File No. S360-36
Order No. GC28-6763-0

OS

Systems Reference Library

IBM System/360 Operating System:

Time Sharing Option

Terminal User's Guide

The Time Sharing Option (TSO) of the IBM System/360 Operating System lets you use the facilities of a computer from a terminal. You define your work to the system through the TSO Command Language. This publication explains to all users of TSO how to use the TSO Command Language to perform the following functions:

- Start and end a terminal session.
- Enter and manipulate data.
- Program at the terminal.
- Test a program.
- Write and use command procedures.
- Control a system with TSO.

After becoming familiar with the information presented in this manual, you may use IBM System/360 Operating System: Time Sharing Option, Command Language Reference, GC28-6732 for review and reference.

Information in this publication for TSO is for planning purposes until that item is available.

Preface

This publication describes how to use the TSO Command Language to all TSO terminal users. The commands can be used to perform the following functions:

- Start and end a terminal session.
- Enter and manipulate data.
- Program at the terminal.
- Test a program.
- Write and use command procedures.
- Control a system with TSO.

This publication tells you what commands to use to perform these functions. For details on how to code each command, refer to the publication IBM System/360 Operating System: Time Sharing Option, Command Language Reference, GC28-6732.

Before reading this manual you should be aware of three facts:

- Program Products are not discussed in this manual.
- All examples in this manual show the user's input in lowercase letters and the system output in uppercase letters.
- All examples in this manual assume that you are using an IBM 2741 Communications Terminal, and that you must press the RETURN key to enter data. For information on your type of terminal refer to the publication IBM System/360 Operating System: Time Sharing Option, Terminals, GC28-6762.

First Edition (March, 1971)

This edition applies to release 20.1, of IBM System/360 Operating System, and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Changes are continually made to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/360 SRL Newsletter, Order No. GN20-0360, for the editions that are applicable and current.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Programming Systems Publications, Department D58, PO Box 390, Poughkeepsie, N.Y. 12602

Contents

Illustrations

Figures

Tables

Introduction

TSO is the time sharing option of the IBM System/360 Operating System. TSO lets you use the facilities of a computer at a terminal. A terminal is a typewriter-like device connected through telephone or other communication lines to the computer. A terminal can be at any distance from the computer -- in the same room or in another city. Because the system processes instructions much faster than you can enter them through the terminal, it can process input from many terminals at the same time besides processing work entered in the conventional manner in the computer room. However, due to the speed of the system, you will think you have almost exclusive use of the system.

You tell the system what work you want done by typing in one or more of the commands that form the TSO command language. The command language can be used to:

- Enter, store, modify, and retrieve data at the terminal.
- Solve mathematical problems.
- Develop programs written in Assembler, FORTRAN, COBOL, PL/I, or other languages.
- Execute programs.
- Control the operation of a system with TSO from the terminal.

Your installation determines which of the facilities of the system you can use. That is, the installation determines which commands are available to you.

When you enter a command in the system, the system performs the work requested by that command and sends messages back to your terminal. The messages tell you the status of your program and whether the system is ready to accept another command. You can interrupt the processing of a command at any time and enter a new one.

If you make a mistake typing in a command, or if you fail to include some necessary information with the command, the system sends you a message prompting you for the necessary information. You then respond by typing in the information requested.

If you receive a message you don't understand, you can type in a question mark to request more information. The system will then send you a more detailed message, if available.

Whenever you are not sure which command to use or how to use a particular command, you can type HELP. HELP is a command that provides you with information on all other TSO commands.

This manual explains how to use the command language. The manual is divided into the following sections:

1. What you must know to use TSO.
2. Starting and ending a terminal session.
3. Entering and manipulating data.
4. Programming at the terminal.
5. Testing a program.
6. Using and writing command procedures.
7. Controlling a system with TSO.

The first three items must be known by all system users. The remaining items depend on what your installation has authorized you to do.

This manual tells you what commands to use to perform the functions mentioned above. For details on how to code each command, refer to the manual IBM System/360 Operating System: Time Sharing Option, Command Language Reference.

What You Must Know to Use TSO

Before you begin a terminal session, you should know:

- How to enter information at the terminal.
- How to use the TSO commands.
- How to interpret TSO messages.
- How to use the attention interruption.
- How to use the HELP command.

Entering Information at the Terminal

All TSO terminals have a typewriter-like keyboard through which you enter information into the system. The features of each keyboard vary from terminal to terminal; for example, one terminal may not have a backspace key, while another may not allow for lowercase letters. The features of each terminal as they apply to TSO are described in the publication, IBM System/360 Operating System: Time Sharing Option, Terminals.

Certain conventions apply to the use of all TSO terminals. They are:

- Any lowercase letters you type are interpreted by the system as uppercase letters. For example, if you type in:

  abcDe8-fg

  the system interprets it as:

  ABCDE8-FG

  The only exceptions are certain text-handling applications which allow you to type in text with both uppercase and lowercase letters. Text handling is discussed in the section "Entering and Manipulating Data".

- All messages or other output sent to you by the system come out in uppercase letters. The only exception is the output from the special text-handling applications mentioned previously which comes out both in uppercase and lowercase.

TSO also provides a way of correcting your typing mistakes. You can request that the character you just typed be deleted or that all the preceding characters in the line be deleted. You can define your own character-deletion and line-deletion characters, or you can use the default characters in the system. For example, if the default characters are the quotation mark (") for deleting the preceding character, and the percent sign (%) for deleting all the preceding characters of the line and you type the following message:

```
first ent%Sect"onft""d ENR"try
```

it is received by the system as:

```
SECOND ENTRY
```

Note that you can use the character-deletion character repetitively to delete more than one of the preceding characters in the line.

The blank space produced when you hit the space bar is also considered to be a character, and you can delete it using the character-deletion or line-deletion characters. For example, if you type the following line:

```
a b%cd  "E "f
```

it is received by the system as

```
CD EF
```

After you type a line and make any necessary corrections, you enter that line as follows:

- Press the RETURN key on an IBM 2741 Communications Terminal.
- Press the RETURN key on an IBM 1052 Printer-Keyboard (If the 1052 does not have the automatic EOB feature, use the ALTN code).
- Hold the CTRL key and press the XOFF key on a Teletype terminal.

Notes:

- All examples in this manual assume that you are using an IBM 2741 Communications terminal, and that you must press the RETURN key to enter a line. For information on your type of terminal, refer to the publication IBM System/360 Operating System: Time Sharing Option, Terminals, GC28-6762.
- If you want to enter a null line, that is a line with no characters in it, press the key used to enter data.

You cannot use the character-deletion and line-deletion characters to make corrections to the line after you enter it. If the line you entered was a command, you must use the attention interruption (described later in this section) to cancel the command, and then you must reenter the command. If the line you entered was data, you can change it using the EDIT command (described in the section, "Entering and Manipulating Data".)

Normally, you will use the default characters in the system. However, you can use the PROFILE command to establish your own character-deletion and line-deletion characters. The PROFILE command is described in the section, "Starting and Ending a Terminal Session". The ability to change the character-deletion and line-deletion characters is particularly useful when you use more than one type of terminal. For example, the backspace key and the attention key are the usual default characters. Any time you have to use a terminal that does not have backspace and attention keys, you can use the PROFILE command to select two other suitable characters as the character-deletion and line-deletion characters.

Commands

You can communicate with the system by typing requests for work, commands, at the terminal. Different commands specify different kinds of work. You can store data in the system, change the data, and retrieve it at your convenience. You can create programs, test them, execute them and obtain the results at your terminal. The commands make the facilities of the system available at your terminal.

When you use a command to request work, the command establishes the scope of the work to the system. For some commands, the scope of the work encompasses several operations that you can identify separately. After entering the command, you may specify one of the separately identifiable operations by entering a subcommand. A subcommand, like a command, is a request for work; however, the work requested by a subcommand is a particular operation within the scope of work established by a command.

The commands and subcommands recognized by TSO form the TSO command language. The command language is designed to be easy to use. The command names and subcommand names are typically familiar English words, often verbs, that describe the work to be done. The number of command names and subcommand names that you must learn has been kept to a minimum. (Your installation can add its own commands to perform functions not provided by the TSO command language.)

Besides entering the name of the command or subcommand, you are often required to specify additional information to pinpoint the function you want performed. You define the additional information with operands (words or numbers that accompany the command names and subcommand names.) Most of the operands have default values that are used by the system if you choose to omit the operand from the command or subcommand. However, some operands do not have default values. If you fail to provide a required operand for which there is no default, the system sends you a prompting message asking you to supply the operands. The publication, IBM System/360 Operating System: Time Sharing Option, Command Language Reference shows all operands for each command, and indicates the default values where applicable, and describes how to enter the commands.

You can abbreviate many of the command names, subcommand names and operands. Together, the defaults and abbreviations decrease the amount of typing required. (The abbreviations and their use are discussed in the publication IBM System/360 Operating System: Time Sharing Option, Command Language Reference.)

Table 1 lists the commands and their subcommands in alphabetical order.

Table 1. TSO Commands and Subcommands, Including Abbreviations

COMMAND (abbreviation) SUBCOMMAND (abbreviation)	COMMAND (abbreviation) SUBCOMMAND (abbreviation)
ACCOUNT	LISTDS (LISTD)
ADD (A)	LOADGO (LOAD)
CHANGE (C)	LOGOFF
DELETE (D)	LOGON
END	*MERGE
HELP (H)	OPERATOR (OPER)
LIST (L)	CANCEL (C)
LISTIDS (LISTI)	DISPLAY (D)
ALLOCATE (ALLOC)	END
*ASM	HELP (H)
*CALC	MODIFY (F)
*DELETE	MONITOR (MN)
*END	SEND
*HELP	STOP (P)
*SAVE	OUTPUT (OUT)
CALL	CONTINUE (CONT)
CANCEL	END
*COBOL (COB)	HELP (H)
*CONVERT (CON)	SAVE (S)
*COPY	PROFILE (PROF)
DELETE (D)	PROTECT (PROT)
EDIT	RENAME (REN)
BOTTOM (B)	RUN (R)
CHANGE (C)	SEND (SE)
DELETE (D)	STATUS (ST)
DOWN	SUBMIT (SUB)
END	TERMINAL (TERM)
FIND (F)	TEST
*FORMAT (FORM)	AT
HELP (H)	CALL
INPUT (I)	DELETE (D)
INSERT (IN)	DROP
LIST (L)	END
*MERGE (M)	EQUATE (EQ)
PROFILE (PROF)	FREEMAIN (FREE)
RENUM (REN)	GETMAIN (GET)
RUN (R)	GO
SAVE (S)	HELP (H)
SCAN (SC)	LIST (L)
TABSET (TAB)	LISTDCB
TOP	LISTDEB
UP	LISTMAP
VERIFY (V)	LISTPSW
EXEC (EX)	LISTTCB
*FORMAT (FORM)	LOAD
*FORT	OFF
FREE	QUALIFY (Q)
HELP (H)	RUN (R)
LINK	WHERE (W)
*LIST (L)	TIME
LISTALC (LISTA)	**END
LISTBC (LISTB)	**PROC
LISTCAT (LISTC)	**WHEN

*Available as program products
**For use in command procedures

WHEN TO ENTER A COMMAND OR SUBCOMMAND

The system lets you know when it is ready to accept a new command by sending you the message:

```
READY
```

The ACCOUNT, EDIT, OPERATOR, OUTPUT and TEST commands have subcommands. After entering one of these commands the system lets you know it is ready to accept a subcommand by sending you back the name of the command. For example, in the following sequence you enter the OPERATOR command after receiving a READY message. The system then sends you the OPERATOR message indicating that you can enter any of the subcommands of the OPERATOR command:

```
READY
operator
OPERATOR
```

If instead of entering a subcommand you want to enter a command, request an attention interruption (described later in this chapter) or enter the END subcommand to make the READY message appear again.

The system remains able to receive commands until you enter one of the five commands that have subcommands. The system then accepts only that command's subcommands until you request a READY message.

HOW TO ENTER A COMMAND OR SUBCOMMAND

After you receive the appropriate message to let you know the system is ready to receive a command or subcommand:

1. Type the command or subcommand name and the selected operands.
2. Correct any typing mistakes with the character-delete and line-deletion characters.
3. Press the RETURN key.

If all the operands do not fit in one line you should follow this sequence:

1. Type the command and subcommand name and the selected operands.
2. Type a hyphen (-) at the end of the line.
3. Press the RETURN key.
4. Continue entering the operands. If they do not fit in the second line repeat from 2.
5. Press the RETURN key to enter the command.

You can type command and subcommand names and operands in either uppercase or lowercase letters. You may prefer to type your commands and subcommands in lowercase so you can distinguish your input from the system's messages in your listing. (The system prints in uppercase letters.) Typing your input in lowercase letters is also faster than typing in uppercase letters. All examples in this manual show the user's input in lowercase letters, and the system output in uppercase letters.

Messages

There are four types of messages:

- Mode messages.
- Prompting messages.
- Informational messages.
- Broadcast messages.

MODE MESSAGES

A mode message tells you when the system is ready to accept a new command or subcommand. (See "When to Enter a Command".) When the system is ready to accept a new command it prints:

```
READY
```

When you enter a command that has subcommands and the system is ready to accept its subcommands, it prints the name of the command which can be any of the following:

```
ACCOUNT
EDIT
OPERATOR
OUTPUT
TEST
```

You can then enter the subcommands you want to use. The TEST message also appears after each TEST subcommand has been processed. If the system has to print any output or other messages as a result of the previous command or TEST subcommand, it does so before printing the mode message. (The use of mode messages in the EDIT command is discussed in the section "Entering and Manipulating Data".)

Sometimes you can save a little time by entering two or more commands in succession without waiting for the intervening READY message. The system then prints the READY messages in succession after the commands. For example, if you enter the DELETE, FREE, and RENAME commands and wait for the intervening mode message between the commands, the output (or listing) will be:

```
READY
delete...
READY
free...
READY
rename...
READY
```

If you enter the same commands without waiting for the intervening mode messages, your listing will be:

```
READY
delete...
free...
rename...
READY
READY
READY
```

There is a drawback to entering commands without waiting for the intervening mode messages. If you make a mistake in one of the commands, the system sends you messages telling you of your mistake, and then it cancels the remaining commands you have entered. After you correct the error, you have to reenter the other commands.

Unless you are sure that there are no mistakes in your input, you should wait for a READY message before entering a new command.

Note: Some terminals "lock" the keyboard after you enter a command, and therefore you cannot enter commands without waiting for the intervening READY message. See the publication IBM System/360 Operating System: Time Sharing Option, Terminals for information on your terminal.

PROMPTING MESSAGES

A prompting message tells you that required information is missing or that information you supplied was incorrectly specified. A prompting message asks you to supply or correct that information. For example, "data set name" is a required operand of the CALL command; if you enter the CALL command without that operand the system will prompt you for the data set name and your listing will look as follows:

```
READY
call
ENTER DATA SET NAME -
```

You should respond by entering the requested operand, in this case the data set name, and by pressing the RETURN key to enter it. For example if the data set name is ALPHA.DATA you would complete the prompting message as follows:

```
ENTER DATA SET NAME-alpha.data
```

To specify whether or not you want to receive prompting messages, use the PROMPT or NOPROMPT operand of the PROFILE command. This command is described in the section, "Starting and Ending a Terminal Session".

Sometimes you can request another message that explains the initial message more fully. If the second message is not enough, you can request a further message that will give you more detailed information, and so on.

To request an additional level of message:

1. Type a question mark (?) in the first position of the line.
2. Press the RETURN key.

If you enter a question mark when there are no other messages that explain the one you received, you receive the following message:

```
NO INFORMATION AVAILABLE
```

You can stop a prompting sequence by entering the requested information or by requesting an attention interruption.

INFORMATIONAL MESSAGES

An informational message tells you about the status of the system and your terminal session. For example, an informational message can tell you how much time you have used. Informational messages do not require a response.

If your informational message ends with a plus sign (+) you can request an additional message by entering a question mark after READY (?) as described for prompting messages. Informational messages only have a second level of messages, while prompting messages may have more than one.

BROADCAST MESSAGES

Broadcast messages are messages of general interest to users of the system. Both the system operator and any user of the system can send broadcast messages. The system operator can send messages to all users of the system or to individual users. For example, he may send the following message to all users:

```
DO NOT USE TERMINALS #4, 5 AND 6 ON 6/30.  THEY ARE RESERVED FOR
DEPARTMENT 791.
```

You, or any other user, can send messages to other users or to the system operator. For example, you may send, or receive, the following message:

```
ACCOUNT NO. 4672 WILL BE CHANGED TO 4675 STARTING 8/25
```

A message sent by another user will show his user identification so you will know who sent you the message.

To find out how to send or receive broadcast messages, refer to the section "Starting and Ending a Terminal Session".

The Attention Interruption

The attention interruption allows you to interrupt processing of your job so that you can enter a new command or subcommand. The ability to interrupt processing prevents you from being "locked out" by the system while a long-running program executes or while voluminous output is displayed at your terminal. You can use the attention interruption for access to the system at any time.

When you enter an attention interruption the system cancels processing and sends you a mode message. If the system was processing a command, the mode message it prints is:

```
READY
```

You can then enter a new command. If the system was processing a subcommand, the mode message will be the name of the command to which the subcommand belongs:

```
ACCOUNT
EDIT
OPERATOR
OUTPUT
TEST
```

If you do not want to enter another subcommand, you should enter another attention interruption which will cause the READY message to appear.

There are two ways to cause an attention interruption:

1. Press the attention key which can be any of the following:

   - ATTN key on an IBM 2741 Communications terminal.
   - LINE RESET key on an IBM 1052 Printer-Keyboard. (If the "proceed" light is on, press the ALTERNATE CODING and "6" keys instead of the LINE RESET key.)

