, the next line number appeared, waiting for a new line. For the third line, instead of entering anything, he pressed Enter immediately. This told the editor that he was done with input mode, so it went back to edit mode.

```
  INPUT
  00010 Lo, a shadow of horror is risen
  00020 In Eternity! Unknown, unprolific?
  00030
  EDIT
 Self-closd, all-repelling: what Demon
  INVALID SUBCOMMAND
 input
  INPUT
  00030 Self-closd, all-repelling: what Demon
  00040
```

Joe then typed the third line, but the editor thought he was typing a command (a subcommand,

actually, since `EDIT` is the command to start the editor, and commands within the editor are considered subcommands). It displayed an error message saying that it couldn't find a subcommand called "SELF-CLOSD." Joe then entered the command `INPUT` to return to input mode, the line number appeared, waiting for his text, and he entered it without a problem.

When editing an unnumbered data set, the lack of line numbers requires you to pay closer attention to which mode you're in. The following example shows the same steps as the previous one, only with an unnumbered data set.

```
  INPUT
 Lo, a shadow of horror is risen
 In Eternity!  Unknown, unprolific?
  EDIT
 Self-closd, all-repelling: what Demon
  INVALID SUBCOMMAND
 input
  INPUT
 Self-closd, all-repelling: what Demon
  EDIT
  INPUT
  EDIT
```

Although no line number appeared after Joe typed the second line, pressing Enter without typing anything on the third line produced the same effect: it put him in edit mode. The same "Self-closd" mistake caused the same error message, and entering the `INPUT` command put him back into input mode.

After inputting the "Self-closd" line properly, Joe pressed the Enter key a few more times. Note how it shifted him between edit and input mode as long as he didn't enter anything at the command line. This shows that pressing Enter once or twice is a good way to switch between modes and to double-check which mode you're in.

When in input mode or edit mode, you can use all the same keys that you use to edit any TSO command line: cursor left and right, insert, and delete.

25.8.5 Displaying the Data Set's Current Contents

Enter `LIST` to see the contents of the data set that you are editing. (This command is easy to remember because it's the same as the TSO command to do the same thing.) If it has too many lines to fit on the screen, you can enter beginning and ending line numbers. For example, if you enter

```
list 20 30
```

while editing the numbered data set shown above, the editor responds with this:

```
  00020 In Eternity! Unknown, unprolific?
  00030 Self-closd, all-repelling: what Demon
```

If you try this with an unnumbered data set, the editor responds with an error message telling you that it can't list lines 20 and 30 because the data set is unnumbered. When editing an unnumbered data set, you must list either the whole data set or just the current line.

25.8.6 The Current Line

At any given time, the editor treats one line of your data set as the "current" line. It's important to be aware of which line is current because certain commands (like `DELETE`) have a default action of only acting on the current line.

With a command whose default action is to act on the whole data set, like `LIST`, you can add a line number to refer to a specific line or an asterisk to refer to the current line. For example, to list just the current line, enter this:

```
list *
```

You'll probably enter this command often, because it's the best way to check on which line is current.

The commands `UP` and `DOWN` let you change the current line. For example, if the fifth line is current, entering

```
up 2
```

makes the third line current. If you enter a number that's too big, like

```
up 1000
```

you won't get an error message; the editor just makes the first line current. `DOWN` works the same way: entering it with a number greater than the number of lines in your data set makes the last line current. (For a simpler way to jump to the top or bottom of your data set, enter the commands `TOP` or `BOTTOM`.)

Be careful when using the `LIST` command to view the whole data set. After viewing the data set, the last listed line becomes the current one regardless of which line was current before you entered the command. Try this by entering `LIST` alone, pressing Enter, and then entering `LIST *` to see which is the new current line.

25.8.7 Adding New Lines

The examples in section 25.8.4, "Input Mode and Edit Mode," show how to use the `INPUT` command to add new lines. Remember, once in input mode, you remain there until you enter a line with nothing on it by pressing Enter when the editor is waiting for you to type in a new line.

If you enter `INPUT` with no operands in edit mode, the editor puts you in edit mode and appends

each new line that you enter to the end of your data set. Adding new lines between existing lines depends on whether you are editing a numbered or unnumbered data set.

When editing a numbered data set, you have a little more flexibility for entering new lines. The simplest way is to enter the new line number followed by the new line while in edit mode.

If you enter

```
25 (You gotta love this Blake guy)
```

and press the Enter key, the editor adds the line and returns you to edit mode. If you then enter the `LIST` command to see how the data set looks, you'll see this:

```
  00010 Lo, a shadow of horror is risen
  00020 In Eternity! Unknown, unprolific?
  00025 (You gotta love this Blake guy)
  00030 Self-closd, all-repelling: what Demon
```

If you want to enter more than one new line, use the `INPUT` command, but with two operands: the number of the line after which you want to begin and the increment to increase the line number. If you omit the second operand, the editor assumes a default of 10. This default value could cause a problem; for the data set above, if you enter

```
INPUT 20
```

then the new line after line 20 would be line 30. As it already has a line 30, it gives you an error message saying

```
INPUT TERMINATED, NEXT LINE NUMBER IS 30
```

If, on the other hand, you entered

```
INPUT 20 2
```

then you would be in input mode to enter line 22, then 24, and so forth until you entered line 28. After 28, you would get the same error message, because 30 comes after 28. For subsequent `INPUT` commands in that editing session, it will assume an increment of 2 until you reset it with a similar command.

After squeezing new lines between the existing ones, you might want to occasionally set all the line numbers back to multiples of 10 by entering the command `RENUM` in Edit mode.