- BREAK key on a Teletype terminal.

If the attention key is also the line-deletion character and you have entered any characters in a line of input, you must press it twice to enter an attention interruption. (You need only press it once if you have not entered any characters in the line.)

2. Use a simulated attention key:

If your terminal does not have a key that can be used for attention interruption, you can use the facilities of the TERMINAL command to simulate the attention key. The TERMINAL command lets you specify a string of characters, such as HALT or ATTN, that when entered as a line of input is interpreted by the system as a request for an attention interruption. The TERMINAL command also lets you request an interruption at specified intervals while output is being produced. The TERMINAL command is described in the section, "Starting and Ending a Terminal Session".

The HELP Command

The HELP command provides you with information about all other TSO commands. At the most general level you can enter:

```
help
```

and receive a list of all commands and a brief explanation of their functions.

If you want all the information available on a specific command, for example CALL, enter the HELP command and use the other command's name as an operand:

```
help call
```

If you want to know only the function, syntax, or operands, of the CALL command, enter one of the following:

```
help call function
help call syntax
help call operands
```

You can also obtain the same information for the subcommands of the ACCOUNT, EDIT, OPERATOR, OUTPUT and TEST commands. To do this, enter the command with any required operands and wait for the mode message. After you have received it, you can enter:

```
help
```

and receive a list of all subcommands.

If you want all the available information on a given subcommand, enter the HELP command and use the subcommand name as an operand. For example, the following sequence could be used to obtain all the information available on the DISPLAY subcommand of the OPERATOR command:

```
READY
operator
OPERATOR
help display
```

If you want to know only the function, syntax, or operands of the DISPLAY subcommand you would enter one of the following:

```
help display function
help display syntax
help display operands
```

There is one restriction on using the HELP command; you cannot use it before you use the LOGON command. As it is explained in the section "Starting and Ending a Terminal Session", LOGON must be the first command used in your session because it identifies you as an authorized user of the system.

Note: Your installation can add "help" information to the system by following the instructions in the publication IBM System/360 Operating System: Time Sharing Option Guide to Writing a Terminal Monitor Program or a Command Processor, GC28-6764.

Starting and Ending a Terminal Session

This section describes the commands you can use to:

- Identify yourself to the system.
- Define operational characteristics of your session.
- Receive and send broadcast messages.
- Display session time used.
- End your terminal session.

Identifying Yourself to the System

The first thing you must do to start your terminal session is to turn on the power according to instructions provided by your installation. In many cases, you will find an instruction sheet such as the one shown in Figure 1 attached to the terminal. In the example shown in Figure 1, instructions 1 through 8 must be followed to turn on the power and to establish the connection with the system. If there is no instruction sheet attached to the terminal, consult the publication, IBM System/360 Operating System: Time Sharing Option, Terminals.

After you turn on the power you must use the LOGON command to identify yourself to the system. You must supply, as operands of LOGON, the user attributes assigned to you by your installation. Your user attributes are:

1. User identification (required) -- The name or code by which you are known to the system.

2. Password (required if your installation assigns you one) -- A further identification used for additional security protection.

3. Account number (optional) -- The account to which your terminal session is charged.

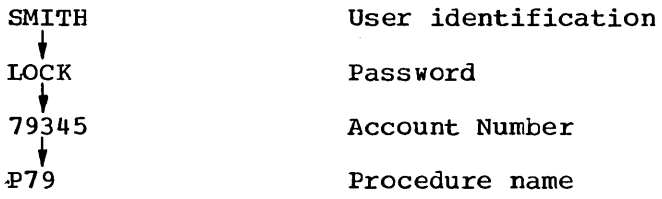
4. Procedure name (optional) -- The name of series of statements that defines your job to the system.

```
                              TERMINAL #7

(Available 9:00 a.m. - 3:00 p.m.
 For additicnal time call A. Jones ext. 1234)

 1.  Turn ON/OFF switch to CN.

 2.  Make sure the COM/LCL switch is set to COM.

 3.  Remove handset from telephone (data set).

 4.  Press TALK button on telephone.

 5.  Dial ext.  5555, 5556, or 5557.

 6.  Wait for a high pitched tone.  When ycu hear this tone you
     are in contact with the computer.  If you get a busy signal
     or no answer, hang up and repeat from 3 trying another
     extension.

 7.  Push the DATA button on the telephone.  If DATA button light
     goes off at any point during session, repeat from (3).

 8.  Replace handset on the cradle.

 9.  Enter LOGON command:

     logon___/___ acct(___)   proc(___) size(___) [notices  ] [mail  ]
                                                  [nonotices] [nomail]
          userid password account  procedure  nnnn

10.  The default TERMINAL command is:

     terminal nclines noseconds noinput break nctimeout linesize (120)

     If you want to change any of the follcwing defaults use this
     TERMINAL command:

     terminal lines(   ) seconds(   ) input(   ) linesize(   )

11.  If you want to change your user profile, use the PROFILE
     command:

          [char( )  ] [line( )    ] [prompt  ] [intercom  ] [pause  ] [msgid  ]
profile   [char(bs) ] [line(attn) ] [noprompt] [nointercom] [nopause] [nomsgid]
          [nochar   ] [noline     ]

     The following operands are recommended for this terminal:
     char(bs)  and  line(attn)

Note: Please turn ON/OFF switch to OFF after you enter LOGOFF.
```

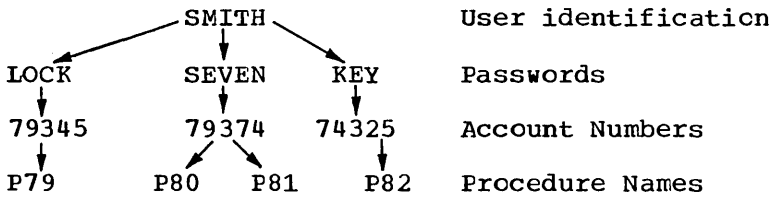
Figure 1. Sample Instruction Sheet for a Terminal

Your user attributes are recorded in the system together with the attributes of all other terminal users. When you log on, the system compares the attributes you specify with the LOGON command tc the recorded attributes of each user to determine if you are an authorized user of the system.

You can have a simple set of attributes, such as the following:



or a more complex set, such as



The latter set has three passwords (LOCK, SEVEN, and KEY) associated with your user identification. If you use the password LOCK, you can only have your processing charged only to account 79345 and you can use only procedure P79. If you use the password SEVEN, you can have your processing charged to either account 79374 or 74325. If you choose account 79374, you can use either procedure P80 or P81. If you choose account 74325, you can use only procedure P82. Another way of using procedure P82 is to choose password KEY. KEY only has account 74325 and procedure P82 associated with it.

The LOGON command is a simple means of telling the system your user identification, password, account number and procedure name. For example, if you want to use procedure P81, you must enter:

```
logon smith/seven acct(79374) proc(p81)
```

Whenever there is only one account number or procedure name associated with the user identification and password the system selects it by default. For example, account 79345 and procedure P79 are the only account and procedure associated with password LOCK. Therefore, when you log on you need only enter:

```
logon smith/lock
```

instead of:

```
logon smith/lock acct(79345) proc(p79)
```

If you choose password SEVEN, you must specify which account number you want. If you select account 74325, you do not have to specify the procedure because there is only one procedure associated with the account.

```
logon smith/seven acct(74325)
```

If you select account 79374, you must also select a procedure name because there are two procedures associated with the account. For example,

```
logon smith/seven acct(79374) proc(p80)
```

If you choose password KEY you do not have to specify an account number and procedure name because there are only one account number and one procedure name associated with KEY.

Note: In some instances your installation may require a modification in the way that you enter the LOGON command; for example, you may have to precede LOGON with a quotation mark ("LOGON). Your installation's management is responsible for advising you of such a change.

Defining Operational Characteristics

Operational characteristics can be divided into terminal characteristics and a user profile. Terminal characteristics identify:

- How you can request an attention interruption.
- Whether the keyboard is to lock up if you do not enter anything for a while.
- What the length of the line that can be displayed or printed at your terminal is.

A user profile identifies:

- What your character-deletion and line-deletion characters are.
- Whether you want to receive prompting messages.
- Whether you will accept messages from other terminals.

Your installation establishes default terminal characteristics for all the TSO terminals. If you want to change any of those characteristics for the duration of your session you can use the TERMINAL command. After your session is over the defaults selected by the installation will again be valid for the terminal. For example, assume that the default for the number of lines of continuous output that are printed before you receive an automatic interruption is 50. You can use the TERMINAL command to request that 100 lines be printed before you receive an interruption. When you log on for your next session, 50 lines will again be the default.

The system has a user profile for you. When you log on that profile will be in effect. If you want to change any item in your profile, you can do so with the PROFILE command. Any change you make becomes part of your profile. That is, the next time you log on that change will be in effect. For example, assume that the line-deletion character in your profile is a percent (%) sign. You can use the PROFILE command to change it to a number (#) sign, throughout the current session. When you log on for your next session your line deletion character will be the number sign. If you want to change it back to the original percent sign you must again use the PROFILE command.

Receiving and Sending Broadcast Messages

There are two types of broadcast messages you can receive:

- Notices -- Messages sent by the system operator to all users.
- Mail -- Messages sent by the operator or other user directly to you.

You can send messages (mail) to other users and to the system operator.

RECEIVING BROADCAST MESSAGES

You can use three commands to control which broadcast messages you receive:

LOGON, PROFILE, and LISTBC

When you log on, broadcast messages sent to all users (nctices) and those broadcast messages intended for you (mail) are displayed at your terminal. You can use the following operands of the LOGON ccmmand to prevent printing either type of messages at your terminal:

- NONOTICES suppresses printing of broadcast messages intended for all terminal users.
- NOMAIL suppresses printing of broadcast messages intended specifically for you.

For example, if you enter:

```
logon smith acct(72411) nomail
```

You will not receive mail but you will receive all notices that are available at the time.

NONOTICES and NOMAIL suppress those broadcast messages outstanding at the time you log on. You will automatically receive any broadcast messages issued after you log on. You cannot stop the operator from sending you notices, but you can specify that you do not want to receive any mail by using the NOINTERCOM operand of the PROFILE command. For example, if you enter the following commands:

```
logon jones/clcud proc(ab)
READY
profile nointercom
```

you request that all broadcast messages available at the time be displayed, but that all mail sent to you after you log on be suppressed throughout your session. (Note that NOINTERCOM can be a default of your user profile, and therefore you may not have to specify it with the PROFILE command.)

At any time during your session you can use the LISTBC command to request that either all available notices for users, or all your mail (or both) be displayed. If you enter:

```
listbc
```

you will get all broadcast messages. If you enter:

```
listbc nomail
```

you will get only notices. If you enter:

```
listbc nonctices
```

you will get only your mail.

The notices you get are both the notices available at the time you logged on and those issued throughout your session. This enables you to see what notices were available at log on time if you specified NONOTICES in your LOGON command. (The system operator can delete notices at any time. Consequently you will get only those notices he has not deleted.)

Mail messages sent directly to you are automatically deleted by the system after you receive them. Therefore the mail you get when you use the LISTBC command are those available at log on time if you specified NOMAIL in your LOGON command, and those suppressed as a result of the NOINTERCOM operand of the PROFILE command. After you use the LISTBC command to see your mail, the NOINTERCOM operand will again be in effect.

If there are no messages available when you use the LISTBC command you will receive the following message:

```
NO BROADCAST MESSAGES
```

If you want to cancel the effect of the NOINTERCOM operand, enter:

```
profile intercom
```

You will receive any mail issued after you enter this command. To obtain your mail messages issued before you entered INTERCOM, use the LISTBC command.

SENDING BROADCAST MESSAGES

You can use the SEND command to send mail messages to another terminal user or to a system operator. The SEND command can be used at any time after you log on.

You can send a mail message to another user only if you know his user identification. For example, the command:

```
send 'do not use procedure 245 until notified' user(jones,dept4)
```

will send the message enclosed in quotes to the two users whose identifications are JONES and DEPT4.

When you send a message to another user, he will receive it immediately provided that he is logged on and is accepting messages. If he is not logged on or is not accepting messages, you are notified and your message is deleted. For example, assume that SMITH is not logged on, JONES is not accepting messages, and CLARK is both logged on and accepting messages. When you send the following message:

```
send 'this is a message' user(smith,jones,clark)
```

SMITH and JONES do not receive the message, you are notified, and the message is deleted. CLARK receives the message.

You can request the system to save your message until the user you sent it to logs on or decides to accept messages, by using the LOGON operand of the SEND command. For example, if you enter:

```
send 'this is a message' user(smith,jones,clark) logon
```

SMITH will receive your message when he logs on, JONES will receive it when he uses the LISTBC command, and CLARK will receive it immediately.

You can send a message to only one operator at a time. With the SEND command, you can identify an operator by a number. For example,

```
send 'important message' operator(7)
```

If there is only one operator at your installation, you can omit the number. For example,

```
send 'important message' operator
```

If there are several operators and you omit the number, your message is sent to the main operator.

Displaying Session Time Used

You can use the TIME command to find out how much time you have used during the current session. If you enter:

```
time
```

the system sends you a message telling you how long you have been using the terminal since you logged on.

If you are executing a program, you can use the TIME command to find out how long the program has been running. You must first enter an attention interruption and then enter the TIME command. The system then sends you a message telling you how long a program has been running. If you want to continue processing the program, press the RETURN key and the program continues. If you want to stop processing the program, enter another attention interruption and wait for the READY message before you enter another command.

Ending Your Terminal Session

You can end your terminal session in either of two ways:

- By entering the LOGOFF command to end the session.
- By entering the LOGON command to start a new session.

The LOGOFF command:

- Displays your user identification.
- Displays the length of time you have been using the terminal, and the time of day and date your session ended.
- Disconnects your terminal from the system.

The LOGON command terminates your current session and starts a new session at the same time. LOGON must be specified as described in "Identifying Yourself to the System".

Entering and Manipulating Data

Almost all system applications are concerned with the processing of data. Therefore, you should learn how to enter data into the system and how to modify, store, and retrieve data after it has been entered. Any group of related data entered into the system is called a data set. For example, a data set may contain:

- Text used for information storage and retrieval.
- A source program.
- Data used as input to a program.

When you create a data set you must give it a name. The system uses the name to identify the data set whenever you want to modify or retrieve it.

The EDIT command which is used to create and manipulate data sets operates in either of two modes: input mode or edit mode. When you use the EDIT command to enter data into a data set, you are using the input mode. When you use the EDIT command to enter subcommands to manipulate the data in a data set you are using the edit mode.

In input mode, you can type a line of data and then enter it into the data set by pressing the RETURN key. You can continue entering lines of data as long as EDIT is operating in input mode. If you enter a command or subcommand while in input mode the system adds it to the data set as input data.

You can have the system assign a line number to each line as it is entered. Line numbers make edit mode operations much easier, since you can refer to each line by its own number. When you are working with a line-numbered data set, you can request the system to print out the new line number at the start of each new input line. If the data set does not have line numbers, you can request that a prompting character be displayed at the terminal before each line is entered.

After you finish entering data in the data set, you can switch to edit mode by entering a null line. (Press the RETURN key to enter a null line.)

The system lets you know you are in edit mode by printing the following message:

```
EDIT
```

In edit mode you can enter subcommands to point to particular lines of the data set, to modify or renumber lines, to add and delete lines, or to control editing of input.

When EDIT is operating in edit mode, it uses an indicator called the current line pointer to keep track of the next line of data to be processed. The operations you indicate with the subcommands are performed starting at the line indicated by the pointer. For example, the DELETE subcommand, deletes the line indicated by the pointer. After a subcommand is executed the system repositions the pointer.

You may want to reposition the pointer before a subcommand is executed. You can do so by using one of two methods: line number editing or context editing. Line number editing can be used only if your data set has line numbers. You can specify a line number as an

operand of a subcommand and the system will move the pointer to that line before it executes the subcommand. Context editing can be used for data sets with or without line numbers. A set of subcommands (UP, DOWN, TOP, BOTTOM, and FIND) allows you to move the pointer up or down a specified number of lines, or to find a line with a particular series of characters in it and move the pointer to it. After the pointer is positioned you can enter the subcommand that performs the functions you require. The subcommand can use an asterisk (*) instead of a line number to specify the line indicated by the pointer.

After you finish editing the data, you can switch to input mode by either of two methods:

1. Entering the INPUT or INSERT subcommand.

2. Entering a null line. (Press the RETURN key to enter a null line.)

The system lets you know you have selected input mode by printing the following message:

```
INPUT
```

You can terminate the EDIT command at anytime by switching to edit mode (if not already in edit mode) and entering the END subcommand. The system then prints a READY message, and you can enter any command you choose.

Note: If you want to enter a blank line in your data set, you must enter a blank by pressing the space bar, and then press the RETURN key. You can then enter other lines after the blank line. If you fail to enter a blank and press only the RETURN key, you enter a null line which causes EDIT to switch modes.

The remainder of this chapter describes how you can:

- Identify a data set.
- Create a data set.
- Place data into columns.
- Find and position the current line pointer.
- Update a data set.
- List the contents of a data set.
- Store a data set.
- End the EDIT functions.

The following functions described in this chapter are performed with commands other than EDIT:

- Rename a data set.
- Delete a data set.
- Establish passwords for a data set.
- Allocate a data set.
- Free an allocated data set.
- List the names of your data sets.