To input new lines between existing lines in an unnumbered data set, use the asterisk denoting "current line" with the `INPUT` command. First, use the `UP, DOWN,` and `LIST *` commands to find the line just before the place where you want to insert new lines. For example, let's say Joe User is editing the following data set.

```
  The shadowy daughter of Urthona stood before red Orc
  When fourteen suns had faintly journey'd o'er his dark abode;
  His food she brought in iron baskets, his drink in cups of iron;
  Crown'd with a helmet & dark hair the nameless female stood.
```

He makes the second line current and enters `LIST *` to make sure it's current. It displays by itself, so he knows that he's in the right place. He enters

```
INPUT *
```

and then, underneath it,

```
(Fourteen suns?  What a wildman!)
```

and then presses Enter twice to show that he only wants to input that one line. When he enters `LIST` again to see the whole data set, he sees this:

```
  The shadowy daughter of Urthona stood before red Orc
  When fourteen suns had faintly journey'd o'er his dark abode;
  (Fourteen suns?  What a wildman!)
  His food she brought in iron baskets, his drink in cups of iron;
  Crown'd with a helmet & dark hair the nameless female stood.
```

25.8.8 Editing Existing Lines

To edit a string in the current line, use the `CHANGE` command. Its operands are the string to replace and the new one to put there, both enclosed in apostrophes. For example, if the current line is

```
 His food she brought in iron baskets, his drink in cups of iron;
```

and you enter

```
change 'cups of iron' 'a plaid Thermos'
```

then entering `LIST *` shows that the line now reads:

```
 His food she brought in iron baskets, his drink in a plaid Thermos;
```

To replace the entire current line, you can enter an asterisk followed by the new line while in edit mode, like this:

```
* Here is text for a new line
```

Both of these tricks for editing existing lines work for numbered and unnumbered data sets. You can take the last trick a step further for numbered data sets; if the line you want to replace is not the current line, but you know its number, enter its number followed by the new contents of the line. (Replacing the current line by entering an asterisk and the new text is actually a variation on this; remember, the asterisk acts as a substitute for the current line's line number.)

25.8.9 Deleting Lines

Entering `DELETE` by itself deletes the current line. To delete a range of lines from a numbered

data set, enter DELETE followed by the line numbers of the first and last lines to delete. For example, entering

```
DELETE 40 90
```

deletes lines 40, 90, and all the lines between them.

To delete multiple lines from an unnumbered data set, first make the first line to delete the current line. Then, enter the command to delete the current line, followed by the total number of lines you want to delete. For example, entering

```
DELETE * 5
```

will delete the current line and the four lines following it.

25.8.10 Copying Lines

Whether you are copying lines in a numbered or unnumbered data set, the syntax for specifying the lines to copy is just like the syntax for indicating lines to delete. The COPY command takes one more parameter: a number telling you where to put the copy.

To copy lines in a numbered data set, you need three parameters after the COPY command: the first of the lines to copy, the last of the lines to copy, and the line number where the copy should start. For example, the command

```
COPY 20 50 95
```

means "make a copy of lines 20 through 50 and start it at line 95." If line 95 exists, it starts the copy right after it.

With an unnumbered data set, you must first make the first of the lines to copy your current line. As with deleting lines in an unnumbered data set, your first parameter is the asterisk and the second is the number of lines you want to copy. The third parameter is the position of the line just before where you want to put your copy. If you picture the data set being numbered in increments of one (as opposed to the increments of 10 more popular in numbered data sets), then this would be the line number of the target line. For example, in the following data set,

```
The shadowy daughter of Urthona stood before red Orc
When fourteen suns had faintly journey'd o'er his dark abode;
His food she brought in iron baskets, his drink in cups of iron;
Crown'd with a helmet & dark hair the nameless female stood.
A quiver with its burning stores, a bow like that of night
When pestilence is shot from heaven--no other arms she need:
```

we want to copy the lines beginning "Crown'd with a helmet," "A quiver with" and "When pestilence" and put the copy after the line "When fourteen suns." First, make the line "Crown'd with a helmet" the first line; then, the command

```
COPY * 3 2
```

tells the editor "copy three lines, beginning with the current one (represented by the asterisk) and put the copy after the second line."

25.8.11 Duplicating Lines

Duplicating the current line of your data set involves using the COPY command with the asterisk. Section 25.8.10, "Copying Lines," shows that when you copy a line or lines, you indicate the lines to copy and the line after which to put the copy. Using the asterisk to indicate the current line as both of these parameters, you can type

```
COPY * *
```

to tell the editor, "Make a copy of the current line, and put the copy right after itself."

25.8.12 Moving Lines

Move lines with the MOVE command. For numbered and unnumbered data sets, the syntax is the same as when copying. The only difference is, after you've executed the command, the original won't exist anymore—it will be in the location specified by the third parameter of either of the following commands:

```
MOVE firstline lastline destination
```

for numbered data sets or

```
MOVE * lines destination
```

for unnumbered data sets.

25.8.13 Searching for Text

The FIND command does a case-sensitive search for the string entered as its operand. Enclose the string in apostrophes. For example, entering

```
find 'helmet'
```

when the first of the earlier passage's lines is current makes the fourth line the current line because it has the phrase "Crown'd with a helmet." Entering

```
find 'potrzebie'
```

causes the editor to look for, but not find this string, and it gives you the message TEXT NOT FOUND. Entering find 'helmet' twice in a row displays the TEXT NOT FOUND message the second time, because the search always begins at the current line and the passage has no more occurrences of the word "helmet" after it finds it the first time.

If you want to search for something with an apostrophe in it, you can enclose your search string in quotation marks. For example, when searching from the top of the passage, you could enter

```
find "o'er"
```

to find the line with the phrase "journey'd o'er his dark abode." That line then becomes the current line.

25.8.14 Saving Your Changes

To save your changes, simply enter SAVE while in edit mode. See the next section for information on quitting and saving in one command.

25.8.15 Quitting the TSO Editor

The END command tells the editor that you want to return to the TSO prompt. If you have any unsaved changes, it tells you:

```
NOTHING SAVED
ENTER SAVE OR END-
```

At this point, enter END to quit without saving your changes or SAVE to save them. If you enter SAVE, it returns you to edit mode, waiting for another command. (You'll probably want to enter END again; this time, it won't give you the message about unsaved changes.)

To save your work and quit with one command, enter the following:

```
END SAVE
```

The editor saves your data set and return you to the TSO prompt. To quit without saving your changes, enter this:

```
END NOSAVE
```

25.8.16 On-line Help and the TSO Editor

Entering HELP EDIT at the TSO prompt gives you several screens of information about the EDIT program. Most of it is information about the command-line parameters that you can add when starting up the EDIT program, and not help about actions you can take within the EDIT program, but this is what the TSO help does: it tells you about the syntax of commands typed at the TSO prompt.