Identifying the Data Set

The EDIT command is used to specify the name of a data set and whether you want to create it or edit it. If you indicate that you are going to create a new data set, the system enters input mode. If you indicate that you are going to edit an existing data set, the system enters edit mode after you enter the EDIT command. For example, the NEW operand in

the following EDIT command specifies that you are going to create a new data set named ACCTS.DATA. After you enter the command the system enters input mode.

```
READY
edit accts.data new
INPUT
```

In the following example, the OLD operand of the EDIT command specifies that you want to edit an existing data set named PARTS.TEXT. After you enter the command, the system enters edit mode.

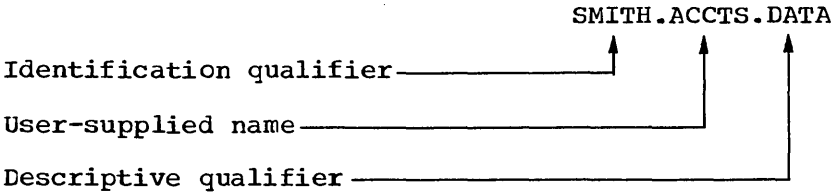
```
READY
edit parts.text old
EDIT
```

As you can see, the NEW operand specifies that you are going to create a data set, and the OLD operand specifies that the data set already exists.

The name you give a data set should follow certain conventions. A data set name has three fields.

1. Identification qualifier.
2. User-supplied name.
3. Descriptive qualifier.

The fields must be separated by periods. The total length of the name, including periods, must not exceed 44 characters. For example, a typical data set name is:



When you create a data set you need only specify the user-supplied name. The system supplies values for the other two fields. The identification qualifier is the user identification you specified with the LOGON command. The descriptive qualifier must be one of those listed in Table 2. The system obtains it from operands you specify in the EDIT command. If you do not supply it in another operand, the system prompts you for a descriptive qualifier. If you prefer, you can specify the descriptive qualifier as part of a data set name, for example,

```
PARTS.DATA
```

You may specify a fully qualified name (a name with all three qualifiers) by enclosing it in apostrophes. For example,

```
'JONES.PROG1.ASM'
```

This is a useful procedure when you have to use a data set with an identification qualifier other than your own user identification.

Table 2. Descriptive Qualifiers

Descriptive Qualifier	Data Set Contents
ASM	Assembler (F) input
BASIC	ITF: BASIC statements
FORT	FORTRAN IV (E, G, G1 or H) statements and free- or fixed-format FORTRAN statements
IPLI	ITF:PL/I statements
PLI	PL/I (F) statements
COBOL	American National Standard COBOL statements
TEXT	Uppercase and lowercase text
DATA	Uppercase text
CNTL	JCL and SYSIN for SUBMIT command
CLIST	TSO commands
STEX	STATIC external data from ITF:PLI
OBJ	Object module
LIST	Listings
LOAD	Load module
LINKLIST	Output listing from linkage editor
LOADLIST	Output listing from loader
TESTLIST	Output listing from TEST command
OUTLIST	Output listing from OUTPUT command

Any name that does not conform to the naming conventions must also be enclosed in apostrophes. For example, if you have a data set named RECORDS, with no identification or descriptive qualifiers, enter

```
'records'
```

The system will not append the identification and descriptive qualifiers to such a name.

You can refer to an existing data set by its user-supplied name. In some cases, you may also have to include the descriptive qualifier. For example, if two of your data sets were named:

```
SMITH.PART1.ASM
SMITH.PART1.DATA
```

and you want to refer to the latter, you must specify:

```
part1.data
```

You can also create and edit partitioned data sets. A partitioned data set consists of one or more data sets called members. Each member can be created and edited separately and each has a name. The member name is enclosed in parentheses and appended to the right of the fully qualified data set name. For example, the fully qualified name of member MEM1 of the SMITH.PART1.DATA data set is:

```
SMITH.PART1.DATA(MEM1)
```

You need only use the user-supplied name and member name to refer to the member. The system appends the identification and descriptive qualifiers. For example, to refer to member MEM1 you can specify:

```
part1(mem1)
```

In the following example you use the EDIT command to create member ONE of a partitioned data set named JONES.T42.DATA. The second EDIT command, creates member TWO of JONES.T42.DATA. Note that the NEW operand must be specified in both cases. The third EDIT command, specifies that changes are to be made to member ONE.

```
READY
edit t42.data(one) new
INPUT
.
.
.
READY
edit t42.data(two) new
INPUT
.
.
.
READY
edit t42.data(one) old
EDIT
.
.
.
```

After you specify the data set name and the NEW or OLD operand, you should specify the data set type. The data set type is an operand that describes the purpose for which the data set is to be or was created. The type operand is one of the sources from which the system can obtain the descriptive qualifier. The valid types are:

```
ASM
COBOL
GOFORT
FORT
FORTE
FORTG
FORTH
PLI
PLIF
IPLI
BASIC
DATA
TEXT
CLIST
CNTL
```

You do not have to specify the type operand if you specify it as the descriptive qualifier. For example, the following two commands have the same effect:

```
edit ab75 new asm
edit ab75.asm new
```

If the system cannot find the data set type from other sources, you are prompted for it.

If you do not want your data set to have line numbers, use the NONUM operand. For example,

```
edit ab75 new asm nonum
```

Do not specify NONUM for the BASIC, IPLI, and GOFORT data set types, because they must always have line numbers.

Except for one case, lines of input are translated to uppercase letters by the system. If you want the system to retain your input in the same form as you enter it (uppercase and lowercase), code the ASIS operand. For example,

```
edit records new data asis
```

Creating a Data Set

You usually create a data set when EDIT is in input mode. You request input mode when you enter one of the following:

- The NEW operand in the EDIT command.
- The INPUT subcommand while you are in edit mode.

After you enter the EDIT command with the NEW operand the system sends you the following message:

```
INPUT
```

After this message is printed the system prints the first line number of your data set unless you specified NONUM in the EDIT command. The first line number printed is 00010. Type the first line of input to the right of the line number and press the RETURN key to enter it. The system then prints the second line number, which is 00020, and you may then enter your second line of input, and so on. When you reach the end of the data you want to enter, press the RETURN key without entering anything (a null line) and the system switches to edit mode. The following example illustrates the points just discussed:

```
READY
edit accts new data
INPUT
00010     #23942      5     @2.75     acme inc
00020     #32135     21     @3.90     bbb corp
00030     #32174     12     @1.80     alpha inds
00040     #49213     35     @7.95     xyz dist
00050     #52221     50     @2.35     beta mfg
00060  (null line)
EDIT
```

In the example, the line numbers have the standard increment of 10. If you prefer a different increment, you can use the INPUT subcommand to create the data set. To do this you must first request a switch to edit mode by entering a null line after you receive the INPUT message. Then enter the INPUT subcommand specifying the number of the first line and the size of the increment. After entering the INPUT subcommand the system switches to input mode and prompts you with the first line number. For example, to start with line 5 and use increments of 5, you could use the following sequence:

```
READY
edit accts new data
INPUT
00010  (null line)
EDIT
input 5 5
INPUT
00005     #23942      5     @2.75     acme inc
00010     #32135     21     @3.90     bbb corp
00015     #32174     12     @1.80     alpha inds
00020     #49213     35     @7.95     xyz dist
00025     #52221     50     @2.35     beta mfg
00030  (null line)
EDIT
```

You can create the same data set in edit mode. However, you must enter the line numbers you wish to use.

```
READY
edit accts new data
INPUT
00010  (null line)
EDIT
5         #23942      5     @2.75     acme inc
10        #32135     21     @3.90     bbb corp
15        #32174     12     @1.80     alpha inds
20        #49213     35     @7.95     xyz dist
25        #52221     50     @2.35     beta mfg
```

Note: Requesting an increment larger than 1, makes it easier for you to insert lines in your data set later on. (See the section "Updating a Data Set" for instructions on how to insert lines in your data set.)

Placing Data into Columns

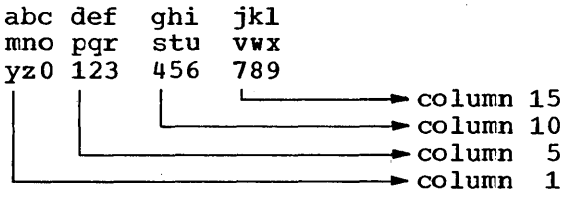
You can use the TAB key of your terminal to align your data in columns, just as you would with an ordinary typewriter. However, this mechanical tab setting is not recognized by the system which interprets each striking of the TAB key as a space. For example, if you enter the following three lines and align them with the TAB key, they appear at the terminal as follows:

```
39427        abcde        49211        72669        ab4
22           fghijkl      441          123456       72de
987654       mnop         2            31           xyz
```

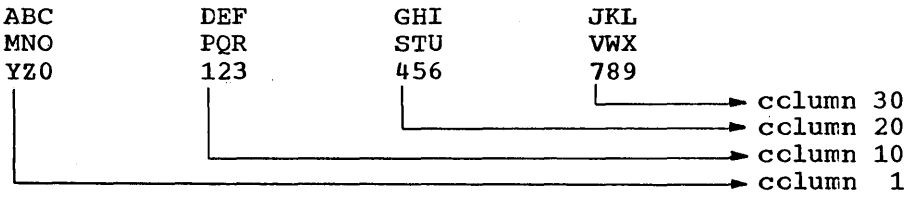
but they are received by the system as follows:

```
39427 ABCDE 49211 72669 AB4
22 FGHIJKL 441 123456 72DE
987654 MNOP 2 31 XYZ
```

If you want the system to place your data into columns, you must establish logical tab settings with the TABSET subcommand of the EDIT command or else use the defaults provided by the system. If you have established logical tab settings for your data set, the system will receive each item in its proper column whenever you press the TAB key. The mechanical tab settings in your terminal need not correspond to the logical tab settings. For example, assume that the logical tab settings for the data set are columns 10, 20, and 30, while the mechanical tab settings in the terminal are columns 5, 10 and 15. When you type in the following three lines using the TAB key:



they are received by the system as follows:



You may find it convenient to make the mechanical tab settings coincide with the logical tab settings.

The default tab settings used by the system vary with the data set type. They are shown in Table 3.

Table 3. Default Tab Settings

Data Set Type	Default Tab Setting Columns
COBOL	8,12,72
PLI	5,10,15,20,25,30,35,40,45,50
PLIF	5,10,15,20,25,30,35,40,45,50
FORT	7,72
ASM	10,16,31,72
TEXT	5,10,15,20,30,40
DATA	10,20,30,40,50,60
CLIST	10,20,30,40,50,60
CNTL	10,20,30,40,50,60
IPLI	5,10,15,20,25,30,35,40,45,50
BASIC	10,20,30,40,50,60

If you want to change the default settings or other settings you previously established, or nullify all tabs, you must use the TABSET subcommand. If you want to change the default settings, you will probably do so before you create the data set. That means you must request edit mode after you enter the EDIT command, then enter the TABSET subcommand and return to the input mode to create the data set. For example, if you want to create a TEXT data set with the logical tabs at columns 10, 25, and 35, you can use the following sequence:

```
READY
edit series new text
INPUT
00010 (null line)
EDIT
tabset 10 25 35
      (null line)

INPUT
00010
```

If you prefer, you can define the tab settings with a line that consists of blanks and t's. For example, to establish tab settings in columns 10, 25, and 35 you can use the TABSET subcommand as follows:

```
tabset image
         t              t         t
```

You must produce the spaces between t's by pressing the space bar as many times as necessary. Do not use the TAB key with mechanical tabs to produce those spaces.

If you want to nullify the existing tab settings for the data set, enter the TABSET subcommand as follows:

```
tabset off
```

Finding and Positioning the Current Line Pointer

Unless you plan to use line numbers for all your edit operations, you should know how to find and reposition the current line pointer. These operations are described in the following paragraphs.

FINDING THE CURRENT LINE POINTER

The location of the current line pointer is determined by the last subcommand you entered. Table 4 shows the location of the pointer at the end of each subcommand. If you do not remember this information, you can use the LIST subcommand with the * operand to find the line at which the pointer is positioned. For example:

```
list *
USER,HOWEVER,MUST SUPPLY A DATA SET
```

You can also have the system display the line at which the pointer is positioned everytime the pointer changes as a result of the CHANGE, TOP, BOTTOM, UP, DOWN, FIND and DELETE subcommands. To do this use the VERIFY subcommand as follows:

```
verify
```

The VERIFY subcommand is in effect until you enter it again with the OFF operand:

```
verify off
```

Table 4. Values of the Line Pointer Referred to by an Asterisk (*)

Edit Subcommands	Value of the Pointer at Completion of Subcommand
BOTTOM	Last line (or line zero for empty data sets)
CHANGE	Last line changed
DELETE	Line preceding deleted line, if any, else zero
DOWN	Line after last line referred to (or line zero for empty data sets)
END	No change
FIND	Found line, if any, else no change
HELP	No change
INPUT	Last line entered
INSERT	Last line entered
Insert/Replace/Delete	Inserted line or line preceding the deleted line
LIST	Last line listed
PROFILE	No change
RENUM	Same relative record
RUN	No change
SAVE	No change
SCAN	Last line referred to, if any
TABSET	No change
TOP	Zero value
UP	Line before last line referred to (or line zero for empty data sets)
VERIFY	No change

POSITIONING THE CURRENT LINE POINTER

You can use the UP, DOWN, TOP, BOTTOM and FIND subcommands to move the current line pointer.

The UP subcommand moves the pointer a specified number of lines towards the beginning of your data set. For example, to move the pointer so that it refers to a line located five lines before the location currently referred to, enter:

```
up 5
```

The DOWN subcommand moves the pointer a specified number of lines towards the end of your data set. For example, to move the pointer so that it refers to a line located 12 lines after the location currently referred to, enter:

```
down 12
```

The TOP subcommand moves the pointer to the position preceding the first line of your data set. TOP is often used in combination with the DOWN subcommand. For example, if you want the pointer to refer to the third line of your data set, use the following sequence:

```
top
down 3
```

The BOTTOM subcommand moves the pointer to the last line of the data set.

The FIND subcommand moves the pointer to a line that contains a specified sequence of characters. For example, to move the pointer to the line that contains PLACED BEFORE ENTRY enter:

```
find xplaced before entry
```

The "x" inserted before "placed" is a special delimiter that marks the beginning of the sequence of characters the system has to search for. The special delimiter can be any character other than a number, semicolon, blank, tab, comma, parenthesis, asterisk, or one of the characters in the sequence you want to find. The special delimiter must be placed next to the first character of the sequence you want to find. Do not insert a blank between the special delimiter and the first character.

If you prefer, you can have the system start the search for the sequence of characters starting at the same column of each line. For example, if you want the search for PLACED BEFORE ENTRY to start in column seven of each line, enter:

```
find xplaced before entry x7
```

Note that the same special delimiter used at the beginning of the sequence of characters must also precede the column number.

The FIND subcommand starts looking for the sequence of characters beginning with the line at which the pointer is located. Therefore, unless you are sure the characters are in a line following the one indicated by the pointer, you should use the TOP subcommand to move the pointer to the beginning of the data set. For example:

```
top
find xplaced before entry
```

Figure 2 shows a data set used to illustrate the examples of positioning the current line pointer. Although this data set has line numbers, they are not used in the examples.

```
00010              TEMPERATURE DATA FOR 7/29/70
00020              HIGHEST, 90 AT 12:30 P.M.
00030              LOWEST, 73 AT 5:40 A.M.
00040              MEAN, 83
00050              NORMAL ON THIS DATE, 77
00060              DEPARTURE FROM NORMAL, +6
00070              HIGHEST TEMPERATURE THIS DATE, 99 IN 1949
00080              LOWEST TEMPERATURE THIS DATE, 59 IN 1914
00090              TEMPERATURE HUMIDITY INDEX, 81
```

Figure 2. Sample Data Set

Assume that you do not know the present location of the current line pointer, and would like to move it to the fifth line (00050). Enter:

```
top
down 5
```

To move the pointer from the fifth line (00050) to the third line (00030), enter:

```
up 2
```

To move the pointer to the line that contains FROM NORMAL enter:

```
find xfrom normal
```

To move the pointer to the last line (00090), enter:

```
bottom
```

Updating a Data Set

The subcommands of the EDIT command allow you to update a data set. That is, they allow you to:

- Delete data from a data set.
- Insert data in a data set.
- Replace data in a data set.
- Renumber lines of a data set.

These functions are described in the following paragraphs.

DELETING DATA FROM A DATA SET

If you want to delete only one line of data you do not need a subcommand. Indicate only the line number or an asterisk. For example, if you want to delete line 30, enter:

```
30
```

If you want to delete the line indicated by the current line pointer, enter:

```
*
```

You can also use the DELETE subcommand to perform the same function. For example,

```
delete 30
```

or

```
delete *
```

DELETE also allows you to delete more than one consecutive line. To do so you can specify the line numbers of the first and last lines to be deleted, or the number of lines to be deleted starting with the line indicated with the current line pointer. For example, if you want to delete all the lines between, and including lines 15 and 75, enter:

```
delete 15 75
```

If you want to delete 12 lines starting with the line indicated by the current line pointer, enter:

```
delete * 12
```

INSERTING DATA IN A DATA SET

To insert only one line of data you do not need a subcommand; indicate only the line number. The line number referred to should not exist. For example, if you want to insert "RECORDED DAILY IN CENTRAL" in line 22, enter:

```
22 recorded daily in central
```

The characters you want to enter must be separated from the line number or the asterisk by one blank or one comma. Any additional blanks or commas are considered to be part of the input data.

To insert more than one line, use the INSERT or INPUT subcommands. INSERT can be used only for data sets without line numbers. INPUT can be used for data sets with or without line numbers.