The first part of the information displayed by HELP EDIT is a section called SUBCOMMANDS that lists the commands that you can use in edit mode. Including the ones described in this section, the editor has over 30 of these subcommands. One of them is HELP, which shows

that you can get help from within the editor.

If you enter HELP by itself in the editor's edit mode, it lists the subcommands and tells you this:

```
FOR MORE INFORMATION ENTER HELP SUBCOMMANDNAME OR HELP HELP
```

In other words, you can enter HELP followed by one of the subcommand names to find out more about that particular subcommand. This includes the HELP subcommand itself—entering HELP HELP tells you more about the use of the HELP command within the text editor.

Chapter 26 Using an MVS System

26.1 Printing Data Sets

There are several ways to print data sets. Many involve printing the result of a batch job; in a case like this, you add certain commands to the JCL part of the job telling it to route output to a particular printer.

The simplest way to print a data set uses the `PRINTDS` or `DSPRINT` programs, whichever is available on your system. At their most basic level, both of these commands require the same information from you:

- The data set you want to print.
- The name of the printer to which you want to send the data set.

The main difference between the two is that `DSPRINT` assumes that the first operand is the data set to print and the second identifies the printer. With `PRINTDS`, you can enter them in any order, but you must include the words `DATASET` and `DEST` (for "destination"), putting the appropriate information in parentheses.

For example, to use `PRINTDS` to send the data set `BLAKE.TEXT` to the printer called `ACCTNG`, enter this:

```
printds dataset(blake.text) dest(acctng)
```

With the `DSPRINT` command, you enter this:

```
dsprint blake.text acctng
```

Ask about the name of the closest mainframe printer, which each site's system administrator assigns.

Other differences between `PRINTDS` and `DSPRINT` lie in the additional operands available to control their print output. One that you will find useful with both is `NONUM`. It tells the command, "If the data set is numbered, don't print the numbers." The TSO on-line help tells you more about other operands for these commands.

26.2 Command Files

A CLIST (pronounced "see-list") is a program written in the TSO command language. In other words, it's a data set consisting of a series of TSO commands. CLISTs can be complex, but simple ones consisting of the TSO commands that you already know can be quite useful.

CLISTs are usually stored as members of a partitioned data set called either CLIST or whatever.CLIST, where whatever is a name you supply.

There are two ways to execute a particular CLIST. The first uses the EXEC command. To execute a CLIST in MYCLISTS.CLIST with a member name of MYTEST, enter the following:

```
exec myclists(mytest)
```

If you stored MYTEST in a partitioned data set called only CLIST, execute it with this command:

```
exec (mytest)
```

You can execute a CLIST by just entering its name. To execute MYTEST, type

```
mytest
```

at the TSO prompt. How will the system know the partitioned data set in which to find it? It looks for a special ddname of SYSPROC. This means that you must allocate the partitioned data set that holds your CLISTs with a ddname of SYSPROC by entering a command like this:

```
allocate dataset(myclists.clist) ddname(sysproc)
```

If you want to let others use the MYCLISTS CLISTs while you use them, add the SHR parameter to this allocation command.

People often use CLISTs to allocate several data sets at once. Many application programs require you to do several allocations before starting them up; creating a new member of a CLIST partitioned data set with the required ALLOCATE commands means that you only have to type that member's name to do all those allocations.

For example, let's say the MVS version of the UpRiteBase database program requires you to allocate the partitioned data set holding your data files with the ddname URDATA and the PDS holding the command procedures to use with that data as URPROC. You store your data as members of the MYDATA.DAT PDS and your procedures in the MYPROCS.PRC PDS, and you start UpRiteBase by entering URBASE. You can automate the allocations and the URBASE part by creating a new member of the partitioned data set holding your CLISTs called UR with the following commands in it:

```
/* UR: allocate procedure and database files for UpRiteBase, */
/*     then start up UpRiteBase.  2/23/94 Joe User           */
FREE DD(URPROC URBASE)
ALLOCATE DS(MYDATA.PRC) DD(URPROC) SHR
ALLOCATE DS(MYPROCS.DAT) DD(URBASE) SHR
URBASE
```

Note three things about this CLIST:

- The first two lines begin with `/*` and end with `*/`. TSO ignores everything between the asterisks, so they are used to demarcate comments describing the purpose of the CLIST. Among other things, the comments shows that this member of the CLISTs PDS is called `UR`.

- The `FREE` command frees up the ddnames before the subsequent lines allocate the data sets, just in case these ddnames are already allocated.

- The CLIST finishes by starting up UpRiteBase for you with the `URBASE` command.

Using this CLIST, you only have to type "UR" and TSO performs all these tasks for you. In addition to TSO commands, CLISTs can contain REXX commands. In their attempt to make a command language that is portable from one operating system to another, IBM made the basic rules of REXX the same on MVS systems as on CMS systems. See section 21.1 (VM/CMS "Command Files") for more on REXX.

26.2.1 The Automatic Logon Command File

When you log on to your MVS account, TSO looks for a sequential data set named `LOGON.CLIST`. If it finds it, it automatically executes it. You will find this useful for automating actions that you want performed every time you log on. For example, if you keep your other CLISTs in a partitioned data set called `MYCLISTS.CLIST`, you could put the line

```
ALLOCATE DATASET(MYCLISTS.CLIST) DDNAME(SYSPROC)
```

in your `LOGON.CLIST` data set so that you could then invoke any one of your CLISTs by typing its name at the TSO prompt. A `LOGON.CLIST` that ends with the command `ISPF` automatically starts up ISPF for you when you log on.

26.3 Communicating with Other Users

MVS offers no built-in mail facility. Many companies buy mail programs to run on their MVS system; check to see whether your site has one installed.