The INSERT subcommand inserts one or more lines of data following the location pointed to by the current line pointer. For example, assume that you have the following data set:

```
A. CARSON      DEPT A72
T. DANIELS     DEPT 795
C. DICKENS     DEPT 981
R. EMERSON     DEPT 245
E. FARRELL     DEPT B32
C. LEVI        DEPT 229
D. MADISON     DEPT D49
```

To insert three lines after the entry for E. FARRELL and before the entry for C. LEVI you must first position the current line pointer at the fifth line. Your listing would look like this:

```
EDIT
top
down 5
insert
INPUT
e. glotz dept 741
p. henry dept 333
h. hill  dept R92
      (null line)
EDIT
```

You must enter a null line to indicate the end of your input.

The INPUT subcommand is used in a manner similar to the INSERT subcommand if your data set does not have line numbers. Use an asterisk in the INPUT subcommand to indicate that the lines of input that follow are to be inserted in the location following the current line pointer. Using the preceding example, to insert the lines:

```
E. GLOTZ       DEPT 741
P. HENRY       DEPT 333
H. HILL        DEPT R92
```

after the line for E. FARRELL and before the line for C. LEVI, your listing would look like this:

```
EDIT
top
down 5
input *
INPUT
e. glotz dept 741
p. henry dept 333
h. hill  dept R92
      (null line)
EDIT
```

Note that after entering the INSERT or the INPUT subcommand EDIT switches to input mode.

If your data set has line numbers, you can use the INPUT subcommand to insert one or more lines of data between two existing lines of the data set. You can also indicate a smaller increment for the new line numbers so that they fit between the line numbers of the existing lines. For example, assume you have the following data set:

```
00010     1932     $1.50
00020     2579     $1.39
00030     4798     $1.75
00040     5344     $2.49
```

To insert three lines between lines 20 and 30, to have the first line numbered 22, and to increment this number by two in the following lines, your listing would look as follows:

```
EDIT
input 22 2
INPUT
00022     2795     $0.79
00024     3241     $2.81
00026     4152     $1.79
00028     (null line)
EDIT
```

The updated data set would look like this:

```
00010     1932     $1.50
00020     2579     $1.39
00022     2795     $0.79
00024     3241     $2.81
00026     4152     $1.79
00030     4798     $1.75
00040     5344     $2.49
```

If you do not change the increment, and there is no room for the new lines, you receive an error message. If you wish, you can renumber the lines of your data set. This procedure is explained in the section "Renumbering Lines of Data".

To enter lines at the end of the data set, enter the INPUT subcommand without operands. If the data set has line numbers you will be prompted with the line number. For example,

```
EDIT
input
INPUT
00050  6211  $3.95
00060  7199  $0.85
00070  (null line)
EDIT
```

REPLACING DATA IN A DATA SET

You can replace an entire line, or a sequence of characters in a line or in a range of lines.

If you are only replacing one line of data, you do not need a subcommand. Indicate only the line number or an asterisk. For example, if you want to replace the contents of line 70 with "SEVERAL REPORTS WERE MADE", enter:

```
70 several reports were made
```

If you want to replace the contents of the line indicated by the current line pointer, enter:

```
* several reports were made
```

The characters you want to enter must be separated from the line number or the asterisk by only one blank or a comma. Any additional blanks or commas are considered to be part of the input data.

You can also replace lines of data when you use the INPUT subcommand. If you use the R operand, the lines starting with the line indicated by the line number or the asterisk are replaced by the lines you enter. For example, assume that you have the following data set:

```
COMPLETION SCHEDULE
STAGE 1   7/19
STAGE 2   8/15
STAGE 3   9/29
```

To replace the third and fourth lines, you must first position the current line pointer at the third line.

```
EDIT
top
down 3
input * r
INPUT
stage 2   8/21
stage 3   9/15
      (null line)
EDIT
```

Your updated data set would look like this:

```
COMPLETION SCHEDULE
STAGE 1   7/19
STAGE 2   8/21
STAGE 3   9/15
```

In the following example, assume that the data set has line numbers:

```
00010  COMPLETION SCHEDULE
00020  STAGE 1  7/19
00030  STAGE 2  8/15
00040  STAGE 3  9/29
```

To replace lines 30 and 40, your listing should look as follows:

```
EDIT
input 30 r
INPUT
00030  stage 2  8/21
00040  stage 3  9/15
00050       (null line)
EDIT
```

Your updated data set will look as follows:

```
00010  COMPLETION SCHEDULE
00020  STAGE 1  7/19
00030  STAGE 2  8/21
00040  STAGE 3  9/15
```

If the data set has line numbers, you can replace a line and insert additional lines. For example, assume the same data set:

```
00010  COMPLETION SCHEDULE
00020  STAGE 1  7/19
00030  STAGE 2  8/15
00040  STAGE 3  9/29
```

To replace line 30 and insert two lines with a line increment of 2, your listing should look as follows:

```
EDIT
input 30 2 r
INPUT
00030  stage 2  part 1  8/15
00032  stage 2  part 2  8/21
00034  stage 2  part 3  9/15
00036         (null line)
EDIT
```

Your updated data set will look as follows:

```
00010  COMPLETION SCHEDULE
00020  STAGE 1  7/19
00030  STAGE 2  PART 1  8/15
00032  STAGE 2  PART 2  8/21
00034  STAGE 2  PART 3  9/15
00040  STAGE 3  9/29
```

To replace more than one line with a greater number of lines, you can also use the DELETE subcommand to delete those lines and then use either INPUT or INSERT to insert the replacement lines. Use this procedure when the data set does not have line numbers.

Use the CHANGE subcommand to change only part of a line or lines. For example, to change the characters "DAILY INVENTORY" to "WEEKLY REPORT" in line 12 of your data set, enter:

```
change 12/daily inventory/weekly report/
```

The "/" placed before the characters to be changed and the replacement characters is a special delimiter that marks the beginning of those sequences of characters. The special delimiter can be any character other than a number, blank, tab, comma, semicolon, parentheses, or asterisk. Make sure the character you select as a special delimiter does not appear in the sequence of characters you specify. If you leave blanks between the last character to be replaced

and the special delimiter for the replacement characters, the blanks are considered part of the characters to be replaced.

Instead of using a line number you can use an asterisk. For example if the change is to be made to the line indicated by the current line pointer, enter:

```
change * xdaily inventoryxweekly reportx
```

You can have the system search for a sequence of characters in a range of lines rather than in one line. You can indicate the range of lines by giving the numbers for the first and last lines of the range, or by indicating the current line pointer and the number of lines you want to have searched. For example, if the characters "DAILY INVENTORY" appear somewhere between lines 15 and 19, enter:

```
change 15 19 !daily inventory!weekly report!
```

If the characters appear within the 10 lines starting with the one indicated by the current line pointer, enter:

```
change * 10 ?daily inventory?weekly report?
```

You can change the sequence of characters every time it appears within the range of lines. To do this specify the ALL operand after the replacement sequence. ALL must be preceded by the same special delimiter that precedes the character sequences. For example,

```
change 15 19 !daily inventory!weekly report!  all
```

or

```
change * 10 !daily inventory!weekly report!  all
```

If you wish, you can have the system locate a sequence of characters in a line and print that line up to those characters. You can then type new characters to complete the line and enter the new line when you press the RETURN key. For example, assume that you want to change the characters "TUESDAY" to "THURSDAY" in the following line:

```
00015 PARTS DELIVERIES ARE MADE ON TUESDAY
```

Your listing will look as follows:

```
change 15/tuesday
00015 PARTS DELIVERIES ARE MADE ON thursday
```

If the characters you want to change are in the line indicated by the current line pointer, your listing would look like this:

```
change */tuesday
00015 PARTS DELIVERIES ARE MADE ON thursday
```

You can also request that the system print out the first few characters of a given line. Then you can enter the characters you want to replace the remaining characters in the line. For example, you can request that the first 29 characters of the line "PARTS DELIVERIES ARE MADE ON TUESDAY" be printed:

```
change 15 29
00015 PARTS DELIVERIES ARE MADE ON thursday
```

You must use line numbers for this type of reference. You can also have the system print the first characters of a range of lines. This is particularly useful when you want to change a column in a table. For example, assume that you have the following data set:

```
00010  ENROLLMENT DATES
00012  P. JONES  MAY 15  JUNE 12
00014  A. SMITH  MAY 31  JULY 19
00016  J. DOE    JUNE 7  JULY 17
00018  B. GREEN  JUNE 9  AUGUST 3
```

If you want to change the data in the last column, which begins in position 17, enter:

```
change 10 18 17
00010  ENROLLMENT DATES
00012  P. JONES  MAY 15  june 25
00014  A. SMITH  MAY 31  july 23
00016  J. DOE    JUNE 7  july 31
00018  B. GREEN  JUNE 9  august 10
```

RENUMBERING LINES OF DATA

You can use the RENUM subcommand to assign line numbers to a data set without line numbers, or to renumber the lines of a data set with line numbers. If you enter:

```
renum
```

the system assigns new line numbers to all the lines of the data set beginning with number 10 and incrementing the following line numbers by 10.

You can assign a number to the first line of the data set. For example if you want the first line to have number 5, enter:

```
renum 5
```

The remaining line numbers will be 15,25,35, etc.

You can specify an increment other than 10 in addition to the number of the first line. For example if you want the first line to be number one, and the remaining line numbers to increase by 3, enter:

```
renum 1 3
```

If your data set already has line numbers you can specify that renumbering is to start at a given line. You must also specify the new number for this line (which must be equal to or greater than the old line number) and the increment. For example, if you want to start renumbering at line 23, and the new line number is to be 25 and the increment is to be 5, enter:

```
renum 25 5 23
```

Listing the Contents of a Data Set

The LIST subcommand allows you to display the contents of a data set at your terminal. To list the entire contents of the data set, enter:

```
list
.
.  your data set is listed here
.
```

To list a group of lines, enter the number of the first and last lines of the group. For example, to list lines 20 through 110 of the data set, enter:

```
list 20 110
```

If your data set does not have lines numbers you can use the current line pointer and the number of lines to be listed. For example, to list the 20 lines that begin with the line indicated by the pointer enter:

```
list * 20
```

To list only one line, indicate the line number or the current line pointer. For example, if you wish to list line 22, enter:

```
list 22
```

If you want to list the line pointed at by the current line pointer, enter:

```
list *
```

You can use the SNUM operand to suppress listing the line numbers of a line-numbered data set. (If your data set does not have line numbers, this operand has no effect.) For example, any of the following commands produces a listing of the lines indicated without their line numbers:

```
list snum
list 20 110 snum
list * 20 snum
list 22 snum
list * snum
```

Storing a Data Set

The data set you have created or the changes you made to a previously existing data set are retained by the system only until you finish using the EDIT command and its subcommands. That is, as soon as you notify the system that you want to use another command and you get a READY message, your newly created data set, or your new set of changes, is discarded. If you want the system to make your new data set a permanent data set, or if you want the system to incorporate your changes into the existing data set, you must use the SAVE subcommand of the EDIT command.

For example, in the following sequence ycu create a data set named RECORDS and ask the system to store it as a permanent data set:

```
READY
edit records new data
INPUT
00010  project 21   7/10-8/25   a. jones
00020  project 23   7/10-9/12   p. smith
00030  project 39   8/1-9/15    r. brown
00040    (null line)
EDIT
save
SAVED
end
READY
```

In the following sequence you add a line to the RECORDS data set and ask the system to make it part of the data set:

```
READY
edit records old data
EDIT
40  project 42  8/15-9/21  s. green
save
SAVED
end
READY
```

In some cases you may want to preserve the existing data set intact and have the system make the changes to a data set that is a copy of the original data set. To do this you must enter a new data set name for the copy when you enter the SAVE subcommand. For example, if you want to keep the RECORDS data set intact, and you want your changes to be made to a copy of RECORDS named PROJS, use the following sequence:

```
READY
edit records old data
EDIT
40  project 42  8/15-9/21  s. green
save projs
SAVED
end
READY
```

Now you have two data sets. The one named RECORDS looks like this:

```
00010  PROJECT 21   7/10-8/25   A. JONES
00020  PROJECT 23   7/10-9/12   P. SMITH
00030  PROJECT 39   8/1-9/15    R. BROWN
```

The data set named PROJS looks as follows:

```
00010  PROJECT 21   7/10-8/25   A. JONES
00020  PROJECT 23   7/10-9/12   P. SMITH
00030  PROJECT 39   8/1-9/15    R. BROWN
00040  PROJECT 42   8/15-9/21   S. GREEN
```

You can use the SAVE subcommand whenever you are using the EDIT command. For example, you can create a data set and save it. Then you can start making changes to the data set and once you are satisfied with those changes you can save them to make them part of the data set. For example, in the following sequence you create a data set, save it, replace line 30, insert three lines after line 50, list the data set, delete line 56, renumber the data set, and save it.

```
READY
edit phones new text
INPUT
00010           telephone listing - sales dept
00020           j. adams      1291
00030           c. allan      2431
00040           a. bailey     3255
00050           b. crane      4072
00060           e. foster     1384
00070           f. graham     2291
00080           d. murphy     9217
00090
EDIT
save
SAVED
30              c. alden      2441
input 52 2
INPUT
00052           l. davis      4119
00054           j. egan       6835
00056           e. foster     1384
EDIT
list


00010           TELEPHONE LISTING - SALES DEPT
00020           J. ADAMS      1291
00030           C. ALDEN      2441
00040           A. BAILEY     3255
00050           B. CRANE      4072
00052           L. DAVIS      4119
00054           J. EGAN       6835
00056           E. FOSTER     1384
00060           E. FOSTER     1384
00070           F. GRAHAM     2291
00080           D. MURPHY     9217
delete 56
renum
save
SAVED
end
READY
```

Ending the EDIT Functions

Use the END subcommand to terminate the operation of the EDIT command. If you have made changes to your data set and have not entered the SAVE subcommand, the system will ask you if you want to save the modified data set. If so you can enter the SAVE subcommand. If you do not want to save the changes, reenter the END subcommand.

After you enter the END subcommand you receive the READY message. You can then enter any command you choose.

Renaming a Data Set

The RENAME command allows you to:

- Change the name of a data set.
- Change the name of a member of a partitioned data set.
- Assign an alias to a member of a partitioned data set.

If you have a data set named SMITH.RECPT.DATA and you want to change it to SMITH.ACCT.DATA, you can do so with any of the following RENAME commands:

```
rename 'smith.recpt.data' 'smith.acct.data'
rename recpt.data acct.data
rename recpt acct
```

Note that the fully qualified name must be enclosed in apostrophes.

The simple user-supplied name can be used if you have only one data set with that name. For example, if you have two data sets named SMITH.RECPT.DATA and SMITH.RECPT.TEXT, you must specify either RECPT.DATA or 'SMITH.RECPT.DATA' in the RENAME subcommand. If you do not specify the descriptive qualifier, the system will prompt you for it.

The following examples show how you can use RENAME to change the identification qualifier or the descriptive qualifier.

```
rename 'smith.acct.data' 'jones.acct.data'
rename acct.data acct.text
```

The following examples show how you can change more than one qualifier.

```
rename 'smith.acct.data' 'jones.recpt.text'
rename acct.data recpt.text
```

When changing the name of a member of a partitioned data set, you must specify the existing data set name and member name and the new member name. For example, to change the name of a member of JONES.AB79.DATA from INPUT to ENTRY, you can do so with any of the following commands:

```
rename 'jones.ab79.data(input)' (entry)
rename ab79.data(input) (entry)
rename ab79(input) (entry)
```

Use the ALIAS operand to indicate that the new member name is an alias and not a replacement. For example to assign the alias DAILY to member INPUT of JONES.AB79.DATA, use any of the following:

```
rename 'jones.ab79.data(input)' (daily) alias
rename ab79.data(input) (daily) alias
rename ab79(input) (daily) alias
```

After entering this command the name of the member is either JONES.AB79.DATA(INPUT) or JONES.AB79.DATA(DAILY).

Sometimes you may have two or more data set names that are identical in all but one of their qualifiers. For example, you may have these data sets:

```
JONES.ALPHA.DATA
JONES.BETA.DATA
```
or
```
JONES.ALPHA.DATA
JONES.ALPHA.ASM
```
or
```
JONES.ALPHA.DATA
SMITH.ALPHA.DATA
```

You can use the RENAME command to replace one or both of their common qualifiers. For example, you may want to change the group:

```
JONES.ALPHA.DATA
JONES.BETA.DATA
```
to
```
JONES.ALPHA.TEXT
JONES.BETA.TEXT
```
or to
```
SMITH.ALPHA.DATA
SMITH.BETA.DATA
```
or to
```
SMITH.ALPHA.TEXT
SMITH.BETA.TEXT
```

In order to make the change, replace the dissimilar qualifier with an asterisk. For example,

```
jones.*.data
```

stands for "all data sets whose identification qualifier is JONES and whose descriptive qualifier is DATA". You can then enter the RENAME command:

```
rename *.data *.text
```

to change the group

```
JONES.ALPHA.DATA
JONES.BETA.DATA
```

to

```
JONES.ALPHA.TEXT
JONES.BETA.TEXT
```

Enter the command

```
rename 'jones.*.data' 'smith.*.data'
```

to change the group

```
JONES.ALPHA.DATA
JONES.BETA.DATA
```

to

```
SMITH.ALPHA.DATA
SMITH.BETA.DATA
```

Enter the command

```
rename 'jones.*.data' 'smith.*.text'
```

to change the group

```
JONES.ALPHA.DATA
JONES.BETA.DATA
```

to

```
SMITH.ALPHA.TEXT
SMITH.BETA.TEXT
```

Deleting a Data Set

Use the DELETE command to delete one or more data sets or one or more members of a partitioned data set.

If you have a data set named BROWN.INPUT.TEXT and you want to delete it, enter

```
READY
delete input
READY
```

If you have two data sets named BROWN.INPUT.TEXT and BROWN.DAYS.DATA and you want to delete them, enter:

```
READY
delete (input days)
READY
```

If you want to delete member FIRST of the MARY.ALPHA.ASM partitioned data set enter:

```
READY
delete alpha(first)
READY
```

If member FIRST has the alias LAST, and you want to delete both the member name and its alias, enter:

```
READY
delete alpha(first) alpha(last)
READY
```

You may have a group of data sets whose names differ only in the user-supplied name or in the descriptive qualifier. For example,

```
HELEN.LIST.DATA
HELEN.LINES.DATA
HELEN.DATES.DATA
```

or

```
LUCY.WEATHER.ASM
LUCY.WEATHER.DATA
LUCY.WEATHER.TEXT
```

To delete the entire group, place an asterisk in the position where the names do not match. (The asterisk cannot replace the user identification.) For example, to delete the first group use the following:

```
READY
delete *.data
READY
```

To delete the second group use the following:

```
READY
delete weather.*
READY
```

Establishing Passwords for a Data Set

Use the PROTECT command to establish passwords for your data set. Passwords prevent unauthorized persons from reading (listing) or writing (making changes to) your data set. Whenever anyone attempts to use a password-protected data set, the system requests a password unless the data set is protected with the same password that was entered in the logon procedure. The system allows two chances to provide the correct password. If your terminal has the "print-inhibit" feature, the system disengages the printing mechanism at your terminal while you enter the password in response. However, the "print-inhibit" feature is not used if the prompting is for a new password you are adding to the data set.

The PROTECT command also specifies what the person who knows the password can do to the data set; that is, whether he is allowed to read it, or write in it, or both. You can require a password for both reading and writing; or just for reading and not writing. You can also assign one password for reading and a different one for writing. The operands that control the type of operations are:

PWREAD -- you must specify a password before you can read from the data set.

PWWRITE -- you must specify a password before you can write in the data set.

NOPWREAD -- you can read from the data set without specifying a password.

NOWRITE -- you cannot write into the data set (with this password).

There are three valid combinations of operands:

PWREAD PWWRITE -- the password is required for either reading or writing your data set.

PWREAD NOWRITE -- the password is required for reading. Writing is not allowed with this password.

NOPWREAD PWWRITE -- you can read without a password. The password allows you to both read and write the data set.

If you specify only one operand you get two values by default. They are:

Operand	Default Values
PWREAD	PWREAD PWWRITE
NOPWREAD	NOPWREAD PWWRITE
PWWRITE	NOPWREAD PWWRITE
NOWRITE	PWREAD NOWRITE

The type of password operand, the number of times the password is used, and optional security information that you can specify are recorded in the PASSWORD data set of the operating system.

The following example adds the password HUSH for reading and writing the JONES.SECRET.DATA data set:

```
READY
protect secret add(hush) pwread
READY
```

The following example adds another password, WHUSH, to the same data set. This password can be used only for reading the data set:

```
READY
protect secret/hush add(whush) nowrite
READY
```

Note how you must use the password in subsequent commands once you have established it.

You can replace a password. For example, to replace the password SESAME for HUSH in the JONES.SECRET.DATA data set, enter

```
READY
protect secret/hush replace(hush,sesame)
READY
```

Note that when you are replacing a password you do not have to specify the function of the password.

You can also delete a password. For example, if you no longer require the WHUSH password for reading the data set, enter

```
READY
protect secret/sesame delete(whush)
READY
```

You can use the DATA operand to specify optional security information to be recorded in the system. For example, when you establish the password AB#72 for the SMITH.SALES.TEXT data set, you can also specify other information:

```
READY
protect sales add(ab#72) data(password changes on monday)
READY
```

To find out what the optional information is, the type of operation allowed, and the number of times the password has been used, use the LIST operand. For example,

```
protect sales list(ab#72)
```

Note: When a data set is renamed you should update the password data set to reflect the change. This prevents your having insufficient space for future entries.

Allocating a Data Set

This section is intended for those users who are going to compile, link edit, or execute (or load) a program. Knowledge of a programming language (such as System/360 Assembler, COBOL, FORTRAN or PL/I) and of the Job Control Language (JCL) statements required to compile, link edit, and execute the program is useful for understanding this section.

The compiler, linkage editor, loader, and your own program require data sets in order to operate. In an operating system without TSO these data sets are defined with data definition (DD) JCL statements. In TSO, these data sets are defined through the EDIT and ALLOCATE commands. You can use the EDIT command to define and create input data sets. You can use the ALLOCATE command to define output and work data sets and libraries, and to allocate the data sets you created with the EDIT command. This section discusses the ALLOCATE command.

Note: Compilers that have prompters associated with them will allocate data sets for you. Your installation can tell you if these Program Product facilities are available to you. The data sets for the linkage editor and loader are allocated for you by the LINK and LOADGO commands, respectively. You need only allocate them if you invoke the linkage editor or the loader with the CALL command.

The number of data sets you need is determined by the program (compiler, linkage editor, loader, or your own program) you are going to use. (The publications associated with the IBM-supplied programs list the data set requirements.) The number of data sets you can allocate depends on the number of data sets assigned to you in your LOGON procedure. The LOGON procedure defines a series of data sets. Some of these data sets are fully defined and correspond to data sets that you always need in your processing. The remaining data sets are left undefined; they are defined when you define a data set with an ALLOCATE or EDIT command.

When you define a data set with the ALLOCATE command, it remains allocated until you use the FREE command to free the data set definition. (The FREE command is described in "Freeing an Allocated Data Set.")

When you create a data set with the EDIT command, the system uses one of the undefined data sets in the LOGON procedure to define the data set. When you save the data set and end the EDIT command, the system saves the data set, enters its name in the system catalog, and frees the definition in the LOGON procedure for further use. When you again use the EDIT command to make changes to the saved data set, the system finds the data set through the system catalog and uses another of the available definitions to define the data set. When you end the EDIT command, the system frees the data set definition. If you want the data set to remain allocated in your LOGON procedure, you must use the ALLOCATE command.

You can list the data sets allocated to you with the LISTALC command (described in "Listing the Names of Your Data Sets"). The system lets you know, as part of the LISTALC listing, how many DD statements are available for allocation. For example, if there are five available data sets you get the following message:

```
5 DATA SETS CAN BE ALLOCATED DYNAMICALLY
```

You can allocate as many data sets as there are available definitions. If you need more data sets you can free a previously allocated data set with the FREE command (described in "Freeing an Allocated Data Set"). After you free a data set, you can use the

available definition to allocate another data set with the ALLOCATE command.

If you have to allocate the same data sets everytime you logon, you can have your installation allocate them in the form of fully defined data sets in the LOGON procedure. In this way you do not have to allocate the same data sets everytime you logon.

The example in Figure 3 illustrates the use of the ALLOCATE command for allocating the data sets required for an execution of the Assembler F compiler. The assembler requires eight data sets for this compilation. They are:

SYSLIB	The macro library (SYS1.MACLIB).
SYSUT1	Work data set.
SYSUT2	Work data set.
SYSUT3	Work data set.
SYSPRINT	Output listing data set. Your terminal is allocated for this purpose.
SYSPUNCH	Data set for a punched deck of an object module. It is to be produced on the standard message output class. (To change this output class to a punch output class, see "Freeing an Allocated Data Set".)
SYSGO	Data set for the object module.
SYSIN	Input source statements to the Assembler. It is created with the EDIT command and defined to the assembler with the ALLOCATE command.

```
.
.
.
READY
edit input.asm new
INPUT
.
.source statements
.
.
EDIT
save
SAVED
end
READY
allocate dataset('sysl.maclib') file(syslib) shr
READY
allocate file(sysut1) new block(400) space(400,50)
READY
allocate file(sysut2) new block(400) space(400,50)
READY
allocate file(sysut3) new block(400) space(400,50)
READY
allocate dataset(*) file(sysprint)
READY
allocate file(syspunch)sysout
READY
allocate dataset(prog.obj) file(sysgo) new block(80) space(200,50)
READY
allocate dataset(input.asm) file(sysin) old
READY
.
.
.
```

Figure 3. Allocating Data Sets for the Assembler F

Freeing an Allocated Data Set

Use the FREE command to release any data sets allocated to you. You can also use this command to change the output class of a SYSOUT data set.

To free a data set specify its data set name or its file name (ddname). If your terminal has been allocated as a data set, you must free it through its file name. You can use the LISTALC command to obtain the file names and data set names of the data sets allocated to you. (LISTALC is described in the Section, "Listing the Names of Your Data Sets".)

The following examples free the data sets allocated in Figure 3 of the section "Allocating a Data Set". The output class of the SYSPUNCH data set is changed to B.

```
free dataset('sys1.maclib',prog.obj,input.asm) file(sysut1,-
sysut2,sysut3,sysprint,syspunch) sysout(b)
```

Listing the Names of Your Data Sets

Use the LISTALC, LISTCAT, and LISTDS commands to list the names of your data sets and obtain further information about them.

LISTALC lists the data sets defined in your LOGON procedure. Both the fully defined data sets and those available for allocation are listed.

LISTCAT lists the names of all cataloged data sets that have your user identification. Cataloged data sets are those whose names are entered in the system catalog. The system catalog is a list the system keeps of the names and locations of cataloged data sets. Your cataloged data sets may or may not be defined in your LOGON procedure. Data sets that are cataloged but not entered in the LOGON procedure are those that you create and save with the EDIT command but do not allccate with the ALLOCATE command. Other cataloged data sets are those that you have created and saved during previous terminal sessions but never deleted with the DELETE command.

LISTDS gives you information on specific data sets which are currently cataloged or allocated, or both. The information you receive includes:

- The serial number of the volume on which the data set resides.
- The record format, logical record length, and blocksize of the data set.
- The data set organization.
- Directory information for a member of a partitioned data set.

This information is described in detail in the publicatcn IBM System/360 Operating System: Job Control Language User's Guide, GC28-6703.

In addition to the information listed above for the three commands, there are certain operands you can use to obtain additional information on the data sets. The operands and the commands to which they apply are:

Operand	LISTALC	LISTCAT	LISTDS
STATUS	X		X
HISTORY	X	X	X
MEMBERS	X	X	X
SYSNAMES	X		
VOLUMES		X	
LEVEL		X	
LABEL			X

The STATUS operand provides you with:

- The file name(ddname) for the data set.
- The scheduled disposition and conditional disposition of the data set. The scheduled disposition determines whether the system will retain or delete the data set after it is used. The conditional disposition determines whether the system is to retain or delete the data set in case of abnormal termination. The keywords that denote the dispositions are CATLG, KEEP, DELETE and UNCATLG. CATLG means that the data set is retained and its name is kept in the system catalog. KEEP means that the data is retained but not cataloged. DELETE means that all references to the data set are to be removed from the system and that the space it occupies is to be released for use by other data sets. UNCATLG means that a previously cataloged data set is retained, but its name is removed from the catalog.

The HISTORY operand provides you with:

- The creation date of the data set.
- The expiration date of the data set.
- An indication as to whether or not the data set has password protection.
- The data set organization.

The MEMBERS operand provides you with a list of the member names of a partitioned data set including any aliases.

The SYSNAMES operand provides you the names assigned by the system to any allocated data set you did not name.

The VOLUMES operand provides you with the serial numbers of the volumes on which your cataloged data sets reside.

The LEVEL operand lets you request a listing of only part of your cataloged data sets, or a listing of some other user's cataloged data sets.

The LABEL operand provides you with the information in the Data Set Control Block (DSCB) of a specific data set.

Programming at the Terminal

You can use the TSO facilities to compile, link edit, and execute (or compile and load) your source program at the terminal. TSO also allows you to use any other program, such as utilities, at the terminal. That is, instead of taking your job to the computer room to run it directly under the operating system, you can use the TSO facilities to enter it through your terminal. These facilities reduce your job turnaround time because you get immediate results at the terminal.

You can also use the terminal to submit your job for processing at the computer in the conventional manner. That is, you submit your job through the terminal but do not want to get immediate results at the terminal. The results are sent to you from the computer room after your job is executed or you may obtain them at the terminal at a later time. Jobs submitted in this manner are called background jobs.

Most compilers or assemblers that can be used under the operating system can be used from your TSO terminal. They can be used to obtain results at the terminal, or for background jobs. In addition to these programs, your installation may have one or more of the special TSO Program Product compilers and other TSO programs for your use at the terminal. They are:

- Interactive Terminal Facility (ITF): PL/I -- A problem-solving language processor.
- Interactive Terminal Facility (ITF): BASIC -- A problem-solving language processor.
- Code and Go FORTRAN -- A FORTRAN compiler designed for a very fast compile-execute sequence at the terminal.
- FORTRAN IV (G1) -- A version of the FORTRAN IV (G) compile modified for the terminal environment.
- TSO FORTRAN Prompter -- An initialization routine to prompt you for options and invoke the FORTRAN IV (G1) Processor.
- FORTRAN IV Library (Mod I) -- Execution-time routines for use with either Code-and-Go FORTRAN or FORTRAN IV (G1).
- Full American National Standard COBOL Version 3 -- A version of the American National Standard COBOL modified for the terminal environment.
- TSO COBOL Prompter -- An initialization routine to prompt you for options and invoke the full American National Standard COBOL Version 3 Processor.
- TSO Assembler Prompter -- An initialization routine to prompt you for options and invoke the Assembler (F).

If your installation has one or more of the TSO Program Products, it will provide you with documentation that explains how to use them. This section explains how to use the programs normally available under the operating system. The following paragraphs describe how you can:

Create a program
Compile your program
Link edit a compiled program

Execute a program
Load a program
Process background jobs

It is assumed that you are familiar with a programming language and with the information in the Guide to Writing a Terminal Monitor Program or a Command Processor or Terminal User's Guide that corresponds to that language. The options and data set requirements of the compilers, linkage editor, and loader are summarized in the publication, IBM System/360 Operating System: Job Control Language User's Guide, GC28-6703.

Creating a Program

Before your source program is compiled you must introduce it to the system. You do so with the EDIT command, as described in the section, "Entering and Manipulating Data".

When you enter the EDIT command you must specify the type operand or give a descriptive qualifier to the data set name. The type (or descriptive qualifier) tells the system which programming language you are using. If you are writing a program and JCL statements to be submitted as a background job, use CNTL as the type or descriptive qualifier.

The EDIT command allows you to specify certain options for your source program. You can use the SCAN operand to request syntax checking when the data set type is GOFORT, FORT, BASIC, PLIF, PLI, or IPLI. You can use the LINE operand specify the length of the input line for PL/I source programs. The length of the input line for the Assembler, FORTRAN, and COBOL is 80 characters.

After you create your source program you must use the SAVE subcommand to save the data set before you end the EDIT command. Your source program is now ready for compilation.

The example in Figure 4 shows the creation of an assembler source program.

```
READY
edit   prog1   new   asm
INPUT
  .
  .
  .    source program
  .
  .
EDIT
save
SAVED
end
READY
```

Figure 4. Creating an assembler source program

Compiling a Program

If you are using a TSO Program Product compiler and prompter, you can ignore this section. The prompter allocates data sets and calls the compiler for you.

You can use the CALL command to invoke the compiler that will compile your source program. Before you use the CALL command to invcke the compiler you must use ALLOCATE commands to allocate all the data sets required for compilation. Data set allocation is discussed in "Allocating a Data Set" in the section "Entering and Manipulating Data". The data sets required by your compiler are described in the Guide to Writing a Terminal Monitor Program or a Command Processor or Terminal User's Guide associated with your compiler.

You must give the data set name of your compiler in the CALL command. The data set names are shown in Table 5.

Table 5. Data Set Names of the Compilers

Compiler	Data Set Name
Assembler F	'SYS1.LINKLIB(IEUASM)'
COBOL E	'SYS1.LINKLIB(IEPCBL00)'
COBOL F	'SYS1.LINKLIB(IEQCBL00)'
American National Standard COBOL	'SYS1.LINKLIB(IFFCBL00)'
FORTRAN E	'SYS1.LINKLIB(IEJFAAA0)'
FORTRAN G	'SYS1.LINKLIB(IEYFORT)'
FORTRAN H	'SYS1.LINKLIB(IEKAA00)'
PL/I F	'SYS1.LINKLIB(IEMAA)'

Note that the data set name is a fully qualified name and must be enclosed in apostrophes. For example, if you want to use the FORTRAN H compiler, enter:

```
READY
call 'sys1.linklib(iekaa00)'
```

In addition to the compiler's data set name, you can enter the compiler options you desire in the CALL command. These opticns are those specified with the PARM parameter of the EXEC statement when you are running your program directly under the operating system rather than through TSO. For example, if you want to use the MAP, NOID, and OPT=2 options of the FORTRAN H compiler, enter:

```
READY
call 'sys1.linklib(iekaa00)' 'map noid opt=2'
```

Any messages and other output produced by the compiler will appear in your listing after the CALL command. Once the compiler completes its processing you receive the READY message. You can then free any allocated data sets you no longer need.