You can use the `SEND` command to send a one-line message to someone else's terminal. (Most operating systems have some equivalent command; it seems a little more important in MVS because of the lack of a built-in mail facility.) To send the message "Are we still on for lunch?" to Mary Jones' MJONES user ID, Joe User would enter

```
send 'Are we still on for lunch?' user(mjones)
```

If Mary is logged on, the message appears on her screen followed by the user ID of the person who sent it:

```
Are we still on for lunch? JOEUSER
```

If Mary is not logged on when Joe enters the SEND command asking her about lunch, he will see a message like this:

```
 USER(S) MJONES  NOT LOGGED ON OR TERMINAL DISCONNECTED, MESSAGE CANCELLED
```

He could then type the command again, adding the SAVE operand at the end. This tells TSO to display the message the next time Mary logs on. But an even better alternative exists: he could have added the operand LOGON when he first typed the command:

```
send 'Are we still on for lunch?' user(mjones) logon
```

This tells the system, "If Mary is logged on now, display this on her screen right away; otherwise, display it the next time she logs on."

You may have noticed that the message is enclosed in apostrophes. If you want one displayed in your message, enter two where you want it to appear. For example, entering

```
send 'Looks like I''ll have to work through lunch' user(mjones)
```

displays the following on Mary's screen:

```
Looks like I'll have to work through lunch JOEUSER
```

Another way to send a message to appear on someone's screen is with the MSGDATASET option of the TRANSMIT command. For more on this, see section 26.3.1, "Sending Files."

26.3.1 Sending Files

Data sets are sent from one ID to another with the TRANSMIT command and received with the RECEIVE command. You have two options when you send a data set with TRANSMIT:

- If you send a data set as a regular data set, RECEIVE stores it with the recipient's other data sets.
- If you send it as a message, RECEIVE displays it on the screen for them.

It's not unusual to send two data sets this way simultaneously. One might be a data file, and the other, a memo about the first.

The simplest form of TRANSMIT just includes the name given to the recipient's system, the user ID, and the data set to send. To send the BLAKE.TEXT data set to the MJONES ID on an MVS system called SATURN, the command is:

```
transmit saturn.mjones dataset(blake.text)
```

After you press the Enter key, TSO displays a message telling you whether it sent the data set

successfully and if not, why.

To send a message stored in the data set YOMARY.TEXT along with BLAKE.TEXT so that the RECEIVE command displays the message on Mary's screen when she uses RECEIVE to pull in BLAKE.TEXT, enter this:

```
transmit saturn.mjones dataset(blake.text) msgdataset(yomary.text)
```

You can't store the message in just any data set; it must meet the following requirements:

- It must be either a sequential data set or a specific member of a partitioned data set.
- It must be allocated with a fixed, blocked record format: RECFM(F,B).
- It must have a logical record length of 80: LRECL(80).

If you try to send a data set that doesn't meet these requirements, you'll get an error message saying that the message data set "contains attributes that are not valid."

The data set sent with the DATASET operand doesn't have to be sequential, but part of this command's flexibility may become a limitation when you try to send a partitioned data set. The flexibility lies in the capability to send data sets to other, non-MVS systems: if your MVS system's company or university site has other systems connected to it through a network, you can send data sets to them using the same syntax. For example, if you want to send the BLAKE.TEXT data set to Mary's MJONES ID on a VAX system called NEPTUNE, enter this:

```
transmit neptune.mjones dataset(blake.text)
```

Although TRANSMIT can send partitioned data sets as well as sequential data sets, remember that the concept of a PDS is peculiar to MVS—sending one to a machine running another operating system doesn't mean that the other operating system can do anything with it.

You can, however, send a particular member of a PDS and instruct TRANSMIT to treat it as a sequential data set by adding the SEQ operand. For example, the following command sends the TUTORIAL member of the REXX.CLIST PDS:

```
transmit neptune.mjones dataset(rexx.clist(tutorial)) seq
```

What happens if you just enter TRANSMIT and a user ID, without specifying anything to send? The TRANSMIT command assumes you want to send a message, not an existing data set. After prompting you to enter your message's recipient it displays an editor to allow you to enter that message, as shown in Figure 26.1.

```
DATA FOR  SATURN.MJONES
  00001
  00002
```

```
00003
00004
00005
00006
00007
00008
00009
00010
00011
00012
00013
00014
00015
00016
00017
00018
00019
00020
00021
00022
PF3 = FINISHED   |   PF7 = BACKWARD,   |   PF8 = FORWARD   |   PA1 = ABORT
```

Figure 26.1 Data entry screen for a message sent with TRANSMIT.

The editor is a simplified version of the ISPF editor. As the bottom line shows, PF7 and PF8 scroll backward and forward a page at a time if you enter more than 22 lines of text, PF3 tells the editor that you are finished, and PA1 indicates that you want to abort this message.

26.3.2 Receiving Mail and Data Sets

The `TRANSMIT` command doesn't automatically add data sets to the recipient's disk space. They wait in a special area until they are pulled in with the `RECEIVE` command. If you enter `RECEIVE` when no data sets or messages are waiting for you to pull them in, TSO displays a message similar to this:

```
You have no messages or data sets to receive.
```

If there are one or more data sets waiting for you, it displays either the first message sent, a prompt asking you about a data set that was sent to you, or both if the message was sent with the data set. The prompt tells you the name of the data set, who sent it, and what system they sent it from. For example, let's say Mary Jones sent the data set `STATS.TEXT` to the JOEUSER ID from her ID on an MVS system identified as SATURN. When Joe enters the `RECEIVE` command, the system responds with a message like this:

```
Dataset MJONES.STATS.TEXT from MJONES on SATURN
```

```
Enter restore parameters or 'DELETE' or 'END' +
```

Although the prompt only offers two choices to respond with, you actually have several. Pressing Enter without entering anything is the equivalent of typing the word RESTORE as a response. It means "put this data set, as is, with the rest of my data sets." If Mary named it MJONES.STATS.TEXT before she sent it and your ID is JOEUSER, RECEIVE stores it as JOEUSER.STATS.TEXT.

Responding with DELETE means "I don't want that data set; delete it." Responding with END tells the RECEIVE program to leave off where it was and return to the TSO prompt. The next time you start up RECEIVE, it gives you the same prompt about the data set where you left off.

To restore the data set with a name that differs from the one shown in the message, enter the word "DATASET" followed by parentheses surrounding the new name. For example, if the RECEIVE program displayed the message above and you want to save the STATS.TEXT data set as STATSFEB.TEXT, respond by entering this:

```
dataset(statsfeb.text)
```

Of course, since you probably don't know the names of all your current data sets, you may not notice that an incoming data set has the same name as one already in your collection. If you press Enter to tell RECEIVE to recover the data set and you already have one with that name, RECEIVE displays a message like this:

```
Dataset 'JOEUSER.STATS.TEXT' already exists.  Reply 'R' to replace it. +
```

As the message tells you, entering an "R" and pressing Enter replaces the existing STATS.TEXT data set with the one being recovered. You can also respond with an alternate name, the same way you can respond to the "Enter restore parameters" prompt with dataset(new.dsname). When you press Enter, RECEIVE saves the incoming data set with the new name. You can also respond to the "already exists" prompt by typing "END," which returns you to the TSO prompt. In fact, this is the default action, so pressing Enter without typing anything also returns you to the TSO prompt, leaving any unrecovered data sets for you to deal with later.

Most other potential responses to the "Enter restore parameters" prompt let you specify allocation details of the data set to recover, if you want to change them. To see them before recovering the data set, enter a question mark (?) in response to the prompt.

If you never respond with END, the RECEIVE program goes through the waiting messages and data sets one a time, giving the same information and asking the same question with each. When none are left, it tells you

```
No more files remain for the receive command to process.
```

Many users want to check whether data sets are waiting for them every time they log on. Your ID may be set up to invoke the RECEIVE command each time you log on; if not, you can easily add it to (or create) a LOGON.CLIST data set. (Section 26.2.1, "The Automatic Logon Com-

mand File," covers `LOGON.CLIST` in more detail.)

26.4 ISPF

Chapter 25, "The MVS ISPF Text Editor," showed how to start ISPF, pick the appropriate choice from the main (or "primary") menu to edit a data set, and fill out the entry panel to indicate the data set you want to edit. Most of the important things that you do with ISPF begin with similar steps: selecting a menu choice and then filling out a panel to indicate the data set that you want to act on.

As you go from menu to menu in ISPF, you may display a menu that you didn't mean to. As the bottom of the screen shows you, the PF3 key ("END") ends the display of the current menu—in other words, it backs out of that menu to the previous one. To leave the main menu, note how the last menu choice has an "X" instead of a number preceding it, as shown in Figure 26.2. This means that entering `X` at the `OPTION ===>` prompt returns you to the TSO prompt.

```
----------------------  ISPF/PDF PRIMARY OPTION MENU  ------------------------
OPTION  ===> _
                                                          USERID   -JOEUSER
   0  ISPF PARMS  - Specify terminal and user parameters    TIME     - 09:24
   1  BROWSE      - Display source data or output listings  TERMINAL - 3278
   2  EDIT        - Create or change source data            PF KEYS  - 24
   3  UTILITIES   - Perform utility functions
   4  FOREGROUND  - Invoke language processors in foreground
   5  BATCH       - Submit job for language processing
   6  COMMAND     - Enter TSO Command, CLIST, or REXX exec
   7  DIALOG TEST - Perform dialog testing
   8  LM UTILITIES- Perform library administrator utility functions
   9  IBM PRODUCTS- Additional IBM program development products
  10  SCLM        - Software Configuration and Library Manager
   D  DB2         - DB2 Facilities
   H  HSM         - DFHSM Facilities
   C  CHANGES     - Display summary of changes for this release
   S  SDSF        - System Display and Search Facility
  SO  DFSORT      - DFSORT Facility
   T  TUTORIAL    - Display information about ISPF/PDF
   X  EXIT        - Terminate ISPF using log and list defaults
 F1=HELP      F2=SPLIT     F3=END       F4=RETURN    F5=RFIND     F6=RCHANGE
 F7=UP        F8=DOWN      F9=SWAP     F10=LEFT     F11=RIGHT    F12=RETRIEVE
```

Figure 26.2 ISPF primary option menu.

If you do anything more than browse around the menus in ISPF, like running programs or copying, renaming, editing or deleting data sets, then ISPF may keep a log of your activity. This log lists the date, time, and nature of each transaction. When you exit from the main menu, you may

see a screen asking what you want to do with that log; Figure 26.3 shows an example.

```
------------------  SPECIFY DISPOSITION OF LOG DATA SET  ---------------------
COMMAND ===>

LOG DATA SET DISPOSITION              LIST DATA SET OPTIONS NOT AVAILABLE
-------------------------             ----------------------------------
Process option   ===>
SYSOUT class     ===> A
Local printer ID ===>

VALID PROCESS OPTIONS:
    PD - Print data set and delete
    D  - Delete data set without printing
    K  - Keep data set (allocate same data set in next session)
    KN - Keep data set and allocate new data set in next session

  Press ENTER key to complete ISPF termination.
  Enter END command to return to the primary option menu.

JOB STATEMENT INFORMATION:  (Required for system printer)
  ===> //JOEUSERA  JOB  (ACCOUNT),'NAME'
  ===> //*
  ===> //*
```

Figure 26.3 Entry panel for dealing with ISPF log.

The four "VALID PROCESS OPTIONS" are fairly self-explanatory; you'll probably want to choose `D` most of the time to delete the log so that it doesn't take up space. You might want to try entering `K` at the `Process Option` prompt some time to see what the log looks like. When ISPF returns you to the TSO prompt, it displays a message similar to this:

```
JOEUSER.SPFLOG1.LIST HAS BEEN KEPT.
READY
```

Now you know the name of the data set where the system stored this log. You can then use the `LIST` command to look at it.

If you choose `K`, ISPF continues to use this data set to log transactions the next time you use ISPF. If you choose `KN`, ISPF saves your transactions but create a new log data set the next time you use ISPF.