Figure 5 shows the commands required to create a COBOL source program, allocate the eight data sets required for compilatien, call the COBOL F compiler, and free all allocated data sets except the one that contains the compiled program (object module). It is assumed you are using your user identification as part of all data set names except SYS1.COBLIB.

```
READY
edit   prog2   new   cobol
INPUT
  .
  .
  .    source program
  .
  .
EDIT
save
SAVED
end
READY
allocate dataset('sys1.coblib') file(syslib) shr
READY
allocate file(sysut1) new block(460) space(700,100)
READY
allocate file(sysut2) new block(460) space(700,100)
READY
allocate file(sysut3) new block(460) space(700,100)
READY
allocate file(sysut4) new block(460) space(700,100)
READY
allocate dataset(*) file(sysprint)
READY
allocate dataset(prog2.obj) file(syslin) new block(80) space(500,100)
READY
allocate data set(prog2.cobol) file(sysin) old
READY
call 'sys1.linklib(ieqcb100)' 'map load nodeck flagw'
  .
  .
  .
  .
  .    COBOL listings and messages
  .
  .
READY
free file(syslib,sysut1,sysut2,sysut3,sysut4,sysprint,sysin)
READY
```

Figure 5. COBOL Compilation

Link Editing a Compiled Program

The LINK command makes available to you the services of the linkage editor. The linkage editor processes the compiled program (object module) and readies it for execution. The processed object module becomes a load module. Optionally, the linkage editor can process more than one object module and/or load module and transform them into a single load module. For complete information on the linkage editor, refer to your Guide to Writing a Terminal Monitor Program or a Command Processor and to the publication, IBM System/360 Operating System: linkage Editor and Loader, GC28-6538.

In your LINK command you must first list the name or names of the object modules you want to link edit. (If you omit the descriptive qualifier the system assumes OBJ.) After the names of the object modules you should use the LOAD operand to indicate the name of a member of a partitioned data set where you want the load module placed. (If you omit the user-supplied name of the load module data set the system assumes it has the same user-supplied name as the object module. If you

omit the descriptive qualifier the system assumes LOAD. If you omit the member name the system assumes TEMPNAME.) For example, if you want to link edit the load module in the JONES.PROG2.OBJ data set and place the resultant load module in member TEMPNAME of the JONES.PROG2.LOAD data set, enter:

```
READY
link prog2
```

If you want to link edit the load module in the JONES.PROG2.OBJ data set and place the resultant load module in member ONE of the JONES.MODS.LOAD data set, enter:

```
READY
link prog2 load (mods(one))
```

The following example shows how to link edit the two object modules in the SMITH.PGM1.OBJ and SMITH.PGM2.OBJ data sets. The resultant load module is placed in member TEMPNAME of the SMITH.LM.LOAD data set.

```
READY
link pgm1,pgm2 load(lm)
```

You can control the link editing process with linkage editor control statements. These control statements can be in a previously created data set, or can be introduced through the terminal. You must give the name of the data set, or an asterisk (indicating that you will introduce the control statements through the terminal) in the list of input data sets. The following example shows how to link edit the object module in the CARTER.TRAJ.OBJ data set. There are control statements in the CARTER.CNTL.DATA data set. The load module is placed in member TEMPNAME of CARTER.TRAJ.LOAD.

```
READY
link(traj,cntl.data)
```

Using the same example, if you want to introduce the control statements through your terminal, enter:

```
READY
link(traj,*)
```

The system will prompt you for the control statements at the appropriate time. You must follow your last control statement with a null line.

You can also have the linkage editor search a subroutine library to resolve external references. (External references are references to other modules.) The subroutine library is usually one of the language libraries and it is specified with one of the following operands:

Operand	Subroutine Library
COBLIB	SYS1.COBLIB
FORTLIB	SYS1.FORTLIB
PLILIB	SYS1.PL1LIB

In addition to, or instead of a language library, you can use the LIB operand to specify the name of one or more user libraries. The libraries are searched in the order you specify.

The following example shows how to link edit the object module in JAMES.PRG.OBJ. The load module is placed in JAMES.PRG.LOAD(TEMPNAME). The libraries SYS1.PL1LIB, and DEPT39.LIB.SUBRT2 are to be searched to resolve external references.

```
READY
link prg plilib lib('dept39.lib.subrt2')
```

The LINK command also lets you specify the linkage editor options. These options are those specified with the PARM parameter of the EXEC statement when you are running the linkage editor directly under the operating system rather than through TSO. For example, if you want to use the LET and XCAL options when the object module in AGNES.RET.OBJ is link edited and placed in AGNES.TBD.LOAD(MOD), enter:

```
READY
link ret load(tbd(mod)) let xcal
```

Linkage editor listings (specified with the MAP, XREF, and LIST options) are directed to a data set or to your terminal. You indicate your choice with the PRINT operand. The following example shows that the object module in BILL.PRGM.OBJ is to be link edited and placed in BILL.PRGM.LOAD(TEMPNAME). The listing produced by the MAP option is to placed in the BILL.LIST.LINKLIST data set.

```
READY
link prgm map print(list)
```

Note that if you omit the descriptive qualifier from the print data set name, the system assumes LINKLIST. If you omit the user-supplied name, the system assumes it has the same user-supplied name as the object module. For example if the listing is to be placed in the BILL.PRGM.LINKLIST data set, enter:

```
READY
link prgm map print
```

Using the same example, if you want the listing to appear on your terminal, enter an asterisk in the PRINT operand.

```
READY
link prgm map print(*)
```

Error messages are listed at the terminal as well as on the print data set when you specify a data set name instead of an asterisk. If you want the error messages to appear only on the print data set, enter the NOTERM operand. For example,

```
READY
link prgm map print noterm
```

Executing a Program

You can use the CALL command to execute your program after it has been link edited. You can also use CALL to execute any other program in load module form, such as utilities and compilers.

Before you use CALL to execute your program you can use the EDIT and ALLOCATE commands to define your data sets. Use EDIT to create your input data sets, and ALLOCATE to allocate your input, work, and output data sets.

You must specify the data set name and member name of the member that contains your program in load module form. For example, if you want to execute a program that resides in DEPTB.PROGS.DAILY(NUM3), enter:

```
READY
call 'deptb.progs.daily(num3)'
```

If you omit the descriptive qualifier and member name, the system assumes LOAD and TEMPNAME, respectively. For example, if your program resides in JONES.LIB.LOAD(MEM2), enter:

```
READY
call lib(mem2)
```

If your program resides in JONES.LIB.LOAD(TEMPNAME), enter:

```
READY
call lib
```

You can pass parameters to your program if you wrote it in assembler or PL/IF. These are the parameters you would specify with the PARM parameter of the EXEC statement if you were running your program directly under the operating system. For example, if you want to pass the parameters OPTION1 and OPTION5 to a program that resides in SMITH.ASMPG.LOAD(MEM3), enter:

```
READY
call asmpg(mem3) 'option1 option5'
```

Figure 6 shows how the COBOL program created and compiled in Figure 5 can be link edited and executed. In Figure 5, the compiled program (object module) was placed in PROG2.OBJ. After link editing, the load module is placed in PROG2.LOAD(TEMPNAME). Your program requires three data sets for execution. The input data set, INPUT.DATA, is created with the EDIT command. ALLOCATE commands are used to allocate the input data set, a work data set, and an output data set. CALL is used to execute your program. The PROG2.COBOL, PROG2.OBJ, PROG2.LOAD, and INPUT.DATA data set are deleted. (The other data sets are automatically deleted because they were not given a data set name when allocated.) It is assumed you are using your user identification as part of the data set names.

```
READY
link prog2  print(*) map
   .
   .
   .
   .
   .    linkage editor messages and listings
   .
   .
   .
READY
edit input.data new
INPUT
   .
   .
   .
   .
   .    input data
   .
   .
   .
   .
   .
EDIT
save
SAVED
end
READY
allocate dataset(input.data) file(input) old
READY
allocate file(work) new block(100) space(300,10)
READY
allocate dataset(*) file(print)
READY
call prog2
   .
   .
   .
   .
   .    output from your program
   .
   .
   .
READY
delete (prog2.* input.data)
READY
```

Figure 6. Link editing and executing a program

Loading a Program

The LOADGO command makes available to you the services of the loader. The loader combines the basic functions of the linkage editor and program fetch. That is the loader link edits and executes your program. Therefore, the LOADGO command combines the basic functions of the LINK and CALL commands. For complete information on the loader, refer to the Guide to Writing a Terminal Monitor Program or a Command Processor and to the publication, IBM System/360 Operating System: Linkage Editor and Loader, GC28-6538.

The loader can process and execute a compiled program (object module) or a link edited program (load module). Optionally, it can combine object modules and/or load modules and execute them. (If you want to load and execute a single load module, the CALL command is more efficient.)

Before you use the LOADGO command you can use the EDIT and ALLOCATE commands to create and allocate any data sets required to execute your program.

In your LOADGO command you must list the name or names of the object and load modules you want to load. For example, if you want to load the object module in JONES.PROG3.OBJ, enter:

```
READY
loadgo prog3
```

If you want to load the object modules in JONES.PROG3.OBJ, JONES.COB.OBJ, and the load module in JONES.COB.LOAD(TWO), enter:

```
READY
loadgo (prog3 cob.obj cob.load(two))
```

You can also pass parameters to your program if you wrote it in assembler or PL/IF. These are the parameters you would specify with the PARM parameter of the EXEC statement if you were running your program directly under the operating system. For example, if you want to pass the parameters OPTION1 and OPTION5 to a compiled program that resides in SMITH.ASMPG.OBJ, enter:

```
READY
loadgo asmpg 'option1 option5'
```

You can have the loader search a subroutine library to resolve external references. The subroutine library is usually one of the language libraries and it is specified with one of the following operands:

Operand	Subroutine Library
COBLIB	SYS1.COBLIB
FORTLIB	SYS1.FORTLIB
PLILIB	SYS1.PL1LIB

In addition to, or instead of a language library you can use the LIB operand to specify the name of one or more user libraries. The libraries are searched in the order you specify.

The following example shows how to load the object module in JAMES.PRG.OBJ. The libraries SYS.PL1LIB, and DEPT39.LIB.SUBRT2 are to be searched to resolve external references.

```
READY
loadgo prg plilib lib('dept39.lib.subrt2')
```

The LOADGO command also lets you specify the loader options. These options are those specified with the PARM parameter of the EXEC statement when you are running the loader directly under the operating system. For example, if you want to use the LET and EP(MAIN) options when the object module in BROWN.CIR.OBJ is loaded, enter:

```
READY
loadgo cir let ep(main)
```

Loader listings (specified with the MAP option) are directed to a data set or to your terminal. You indicate your choice with the PRINT operand. The following example shows that the object module in BILL.PRGM.OBJ is to be loaded. The listing produced by the MAP option is to be placed in the BILL.LIST.LOADLIST data set.

```
READY
loadgo prgm map print(list)
```

Note that if you omit the descriptive qualifier from the print data set name, the system assumes LOADLIST. If you omit the user-supplied name, the system assumes it has the same user-supplied name as the object module. For example, if the listing is to be placed in the BILL.PRGM.LOADLIST data set, enter:

```
READY
loadgo prgm map print
```

Using the same example, if you want the listing to appear on your terminal, enter an asterisk in the PRINT operand.

```
READY
loadgo prgm map print(*)
```

Error messages are listed on the terminal as well as on the print data set when you specify a data set name instead of an asterisk. If you want the error messages to appear only on the print data set, enter the NOTERM operand. For example,

```
READY
loadgo prgm map print noterm
```

Figure 7 shows how the COBOL program created and compiled in Figure 5 can be loaded. The loading operation shown in Figure 7 is the equivalent of the link editing and execution shown in Figure 6. The same data sets required for execution of your program in Figure 6 are required in this example. The object module resides in PROG2.OBJ. A load module is not produced by the loader, therefore, only PROG2.COBOL, PROG2.OBJ, and INPUT.DATA are deleted at the end. It is assumed you are using your user identification as part of the data set names.

```
READY
edit input.data new
INPUT
  .
  .
  .
  .
  .
  .    input data
  .
  .
  .
EDIT
save
SAVED
end
READY
allocate dataset (input.data) file(input) old
READY
allocate file(work) new block(100) space(300,10)
READY
allocate dataset(*) file(print)
READY
loadgo prog2 map print(*)
  .
  .
  .
  .
  .    loader listings and output from your program
  .
  .
  .
READY
delete(prog2.* input.data)
READY
```

Figure 7. Loading a Program

Processing Background Jobs

You can submit background jobs for processing if your installation authorizes you to do so. This authorization is recorded in the system with your user attributes. If you have this authorizatiocn, the system lets you use the four commands (SUBMIT, STATUS, CANCEL and OUTPUT) that control the processing of background jobs. You can use those commands to submit a background job, to display the status of a background job, to cancel execution of a background job, and to control the output of a background job.

SUBMITTING BACKGROUND JOBS

Before you submit a background job with the SUBMIT command ycu can use the EDIT command to create a data set (or a member of a partitioned data set) that contains the job or jobs you want to submit. Each job consists of Job Control Language (JCL) statements and of program instructions and/or data.

The JCL Statements required for a job must conform to System/360 Operating System (MVT) standards. They are described in the publications, IBM System/360 Operating System: Job Control Language User's Guide, GC28-6703, and IBM System/360 Operating System: Job Control Language Reference, GC28-6704.

The first JCL statement in the data set is usually a JOB statement. The jobname in the JOB statement can be up to eight characters in length and consists of your user identification followed by one or more letters or numbers. For example SMITH23 or JONESXYZ.

If the jobname consists of only your user identification, the system will prompt you for a single character to complete the jobname. When you submit the job with the SUBMIT command this allows you to change jobnames without re-editing the data. For example, you may submit the same job several times, and supply a different character for the job name each time you are prompted.

If the jobname does not begin with your user identification, you can submit it with the SUBMIT command and request its status with the STATUS command, but you cannot refer to it with the CANCEL or OUTPUT command.

If the first statement of your data set is not a JOB statement, the system generates the following JOB statement when you submit it with the SUBMIT command.

```
//userid JOB     ,GENERATED JOB STATEMENT
//              userid,
//              MSGLEVEL=(1,1),
//              NOTIFY=userid
```

You will be prompted for a character to complete the jobname. The account number used is the same you used in the LOGON command.

When you enter the SUBMIT command you must give the name of the data set (or data sets) that contains the background jobs. You can also specify the NONOTIFY operand to specify that you do not want to be notified when a background job with a generated JOB statement terminates.

Figure 8 shows how to create and submit a background job. Note that the data set type in the EDIT command must be CNTL.

You may include more than one job in one data set. You can omit the JOB statement for the first job, but all jobs after the first must have their own JOB statement. Although you submit all jobs in the data set with one SUBMIT command, you can subsequently refer to each job with separate STATUS, CANCEL, and OUTPUT commands.

If an error occurs while the jobs are being processed by TSO before actually being submitted, further processing will be terminated. No other input specified by the SUBMIT command will be processed. When you submit more than one job with a single command, and TSO finds an error while processing the first job, the second job is not processed. An error that occurs in the second job does not affect the first. Any jobs processed prior to the error are submitted for execution; jobs that were not processed because of the error cannot be submitted.

```
READY
edit  backpgm  new  cntl  nonum
INPUT
//smith3     job    7924,smith,msglevel=(1,1)
//step1      exec   pgm=iepck100,parm=(deck,maps,list)
//syslib     dd     dsname=sys1.coblib,disp=shr
//sysut1     dd     unit=2311,space=(trk,(50,10))
//sysut2     dd     unit=2400
//sysut3     dd     unit=2400
//sysprint   dd     sysout=a
//syspunch   dd     dsname=comp.cobol,disp=(,catlg),unit=2400
//sysin      dd     *
  .   source statements
  .
  .
  .
  .
/*
//step2      exec   pgm=loader,parm=(map,let,call)
//syslib     dd     dsname=sys1.coblib,disp=shr
//syslout    dd     sysout=a
//syslin     dd     dsname=*.step1.syspunch
//master     dd     dsname=order,disp=old
//print      dd     sysout=a
//input      dd     *
  .
  .
  .   input data
  .
  .
/*
//
(null line)
EDIT
save
SAVED
end
READY
submit  backpgm  nonotify
READY
```

Figure 8. Submitting a Program as a Background Job

DISPLAYING THE STATUS OF BACKGROUND JOBS

Any time after you submit a background job you can use the STATUS command to have its status displayed. The display will tell you whether the job is awaiting execution, is currently executing, or has executed. For example, if you want to display the status of SMITH23, enter:

```
READY
status smith23
```

If you want to know the status of all the jobs that begin with your user identification, enter the STATUS command without operands:

```
READY
status
```

CANCELLING BACKGROUND JOBS

You can use the CANCEL command to cancel execution of a background job. If the job has already been executed, the CANCEL command has no effect.

For example, if you want to cancel job JONESAB, enter:

```
READY
cancel jonesab
```

After you enter the CANCEL command, the system will send you a message telling you that the jobs specified have been cancelled.

CONTROLLING THE OUTPUT OF A BACKGROUND JOB

You can use the OUTPUT command to:

- Direct the JCL statements and system messages (MSGCLASS) and system output data sets (SYSOUT) produced by a background job to your terminal.
- Direct the MSGCLASS and SYSOUT output from a background job to a specific data set.
- Change an output class used in a background job.
- Delete the output data sets (SYSOUT) or the system messages (MSGCLASS) for background jobs.

Unless you use the NONOTIFY operand of the SUBMIT command, a message is placed in the broadcast data set when the background job terminates. You can then use the OUTPUT command to control the output produced by the job on the MSGCLASS and SYSOUT classes before the system processes them.

For example, assume that job GREEN 67 produces output on classes A, B, D, G, and M. If you want the output on classes G and M listed at the terminal, enter:

```
READY
output  green67  class(g m)  print(*)
```

If you want the output of class B to be listed in the GREEN.KEEP.OUTLIST data set, enter:

```
READY
output  green67  class(b)  print(keep)
```

If you want to change the output in class A to class C, enter:

```
READY
output  green67  class(a)  noprint(c)
```

If you want to delete the output from class D, enter:

```
READY
output  green67  class(d)  noprint
```

If you wish, you can enter the PAUSE operand in the OUTPUT command. PAUSE will make the system stop after each data set is listed on your terminal or on the data set you indicate with the PRINT operand. When the system pauses it sends you the message OUTPUT. You then have the option of pressing the RETURN key to continue processing or entering the CONTINUE or SAVE subcommand.