26.4.1 Allocating Data Sets

Selecting `UTILITIES` from the main ISPF menu displays a new menu that looks similar to the

one shown in Figure 26.4. As the description of the DATASET choice tells you, choosing it allows you to allocate data sets. Selecting DATASET displays a menu similar to the one shown in Figure 26.5.

```
-------------------------  UTILITY SELECTION MENU  ----------------------------
OPTION  ===>

    1  LIBRARY    - Compress or print data set.  Print index listing.
                    Print, rename, delete, or browse members
    2  DATASET    - Allocate, rename, delete, catalog, uncatalog, or
                    display information of an entire data set
    3  MOVE/COPY  - Move, copy, or promote members or data sets
    4  DSLIST     - Print or display (to process) list of data set names
                    Print or display VTOC information
    5  RESET      - Reset statistics for members of ISPF library
    6  HARDCOPY   - Initiate hardcopy output
    8  OUTLIST    - Display, delete, or print held job output
    9  COMMANDS   - Create/change an application command table
   10  CONVERT    - Convert old format menus/messages to new format
   11  FORMAT     - Format definition for formatted data Edit/Browse
   12  SUPERC     - Compare data sets (Standard dialog)
   13  SUPERCE    - Compare data sets (Extended dialog)
   14  SEARCH-FOR - Search data sets for strings of data
```

Figure 26.4 ISPF Utility menu.

```
---------------------------  DATA SET UTILITY  ------------------------------
OPTION  ===>

   A - Allocate new data set                C - Catalog data set
   R - Rename entire data set               U - Uncatalog data set
   D - Delete entire data set               S - Data set information (short)
   blank - Data set information

ISPF LIBRARY:
   PROJECT ===> JOEUSER
   GROUP   ===> FIELDING
   TYPE    ===> TEXT

OTHER PARTITIONED OR SEQUENTIAL DATA SET:
   DATA SET NAME  ===>
   VOLUME SERIAL  ===>          (If not cataloged, required for option "C")

DATA SET PASSWORD ===>          (If password protected)
```

Figure 26.5 ISPF data set utility menu.

Enter the letter "A" at the OPTION ===> prompt and the name of the data set to allocate in the ISPF LIBRARY section of the panel. When you press Enter, ISPF displays ALLOCATE NEW DATA SET screen, where you enter the details of how you want your data set allocated. As you can see in Figure 26.6, this screen includes the data set name you entered on the DATA SET UTILITY screen. The crucial fields on the ALLOCATE NEW DATA SET screen are covered in section 24.1.7, "Allocating Data Sets."

```
------------------------  ALLOCATE NEW DATA SET  -----------------------------
COMMAND ===>

DATA SET NAME: JOEUSER.FIELDING.TEXT

   VOLUME SERIAL      ===> USERMD      (Blank for authorized default volume) *
   GENERIC UNIT       ===>             (Generic group name or unit address) *
   SPACE UNITS        ===> TRACK       (BLKS, TRKS, or CYLS)
   PRIMARY QUANTITY   ===> 2           (In above units)
   SECONDARY QUANTITY ===> 2           (In above units)
   DIRECTORY BLOCKS   ===> 0           (Zero for sequential data set)
   RECORD FORMAT      ===> FB
   RECORD LENGTH      ===> 80
   BLOCK SIZE         ===> 27920
   EXPIRATION DATE    ===>             (YY/MM/DD, YYYY/MM/DD
                                        YY.DDD, YYYY.DDD in Julian form
                                        DDDD for retention period in days
                                        or blank)

   ( * Only one of these fields may be specified)
```

Figure 26.6 Data set allocation screen.

It's still a lot of trouble, dealing with all those allocation parameters; luckily, ISPF offers you a way to model a new data set's allocation parameters on those of an existing data set. On the DATA SET UTILITY screen, if you enter an existing data set's name and press the Enter key without entering anything at the OPTION ===> prompt (note how the OPTION choices include "blank - Data set information") ISPF displays allocation information about the data set whose name you entered. Figure 26.7 shows an example of the displayed information.

```
-------------------------  DATA SET INFORMATION  ----------------------------
COMMAND ===>

DATA SET NAME: JOEUSER.BLAKE.TEXT

GENERAL DATA:                              CURRENT ALLOCATION:
   Volume serial:         USERMC             Allocated Tracks:           1
   Device type:           3390               Allocated extents:          1
```

```
    Organization:          PS
    Record format:         VB
    Record length:         84
    Block size:            6233              CURRENT UTILIZATION:
    1st extent Tracks:     1                    Used Tracks:                 1
    Secondary Tracks:      5                    Used extents:                1

    Creation date:         1994/06/04
    Expiration date:       ***NONE***
```

Figure 26.7 Data set allocation information.

When you leave this screen and go through the steps of allocating a new data set, the next time you reach the ALLOCATE NEW DATA SET screen, ISPF fills out the allocation parameters of the new data set with those of the existing data set about which you just inquired. Since you entered the name of the new data set that you want to allocate in order to reach the ALLOCATE NEW DATA SET screen, all that remains when viewing that screen is to press Enter to allocate the new data set with the parameters of the existing data set.

26.4.2 Copying Data Sets

Copying a data set in TSO or a file in another operating system usually means "make me a new file that is a copy of this data set, and give the copy the following name." When you do this in ISPF, there's a catch: the data set specified as the destination of the copy operation must already exist. In other words, you must allocate it before you begin the copy operation. The best way to do this is to allocate the copy with the same allocation parameters as the data set being copied, as explained in section 24.4.1, "Allocating Data Sets."

Once it's allocated, you can perform the copy operation. We saw in the last section that after selecting UTILITIES from the main ISPF menu, selecting DATASET from the UTILITIES menu makes it possible to allocate data sets. Another choice on the UTILITIES menu is MOVE/COPY. Selecting it displays the panel shown in Figure 26.8 for you to fill out.

```
 --------------------------  MOVE/COPY UTILITY  ------------------------------
 OPTION  ===>

    C - Copy data set or member(s)              CP - Copy and print
    M - Move data set or member(s)              MP - Move and print
    L - Copy and LMF lock member(s)             LP - Copy, LMF lock, and print
    P - LMF Promote data set or member(s)       PP - LMF Promote and print

 SPECIFY "FROM" DATA SET BELOW, THEN PRESS ENTER KEY
```

```
FROM ISPF LIBRARY:        ------ Options C, CP, L, and LP only -------
   PROJECT ===> JOEUSER   |                                           |
   GROUP   ===> BLAKE     ===>           ===>           ===>
   TYPE    ===> TEXT
   MEMBER  ===>                 (Blank or pattern for member selection list,
                                 '*' for all members)

FROM OTHER PARTITIONED OR SEQUENTIAL DATA SET:
   DATA SET NAME  ===>
   VOLUME SERIAL  ===>          (If not cataloged)

DATA SET PASSWORD ===>          (If password protected)
```

Figure 26.8 ISPF screen for moving or copying data sets.

To copy a data set, enter "C" at the `OPTION ===>` prompt and the name of the data set to copy in the FROM ISPF LIBRARY section of the panel. When you press Enter, ISPF displays a screen similar to the one shown in Figure 26.9 to find out the name of your copy.

```
COPY --- FROM JOEUSER.BLAKE.TEXT ------------------------------------------
COMMAND ===>

SPECIFY "TO" DATA SET BELOW.

TO ISPF LIBRARY:
   PROJECT ===> JOEUSER
   GROUP   ===>
   TYPE    ===>
   MEMBER  ===>

TO OTHER PARTITIONED OR SEQUENTIAL DATA SET:
   DATA SET NAME  ===>
   VOLUME SERIAL  ===>          (If not cataloged)

DATA SET PASSWORD ===>          (If password protected)

"TO" DATA SET OPTIONS:
   IF PARTITIONED, REPLACE LIKE-NAMED MEMBERS ===>        (YES or NO)
   IF SEQUENTIAL, "TO" DATA SET DISPOSITION   ===>        (OLD or MOD)
   SPECIFY PACK OPTION FOR "TO" DATA SET      ===>        (YES, NO or blank)
```

Figure 26.9 ISPF screen prompting for destination of copy operation.

In the TO ISPF LIBRARY section, enter the name of the data set to which you want to copy your "FROM" data set. Press Enter, and ISPF makes the copy and returns you to the MOVE/COPY UTILITY screen, with the message "DATA SET COPIED" displayed in the upper-right.

26.4.3 Renaming Data Sets

We've already seen (Figure 26.4) the menu that appears when you select UTILITIES from the main ISPF menu. On this menu, the LIBRARY choice offers you utilities for dealing with members of partitioned data sets, and the DATASET choice offers you utilities for dealing with sequential data sets. So, to rename (or delete) a member of a PDS, select LIBRARY; to rename or delete a sequential data set, select DATASET.

When you select LIBRARY, ISPF displays a panel similar to the one shown in Figure 26.10. At the `OPTION ===>` prompt, enter an "R" and use your Tab key to move to the ISPF LIBRARY part of the panel so that you can specify the data set member to rename. In the PROJECT, GROUP, and TYPE fields, enter the name of the partitioned data set containing the member you want to rename. In the MEMBER field, enter the member's name, and in the NEWNAME field, enter its new name. (When renaming a member of a PDS you can't change any part of the PDS name; whatever the member's name becomes, it's still part of the same PDS.) When you press Enter, ISPF renames the data set and displays a message in the upper-right of the screen telling you that it did so.

```
----------------------------  LIBRARY UTILITY  ----------------------------
OPTION  ===>

   blank - Display member list          B - Browse member
   C - Compress data set                P - Print member
   X - Print index listing              R - Rename member
   L - Print entire data set            D - Delete member
   I - Data set information             S - Data set information (short)

ISPF LIBRARY:
   PROJECT ===> JOEUSER
   GROUP   ===> URBASE    ===>          ===>          ===>
   TYPE    ===> CLIST
   MEMBER  ===> URTEST          (If "P", "R", "D", "B", or blank selected)
   NEWNAME ===> UR              (If "R" selected)

OTHER PARTITIONED OR SEQUENTIAL DATA SET:
   DATA SET NAME  ===>
   VOLUME SERIAL  ===>          (If not cataloged)

DATA SET PASSWORD ===>          (If password protected)
```

Figure 26.10 The ISPF library utility screen.

If Joe User pressed the Enter key after filling out the panel as shown in Figure 26.10, he would rename the URTEST member of the URBASE.CLIST partitioned data set with a new name of UR. After he pressed Enter, ISPF would display the message MEMBER URTEST RENAMED in the upper-right of the screen.

To rename a sequential data set or an entire partitioned data set (as opposed to just renaming one of the members of a PDS), you'll use the same panel you used to allocate a data set: the DATA SET UTILITY screen (Figure 24.5). Display it by selecting DATASET from the UTILITIES menu.

Enter "R" at the OPTION ===> prompt and the name of the data set to rename in the ISPF LIBRARY section of the panel. Press Enter, and ISPF displays a panel similar to the one shown in Figure 26.11 to learn the new name you want to give this data set.

```
----------------------------  RENAME DATA SET  -------------------------------
COMMAND ===>

DATA SET NAME: JOEUSER.BLAKE.TEXT
VOLUME:        USERME

ENTER NEW NAME BELOW:     (The data set will be recataloged.)

ISPF LIBRARY:
   PROJECT ===> JOEUSER
   GROUP   ===> BLAKE
   TYPE    ===> BACKUP

OTHER PARTITIONED OR SEQUENTIAL DATA SET:
   DATA SET NAME  ===>
```

Figure 26.11 ISPF screen prompting you for the new name of your renamed data set.

The DATA SET NAME field at the top of this panel shows the name of the data set that you said you wanted to rename. Now you only need to fill out the new name in the GROUP and TYPE fields. After you do this and press Enter, ISPF displays the message "DATA SET RENAMED" in the upper-right of the screen.

26.4.4 Deleting Data Sets

You may have noticed that the LIBRARY UTILITY and DATA SET UTILITY screens used to rename data set members or entire data sets offered "D" as an alternative command to enter at the

`OPTION ===>` prompt. When you enter this and enter a data set member name on the LIBRARY UTILITY screen, you delete that member from the specified partitioned data set, leaving the PDS otherwise intact.

When you enter a "D" at the `OPTION ===>` prompt of the DATASET screen and a data set name in the ISPF LIBRARY section, you delete the whole data set, whether sequential or partitioned. When you press Enter, ISPF displays a screen similar to the one shown in Figure 26.12 to make sure that you really want to delete the specified data set.

```
----------------------------  CONFIRM DELETE  -------------------------------
COMMAND ===>

DATA SET NAME: JOEUSER.BLAKE.TEXT
VOLUME:        USERME
CREATION DATE: 1994/06/20


INSTRUCTIONS:

   Press ENTER key to confirm delete request.
      (The data set will be deleted and uncataloged.)

   Enter END command to cancel delete request.
```

Figure 26.12 ISPF screen confirming a request to delete a data set.