The CONTINUE subcommand allows you to continue processing your output after an interruption occurs. An interruption occurs when:

- An output operation completes and you used the PAUSE operand in the OUTPUT command.
- An output operation terminates because of an error condition.
- You press the attention key.

When you enter the CONTINUE subcommand, the system will resume printing with the next data set being processed or with the next message if a block of messages is being processed. In the following example you request that the data sets in output classes B and C be listed at your terminal. The system pauses after printing the data set in B. You enter the CONTINUE subcommand to resume processing with the data set in C.

```
READY
output  jones2  class(b c)  print(*)  pause
.
.
.
.
.   output class B
.
.
.
OUTPUT
continue
.
.
.
.   output class C
.
.
.
```

If the interruption was not caused by a pause, you may prefer to resume printing at the beginning of the data set being processed or approximately ten lines before the interruption. If you want to resume printing at the beginning, enter:

```
OUTPUT
continue  begin
```

If you prefer to resume printing a few lines before the interruption occurred, enter:

```
OUTPUT
continue  here
```

The CONTINUE subcommand also lets you respecify the PAUSE operand of the OUTPUT command. If you entered PAUSE in the OUTPUT command, you can enter NOPAUSE in the CONTINUE subcommand, for example,

```
READY
output  smithc  class(d)  print(data)  pause
.
.
.
.
.
OUTPUT
continue  begin  nopause
```

If you did not specify PAUSE in the OUTPUT command, you can do so in the CONTINUE subcommand. This causes the system to pause at the end of each data set processed subsequently.

The SAVE subcommand allows you to place the data set listed before the pause into another data set. This allows you to retrieve the data set at a later time. In the following example you request that data sets in output classes E and F be listed at your terminal. After listing the data set in E you request that it be saved in the BROWN.OUTDATA.OUTLIST data set. You continue processing the next data set after saving the data set in class E.

```
READY
output  browne  class(e f)  print(*)  pause
.
.
.
.
.
OUTPUT
save  outdata
OUTPUT
continue
.
.
.
.
```

The END subcommand is used to terminate the OUTPUT command. For example,

```
READY
output  dept30a  class(a)  print(*)  pause
.
.
.
.
OUTPUT
end
READY
```

Testing a Program

The operating system provides you with facilities to test your program from the terminal. They are the test facilities, if any, provided by your compiler, and the TSO TEST command. The compiler test facilities are described in the publications associated with the compiler. A brief description of the TEST command follows.

The TEST command allows you to "debug" your program. That is, it helps you to test a program for proper execution and to find programming errors. To use TEST effectively, you should be familiar with the assembler language. If you are using another language, for example COBOL, you can still use the TEST command to obtain listings and other information to give to your installation's system programmer who can help you debug your program. (You can use the full facilities of the TEST command to debug your program if you can correlate the statements in your source program listing to the resultant assembler language statements in the object listing.)

If you are an assembler programmer, refer to the publications IBM System/360 Operating System: Time Sharing Option, TSO Guide to Writing a Terminal Monitor Program or a Command Processor and IBM System/360 Operating System: Time Sharing Option, Command Language for a complete description of the facilities of the TEST command.

If you are not an assembler programmer, your system programmer will probably provide you with a test procedure. The most common situation he may provide for occurs when your program is executing and you receive a message that the program has begun to abnormally terminate. He may tell you to enter the TEST command and then the LOAD subcommand with the name of a program that will test your program. For example, if the name of the program that will test yours is DPTEST, use the following sequence.

```
test
TEST
load (dptest)
```

If the system programmer does not give you the name of a testing program, he may instruct you to use the TEST command and a set of its subcommands that produce listings of your program and other pertinent information. For example, he could ask you to perform procedures similar to the following.

Example 1:

```
test
TEST
listpsw
FFE5006 40088540
where 88540.
88540.  LOCATED AT +3C IN (load-module-name.csectname) UNDER TCB
 LOCATED AT 86B68.
list 88540.  -32n length (32)
```

First, you begin testing by entering the TEST command. You can now use the subcommands of TEST to "debug" your program.

Enter the LISTPSW subcommand to determine the address of the instruction that failed in your program. The last five characters of the PSW that is listed can then be entered after the WHERE subcommand and the system

will then provide the location and the program name in which the ABEND occurred. When LIST is entered in the preceeding manner, the thirty-two bytes of instructions prior to the ABEND will be displaed.

At this time all the registers may be listed in the following manner to aid you in solving the problem:

```
list OR:15R
```

If you wish to trace the execution of your program you may enter the following:

Example 2:

```
at +0:  9000 (go)
at +32
at +8c
at +10a
go
```

In this case breakpoints will be set starting at +0 and ending when an invalid instruction is encountered. Specifying the GO subcommand will allow continued execution until one of the breakpoints set in subsequent AT subcommands is encountered. The setting of these additional breakpoints allows you to interrupt execution so that you can examine registers or storage at predetermined intervals.

Example 3:

To supply new values for a range of registers, you can enter:

```
0r=(x'0',x'0',x'0')
```

The values specified would be assigned starting with register 0, register 1, etc. until all values in the list have been assigned.

Example 4:

If you want to display storage at a known relative address you may enter:

```
list +34
+34 47F0C220
```

If you want not only to display storage, but also to find out the absolute address associated with the relative address, you can enter:

```
list +34+0
A0680.  47F0C220
```

If you prefer, you can elect not to test your program. To do so, press the RETURN key after you receive the message informing you that your program is abnormally terminating. You will then receive a READY message, which allows you to enter any command you wish.

Using and Writing Command Procedures

In many cases a given function is performed by a sequence of commands. For example, several commands are needed to allocate data sets for a compilation. Every time you want to accomplish that function you must enter the same sequence of commands, or else, you can simplify your work by using a command procedure. A command procedure is a set of TSO commands, and, optionally, subcommands and data that have been placed in a data set. Whenever you want to accomplish the functions performed by the command procedure you can use the EXEC command to call the procedure. The command procedure you call may contain symbolic values. A symbolic value stands as a symbol for an operand or the value of an operand. Symbolic values are used so that the command procedure can be easily modified when it is called by the EXEC command.

This section consists of two parts. The first part, "Using Command Procedures", describes how to call a command procedure and how to assign actual values to symbolic values. The second part, "Writing Command Procedures" describes how to write a command procedure and place it in a data set.

Using Command Procedures

Use the EXEC command to call a command procedure and to assign values to any symbolic values it may contain. You will not get any prompting messages once execution of the command procedure has begun.

CALLING A COMMAND PROCEDURE

To call a command procedure, enter an EXEC command. In the EXEC command you identify the command procedure in one of two ways:

1. If the command procedure is in a data set, enter EXEC followed by the name of the data set. The following example calls the command procedure that resides in the JP.COMPROC.CLIST data set:

   ```
   READY
   exec  comproc
   ```

   Note that if you omit the descriptive qualifier the system assumes CLIST. If the descriptive qualifier is not CLIST you must enter the fully qualified name enclosed in apostrophes. For example, if the command procedure resides in the data set JP.COMPROC.CP, you must enter:

   ```
   READY
   exec 'jp.comproc.cp'
   ```

2. If the command procedure resides in a member of a partitioned data set (called a command procedure library) enter only the member name. (The command procedure library must have been defined by your installation.) The following example shows how to call the command procedure in member PROC3 of your command procedure library:

   ```
   READY
   proc3
   ```

ASSIGNING VALUES TO SYMBOLIC VALUES

If the command procedure contains symbolic values, the installation should provide you with a list of the symbolic values used, what meaning is associated with each symbolic value, whether you must supply an actual value for each symbolic value, and whether a symbclic value will assume a default value if you fail to provide one. Figure 9 shows a sample sheet for a command procedure such as your installatiсn may provide you

```
Command Procedure: LISTUPDT (member name)
Purpose: Update inventory list
Symbolic values:
   WEEKIN  WEEKOUT  NEW  OUTPUT(*)

      WEEKIN:  Required.  Replace with name of input data set.
     WEEKOUT:  Required.  Replace with name of output data set.
         NEW:  Optional.  Code NEW if output data set does not exist.
               Omit if output data set already exists.
   OUTPUT(*):  Optional.  Directs reports prepared by procedure to
               your terminal.  If you want to direct reports to a
               data set, replace the * with the data set name.
```

Figure 9. Symbolic Values for a Command Procedure

After you decide which values you are going to replace for the required symbolic values, and which optional symbolic values you are going to use, enter the values in the EXEC command used to call the procedure. The values must follow the name of the data set or member that contains the procedure. If the procedure resides in a data set, enclose the values in apostrophes. The required values must be entered in the order given to you. Optional values can be entered in any order after you enter the required values. The following example calls the procedure shown in Figure 9. The name of the input data set is JONES.W26IN.DATA. The name of the output data set is JONES.W26OUT.DATA. The output data set does not yet exist. The reports produced by the command procedure are directed to the JONES.W26REP.DATA data set.

```
READY
listupdt  w26in  w26out  output(w26rep)  new
```

Writing Command Procedures

Functions that are performed on a regular basis, such as calling a compiler, can be simplified when the commands that perform the functions are kept as command procedures. Once the commands are placed in a partitioned or sequential data set or in a command procedure library (a partitioned data set), any terminal user who wants to perform those functions need only enter an EXEC command.

Command procedures contain commands and, optionally, subcommands, data and line numbers. A command procedure may also contain command procedure statements (PROC, WHEN, and END) that control execution of the procedure. The PROC statement defines symbolic values in the procedure. The WHEN statement initiates or terminates a procedure according to certain conditions. The END statement marks the end of the procedure.

The command procedure is entered in the data set or into a member of a command procedure library with the EDIT command. The descriptive qualifier normally used is CLIST. You must also use the SAVE subcommand to save the command procedure.

ASSIGNING SYMBOLIC VALUES

When you enter the commands and subcommands in the procedure, you can include symbolic values for any operand or value of an operand. A symbolic value is characterized by a name preceded by an ampersand (&). The name consists of letters and numbers, but it must begin with a letter. For example, if you want to substitute the symbolic value &DSNAME for the 'data set name' operand in the following statement:

```
EDIT data set name NEW DATA
```

enter:

```
edit &dsname new data
```

If the symbolic value must be immediately followed by a character (such as a right parenthesis or an apostrophe), the symbolic value must end with a period. For example, if you want to substitute the symbolic value &DSNAME for the "data set name" operand in the following expression:

```
DATASET(data set name)
```

enter:

```
dataset(&dsname.)
```

A command procedure that contains symbolic values must begin with a PROC statement. The symbolic values that are identified by ampersands are defined by the operands of the PROC statement. There are two types of symbolic values:

- Required -- a positional operand that must be replaced by the user in the EXEC command. It can contain up to 252 characters.
- Optional -- a keyword operand that can be replaced by the user if desired. It can contain up to 31 characters.

The PROC statement must indicate the number of required symbolic values to be supplied by the user. (If none of the symbolic values are required, enter zero.) After the number, list the required symbolic values omitting their ampersands. After the required symbolic values, list the optional symbolic values omitting their ampersands. For example, assume you have the following command procedure named PR39:

```
PROC 3 INPUT OUTPUT LIST LINES()
ALLOCATE DATASET(&INPUT.)  FILE(INDATA) OLD
ALLOCATE DATASET(&OUTPUT.)  BLOCK(100) SPACE(300,10)
ALLOCATE DATASET(&LIST.)  FILE(PRINT)
CALL PROG2 '&LINES.'
END
```

The PROC statement indicates that the three symbolic values &INPUT, &OUTPUT, and &LIST are required, and that the symbolic value &LINES is optional. When the user substitutes values for the required symbolic values in the EXEC command he must provide the required values in the same order in which they appear in the PROC statement. The optional values can follow the required values in any order. For example, if the user wants to replace ALPHA for INPUT, BETA for OUTPUT, COMMENT for LIST, and 20 for LINES, he would enter:

```
READY
pr39 alpha beta comment lines(20)
```

In this case, the following substitutions will be made in the command procedure:

```
ALLOCATE DATASET(ALPHA) FILE(INDATA) OLD
ALLOCATE DATASET(BETA) BLOCK(100) SPACE(300,10)
ALLOCATE DATASET(COMMENT) FILE(PRINT)
CALL PROG2 '20'
END
```

You can also use the PROC statement to assign default values to optional symbolic values. That is, if the user fails to provide an actual value for the symbolic value, the system will use the default value to replace the symbolic value. You assign a default value by enclosing it in parentheses after the symbolic value in the PROC statement. For example, in the command procedure illustrated above, you may want to assign 35 as a default value for &LINES. To do this, enter LINES(35) in the PROC statement. That is, the PROC statement would be as follows:

```
PROC 3 INPUT OUTPUT LIST LINES(35)
```

If the user enters the following EXEC command:

```
READY
pr39 alpha beta comment
```

the following substitutions will be made in the command procedure:

```
ALLOCATE DATASET(ALPHA) FILE(INDATA) OLD
ALLOCATE DATASET(BETA) BLOCK(100) SPACE(300,10)
ALLOCATE DATASET(COMMENT) FILE(PRINT)
CALL PROG2 '35'
END
```

TESTING CONDITIONS FOR TERMINATION

The programs invoked with a CALL or LOADGO command can issue a return code (a number) to indicate its relative "success". The return codes of IBM-supplied programs are listed in the publications associated with the program. Only those user programs written in the assembler language or PL/I can issue return codes. User return codes are usually standardized in each installation.

You can insert a WHEN statement after any CALL or LOADGO command or a processor (such as a compiler or link editor) in the command procedure to test its return code. If the test you request is true, you have the option of ending the command procedure or of executing another procedure or another command. If the test you request is not true, the command procedure will continue its course. The test is specified with the SYSRC operand of the WHEN statement. For example, assume that you want to end a procedure if a given CALL command produces a return code of 8. Enter the following WHEN statement after the command you want to test:

```
.
.
.
call 'sys1.linklib(ieqcb100)' 'nodeck'
when sysrc(eq 8) end
.
.
.
```

If instead of ending the procedure when the test is true, you want to execute another procedure that resides in the JONES.PROC5.CLIST data set, enter:

```
.
.
.
when sysrc(eq 8) exec proc5
.
.
.
```

If instead of executing a procedure, you want to enter a LIST command, enter:

```
.
.
.
when sysrc(eq 8) list pgm.list snum
.
.
.
```

ENDING THE COMMAND PROCEDURE

You must write an END statement after the last line of the command procedure. When the system encounters an END statement in a command procedure it sends a READY message to the terminal so you can enter another command.

Controlling a System with TSO

Two commands are used to control TSO: OPERATOR and ACCOUNT. The OPERATOR command is used to regulate the operation of the system from a terminal. The ACCOUNT command is used to maintain the list of authorized users of the system.

You must have authorization from your installation to use either the OPERATOR or the ACCOUNT command. This authorization is recorded in the system with your user attributes. Use of the OPERATOR command is restricted to terminals that have the transmit-interrupt capability.

The Operator Command

The OPERATOR command, through its subcommands, allows you to perform the following functions:

- Monitor terminal activity (MONITOR and STOP subcommands).
- Display TSO information (DISPLAY subcommand).
- Cancel a terminal session or a background job (CANCEL subcommand).
- Send messages to terminal users (SEND subcommand).
- Modify time sharing parameters (MODIFY subcommand).
- End operation of the OPERATOR command (END subcommand).

You must first enter the command and then the subcommand you wish to use. For example, use the following sequence to enter the MONITOR subcommand:

```
READY
operator
OPERATOR
monitor...
```

For further information on system operator commands and procedures refer to the publications, IBM System/360 Operating System: Time Sharing Option, Command Language Reference, and IBM System/360 Operating System: Operator's Procedures, GC28-6692.

MONITORING TERMINAL ACTIVITY

The MONITOR subcommand lets you keep track of the users of the system and of any background jobs submitted with the SUBMIT command.

If you want to be notified whenever a terminal session starts or ends, enter the SESS operand of the MONITOR subcommand. For example, after using the following sequence:

```
READY
operator
OPERATOR
monitor sess
```

you will receive messages, such as the following, interspersed with other messages and input at your terminal:

```
IEF125I JONES LOGGED ON
          .
          .
          .
IEF125I SMITH LOGGED ON
          .
          .
          .
IEF126I JONES LOGGED OFF
          .
          .
          .
IEF125I BROWN LOGGED ON
          .
          .
          .
IEF126I BROWN LOGGED OFF
          .
          .
          .
IEF126I SMITH LOGGED OFF
          .
          .
          .
```

The message informing you that a user logged on, consists of his user identification, for example,

```
JONES LOGGED ON
```

The message informing you that a user's session has ended (logged off) consists of the user identification and the words "LOGGED OFF", for example,

```
JONES LOGGED OFF
```

You can also request the time at which the session starts and ends as part of the message. You do this by entering SESS,T with the MONITOR subcommand. For example, if you enter:

```
monitor  sess,t
```

the message informing you that JONES logged on may appear as follows:

```
IEF125I JONES LOGGED ON TIME = 1.35.05
```

The LOGON time is shown in hours, minutes and seconds.