Press Enter to follow through and delete the data set or enter "END" at the `COMMAND ===>` prompt to cancel the deletion. Either way, you return to the DATA SET UTILITY screen, which will have a message in the upper right telling you either "DATA SET DELETED" or "DATA SET NOT DELETED."

26.4.5 Displaying A Data Set's Contents

Because ISPF takes better advantage of your terminal than TSO, it has no direct equivalent to the `LIST` command to put a data set's contents on your screen one screenful at a time from beginning to end. Instead, ISPF has a utility called BROWSE that gives you more flexibility when viewing a data set. Basically, it's just like the ISPF editor, but it won't let you change a data set's contents.

All of the editor's commands for moving around and searching for text work the same when you browse a data set. This means that when looking at the middle or end of a data set, you can go backwards and look at the text before the currently displayed text, which you can't do when you

use `LIST` at the TSO command line.

To browse a data set, select BROWSE from the ISPF main menu. On the next screen, enter the name of a sequential data set or a member of a partitioned data set and press Enter. ISPF starts the BROWSE program with the beginning of your data set displayed on the screen.

26.4.6 Printing a Data Set

Print a data set or a member of a partitioned data set with the same LIBRARY UTILITY panel (accessible by selecting LIBRARY from the UTILITIES menu) that you used to rename a member of a PDS (Figure 26.10). Instead of entering "R" for "rename," you enter either "L" to print a sequential data set or "P" to print a member of a PDS. Type the sequential data set or PDS member's name in the ISPF LIBRARY section of the panel, press Enter, and you're done.

26.5 A Sample MVS Session

One morning you log on to your JOEUSER ID, and you see the screen shown in Figure 26.13.

```
ICH70001I JOEUSER   LAST ACCESS AT 14:40:39 ON MONDAY, JANUARY 18, 1994
JOEUSER LOGON IN PROGRESS AT 14:42:38 ON JANUARY 18, 1994
 ********************************************************************
 *    The system will be unavailable from 12:01 AM to 10 AM         *
 *    Sunday January 24th for maintenance.                          *
 ********************************************************************
 Are those CLISTs ready or what? Give me whatever you have. LNIVEN
You have no messages or data sets to receive.
**** NATIVE TSO READY ****
READY
```

Figure 26.13 Joe User's logon messages one morning.

The message from the system administrator doesn't concern you, because you had no plans to log on Sunday morning. The other message, from your friend Larry Niven, is more urgent; you had promised to write him several CLISTs to use with the inventory database system he is developing.

You may as well send him what you've done so far, but you want to see what you have first. They're in a partitioned data set called `INVEN.CLIST`. You you list out the members of this data set by entering the `LISTDS` command with the `MEM` operand and see the output shown in

Figure 26.14.

```
listds inven.clist mem
 JOEUSER.INVEN.CLIST
 --RECFM-LRECL-BLKSIZE-DSORG
   FB    80    3120    PO
 --VOLUMES--
   USERMD
 --MEMBERS--
   ADD
   DELETE
   UPDATE
   REPORT
   TESTING
```

Figure 26.14 List of members in INVEN.CLIST.

The TESTING one was for playing with some of the CLIST commands. You delete it from the PDS, because you don't need to send that one to Larry:

```
delete inven.clist(testing)
```

The other CLISTs were pretty good, but the UPDATE one is still giving you some problems. No use putting off Larry further; you'll send him the whole PDS of CLISTs, but warn him about the UPDATE CLIST in a separate message.

You type the message into a data set called FORLARRY.TEXT. The TRANSMIT command can be picky about how these message data sets are allocated, so you allocate it by copying the allocation details from the data set FORMARY.TEXT, which you sent to Mary Jones last week:

```
allocate dataset(forlarry.text) like(formary.text)
```

You enter text into FORLARRY.TEXT with the ISPF editor. Because the editor is the second choice on the ISPF main menu, you jump right to the editor by including a "2" on the command line when you start up ISPF:

```
ispf 2
```

After indicating the data set you want to edit on the EDIT ENTRY PANEL, you type in your message. Figure 26.15 shows how it might look when you are done.

```
 EDIT ---- JOEUSER.FORLARRY.TEXT ------------------------------ COLUMNS 001 072
 COMMAND ===>                                                  SCROLL ===> PAGE
 ****** ***************************** TOP OF DATA ******************************
```

```
000100 Larry -
000200 I'm sending you INVEN.CLIST, which has the CLISTs.  Most of them
000300 work fine, but the UPDATE one is still a little flaky.  Let me
000400 know if you have any questions.
****** **************************** BOTTOM OF DATA ****************************
```

Figure 26.15 Joe's message to be sent to Larry.

After writing the message, you press PF4 to show that you've finished editing. (If you had wanted to edit another data set, you would have pressed PF3, which would return you to the screen where you enter the name of the data set to edit. This time, however, you no longer need the editing program.)

When you return to the TSO prompt, you are ready to send the CLISTs and their accompanying message to larry on the SATURN system. You can do this with one command. It's a long one, though; after you enter the first part, you type a plus sign to show TSO that you haven't finished entering the command.

```
transmit saturn.lniven dataset(inven.clist) +
```

Then, you press Enter, finish typing the command,

```
msgdataset(forlarry.text)
```

and press Enter again. TSO displays a message showing you that it successfully transmitted the CLIST partitioned data set and the accompanying message.

You'd better straighten out the UPDATE CLIST as soon as possible. It would be a good idea to print out a hard copy so that you can sit down with a cup of coffee and write some notes on it. Since the accounting department's mainframe printer is near the coffee machine, you'll send the printout there so that you can pick it up and get your coffee in one trip. You enter the command to print it:

```
printds dataset(inven.clist(update)) dest(acctng)
```

You're not even going to try to fix the UPDATE CLIST until you've had a good hard look at the printout. You log off by typing

```
LOGOFF
```

and head for the printer and coffee machine.