If you want the name of each background job submitted during a terminal session displayed when the job starts and ends, you must enter another MONITOR subcommand. For example, after using the following sequence:

```
OPERATOR
monitor jobnames
```

you will start receiving messages, such as the following, interspersed with other messages and input at your terminal:

```
IEF403I JONES79 STARTED
        .
        .
        .
IEF403I COPYDS STARTED
        .
        .
        .
IEF404I JONES79 ENDED
        .
        .
        .
IEF404I COPYDS ENDED
```

The message informing you that a background job started execution, consists of the jobname, for example,

```
IEF403I JONES79 STARTED
```

The message informing you that a background job has ended consists of the jobname and the word "ENDED", for example,

```
IEF404I JONES79 ENDED
```

You can also request the time at which the background job starts and ends as part of the message. You do this by entering JOBNAMES,T in the subcommand. For example, if you enter:

```
monitor jobnames,t
```

the message informing you that job COPYDS ended may appear as follows:

```
IEF404I COPYDS ENDED TIME = 17.11.58
```

where the time the background job ended is shown in hours, minutes, and seconds.

You can also use MONITOR subcommands to obtain information on data sets and space available on direct access devices. The following subcommand:

```
monitor status
```

requests that the data set names and volume serial numbers be displayed whenever data sets with dispositions of KEEP, CATLG, or UNCATLG are unallocated.

The following subcommand:

```
monitor space
```

requests that the system display in demount messages the amount of space available in a direct access device. (Demount messages are explained in the publication IBM System/360 Operating System: Operator's Procedures.)

The following subcommand:

```
monitor dsname
```

requests that the system display within the mount and K-type demount messages, the name of the first nontemporary data set allocated to the volume to which the message refers. (These concepts are explaned in the publication IBM System/360 Operating System: Operator's Procedures.)

You can use the STOP subcommand to stop the monitoring operations of the MONITOR subcommand. For example, if you issue the following subcommands:

```
READY
operator
OPERATOR
monitor jobnames, t
monitor space
monitor status
monitor sess
```

and you want to stop receiving messages about background jobs and freed data sets, enter:

```
stop jobnames
stop status
```

DISPLAYING TSO INFORMATION

You can use the DISPLAY subcommand to obtain information about users currently logged on. If you enter:

```
display user
```

you will get the number of active terminals, the identification of each user and the corresponding region number of each user. If you want to know only the number of active terminals, enter:

```
display user=nmbr
```

You can also use DISPLAY to obtain a list of the jobnames of background jobs on the input, hold, output, BRDR, and ASB queues. (These queues are described in the publication IBM System/360 Operating System: Operator's Procedures.) To obtain this list enter:

```
display n
```

If you want only the jobnames in up to four specific queues enter the input work queue name (A-O), SOUT for system output queues, BRDR for background reader, or HOLD for system hold queue. For example, if you want the jobnames of background jobs in queues B, F, M, and the hold queue, enter:

```
display n=(b,f,m,hold)
```

If you want to know only the number of entries on the input, hold, output, BRDR and ASB queues, enter:

```
display q
```

You can also obtain the number of entries in up to four specific queues, for example:

```
display q=(b,f,m,hold)
```

You can enter a jobname as the operand of DISPLAY to obtain status information about that job. The status information consists of jobname, class, job priority, type of queue the job is in, and the job's position in the queue. For example, to obtain the status of job JONES79, enter:

```
display jones79
```

DISPLAY also lets you obtain a listing of messages from background jobs that are awaiting reply from an operator. To obtain such a listing enter:

```
display r
```

If you want to know the time of day and the date, enter:

```
display t
```

CANCELLING A SESSION OR BACKGROUND JOB

You can use the CANCEL subcommand of the OPERATOR command tc cancel a terminal session or a background job submitted by a terminal user. To cancel a session enter the U=user identification operand in the CANCEL subcommand. For example, if you want to cancel the session of user SMITH, enter:

```
cancel u=smith
```

SMITH will be presented with information that notifies him of the end of his session.

To cancel a background job, enter its jobname in the CANCEL subcommand. For example, if you want to cancel job AB999, enter:

```
cancel ab999
```

You can also request that when the job is cancelled a dump be taken of any step of that job currently being executed, for example,

```
cancel ab999,dump
```

In addition to the dump, you can request that all input and output for the job be cancelled. For example,

```
cancel ab999,dump,all
```

SENDING MESSAGES TO TERMINAL USERS

You can use the SEND subcommand to send broadcast messages (notices) to all users or to individual users. For example, if you want to send the message TSO NOT AVAILABLE ON TUESDAY 9/29 to all users, enter:

```
send 'tso not available on tuesday 9/29'
```

If you only want users SMITH and JONES to receive the message, enter:

```
send 'tso not available on tuesday 9/29', user=(smith jones)
```

SMITH and JONES will receive the message only if they are logged on. If you want to make sure that they receive the message when they log on, enter

```
send 'tso not available on tuesday 9/29', user=(smith jones) logon
```

When the LOGON operand is specified and Smith and Jones are already logged on, the message will be put in the Broadcast Data Set. It will be issued to the specified user only when he enters either LISTBC or another LOGON command.

Messages that you send to all users are given a number and are retained by the system. If you want to receive a list of all retained messages, enter

```
send list
```

If you want to delete a given message, enter its number in the SEND subcommand. For example, if you want to delete message number three enter:

```
send 3
```

If you want to list a given message without deleting it, enter the LIST operand. For example

```
send 3,list
```

MODIFYING TIME SHARING PARAMETERS

You can use the MODIFY subcommand to change the time sharing parameters specified during system generation or specified by the system operator with the START command. For information on this subcommand refer to the publication, IBM System/360 Operating System: Time Sharing Option, Command Language Reference, and IBM System/360 Operating System: Operator's Procedures.

ENDING OPERATION OF THE OPERATOR COMMAND

Whenever you want to end the OPERATOR command, enter the END subcommand. After you enter the END subcommand you receive the READY message. You can then enter any command you choose.

The Account Command

The user attributes of each authorized user of TSO are recorded in the User Attribute Data Set (UADS). There is an entry in the UADS for each user. Each entry contains:

1. A single user identification.
2. One or more passwords, or a single null field, associated with the user identification.
3. One or more account numbers, or a single null field, associated with each password.
4. One or more procedure names associated with each account number. Each procedure name identifies a LOGON procedure that is invoked when the user begins a terminal session by entering the LOGON command.
5. The main storage region size requirements for each procedure.
6. The name of the group of devices that the user is allowed to use when he does not request specific devices.
7. The authority to use, or a restriction against using, the ACCOUNT command.

8. The authority to use, or a restriction against using, the OPERATOR command.

9. The authority to use, or a restriction against using, the SUBMIT, STATUS, CANCEL, and OUTPUT commands.

10. The maximum main storage region size authorized for this user.

Figure 10 shows the simplest structure that an entry in the UADS can have, and Figure 11 shows a more complex structure.

The ACCOUNT command allows you to update entries in the UADS. Specifically, it allows you to:

- Add new entries or more data to an existing entry.
- Delete entries or parts of entries.
- Change data in an entry.
- Display the contents of an entry.
- Display the user identifications for all entries.
- End operation of the command.

These functions are performed with the subcommands of the ACCOUNT command. You must first enter the command and then the subcommand you want to use. For example, use the following sequence before entering the ADD subcommand:

```
READY
account
ACCOUNT
add...
```

ADDING NEW ENTRIES OR DATA TO AN ENTRY

You can use the ADD subcommand to add a new entry to the UADS or to add new data to an existing entry.

To add a new entry, enter the user identification, password, account or procedure name. For example, to add the following entry:

```
JONES
  ↓
ZZZ
  ↓
D993
  ↓
PROCAB
```

enter

```
add (jcnes zzz d993 procab)
```

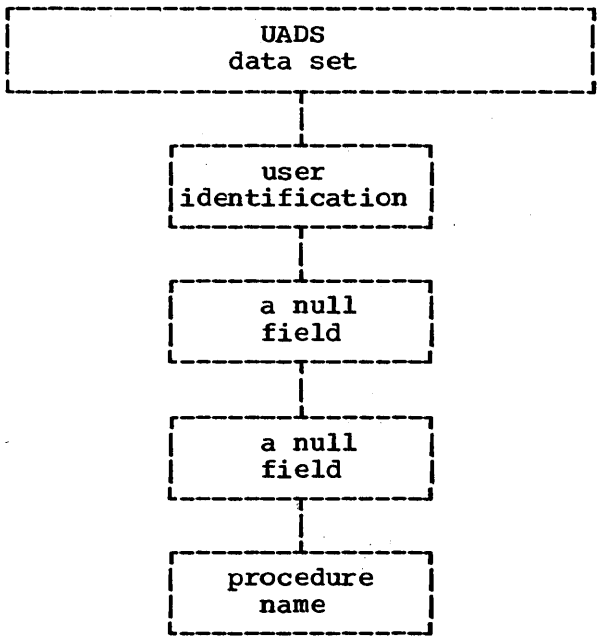


Figure 10. The Simplest Structure That an Entry in the UADS Can Have

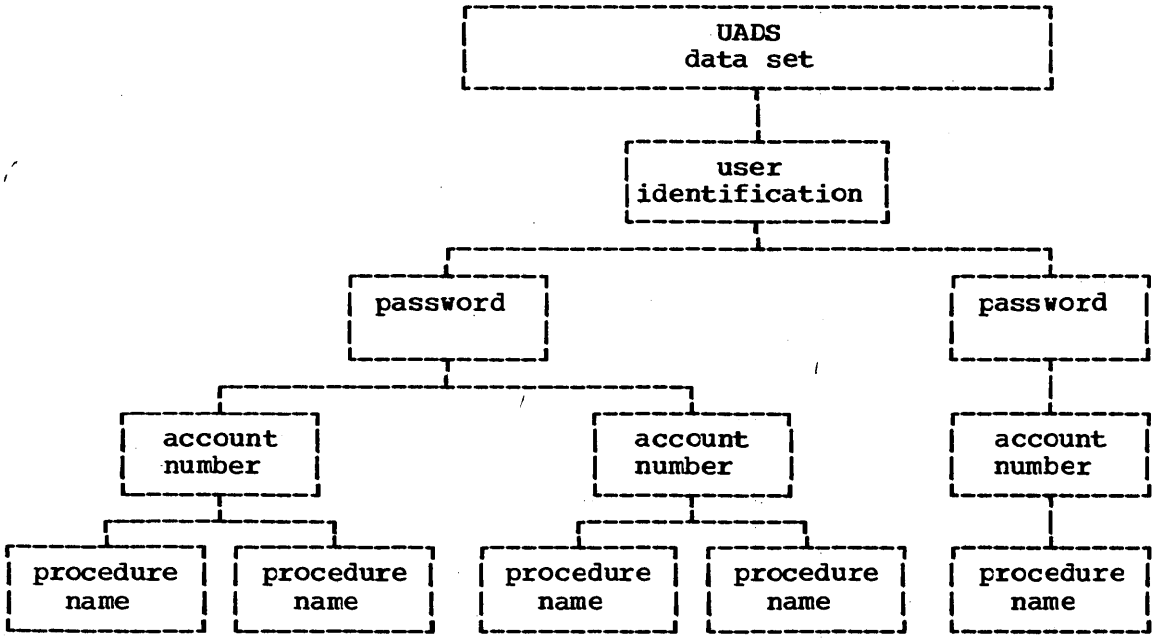


Figure 11. A Complex Structure for an Entry in the UADS

If either the password or the account (or both) is a null field, enter an asterisk to indicate its absence. For example, to add the following entry:

```
SMITH
  ↓
null
  ↓
null
  ↓
PRO7
```

enter

```
add (smith * * pro7)
```

In addition to the user identification, password, account, and procedure name, you can enter one or more of the following operands:

Operand	Meaning
SIZE(integer)	Region size (in units of 1024 bytes) of the procedure added. For example for a 10K region size specify SIZE(10). If you omit this parameter the minimum region size established by the installation is assumed.
UNIT(name)	The name of the group of devices that can be used for the user's data set.
MAXSIZE(integer)	The maximum region size (in units of 1024 bytes) that the user can request when he logs on. If you omit this parameter, no maximum limit is enforced.
ACCT	Authorization to use the ACCOUNT command.
OPER	Authorization to use the OPERATOR command.
JCL	Authorization to use the SUBMIT, STATUS, CANCEL, and OUTPUT commands.

You can use the MAXSIZE, ACCT, OPER, and JCL operands only when you are adding a complete entry to the UADS.

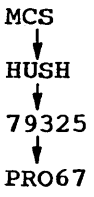
For example, if you want to add the following entry

```
BROWN
  ↓
null
  ↓
DEPT5
  ↓
PR37
```

and you also want to establish the region size for PR37 as 12K, and authorize the user to submit background jobs, enter:

```
add (brown * dept5 pr37) size(12) jcl
```

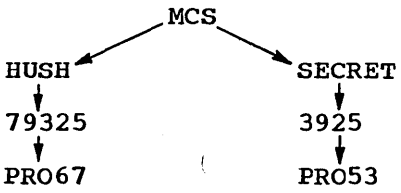
You can follow a similar procedure to add data to an existing entry. For example, assume the following entry already exists in the UADS:



If you want to add the password SECRET with account 3925 and procedure PRO53, enter:

```
add (mcs) data(secret 3925 pro53)
```

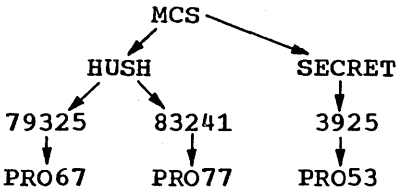
the resulting entry will be:



Now assume you want to add to password HUSH, account 83241 and procedure PRO77. Enter:

```
add (mcs hush) data (83241 pro77)
```

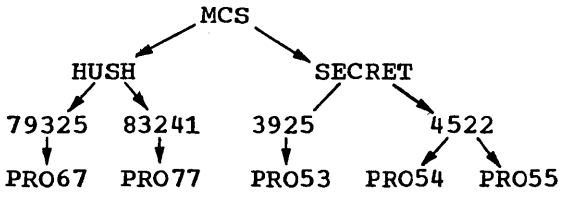
The resulting entry will be:



If you want to add account 4522 and procedures PRO54 and PRO55 to password SECRET, enter:

```
add (mcs secret) data(4522 (pro54 pro55))
```

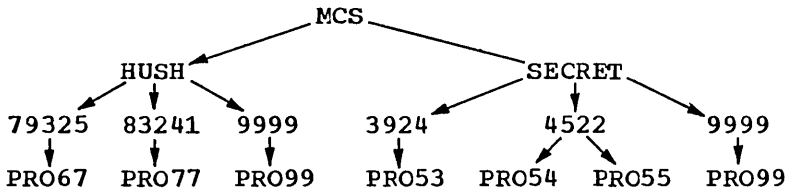
The resulting entry will be:



If you want to add the same data to all user identifications, or passwords, or account numbers, replace that field with an asterisk. For example, if you want to add account 9999 and procedure PRO99 to all passwords in the MCS entry, enter:

```
add (mcs *) data (9999 pro99)
```

The resulting entry will be:



When you are adding data to an existing entry, you can specify the SIZE operand, to give the region size of the new procedure. For example, if the region size of procedure PRO99 is 25K, enter:

```
add (mcs *) data(9999 pro99) size(25)
```

Note: You cannot add a password or an account number to an entry that has a null field for that item. You must delete the old entry that has the null fields, then add a new entry including the new password and account number.

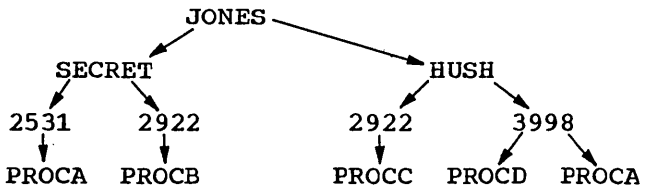
DELETING ENTRIES OR PARTS OF ENTRIES

You can use the DELETE subcommand to delete an entry or portions of an entry.

To delete an entire entry, simply enter the user identification in the DELETE subcommand. For example, to delete the entry for SMITH, enter:

```
delete (smith)
```

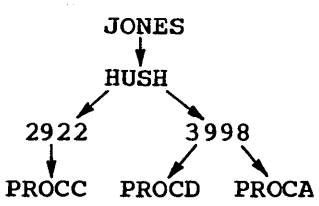
To delete a password, and consequently all accounts and procedures associated with the password, enter the password in the DATA operand. For example, assume the following entry:



If you want to delete password SECRET and its accounts and procedures, enter:

```
delete (jones) data(secret)
```

The resultant entry is:

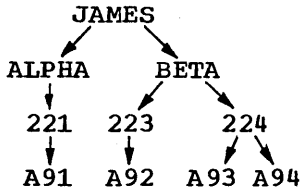


If the password happens to be the only password associated with the user identification, the entire entry is deleted. For example, if you now enter:

```
delete (jones) data(hush)
```

the entire entry is deleted.

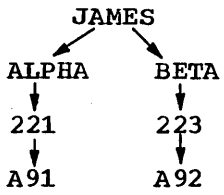
To delete an account number, and consequently all procedures associated with the account, enter the account number in the DATA operand. For example, assume the following entry:



To delete account 224 and its procedures, enter:

```
delete (james beta) data(224)
```

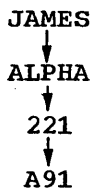
the resultant entry is:



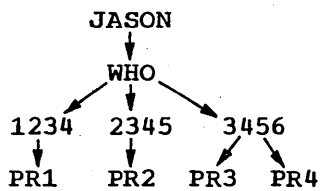
If the account number happens to be the only account associated with the password, then the password is also deleted. For example, if you now enter:

```
delete (james beta) data(223)
```

The resultant entry is:



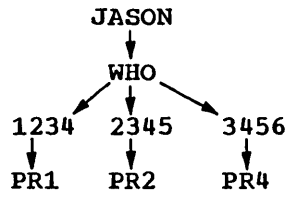
To delete a procedure, enter the procedure name in the DATA operand. For example, assume the following entry:



To delete procedure PR3, enter:

```
delete (jason who 3456) data(pr3)
```

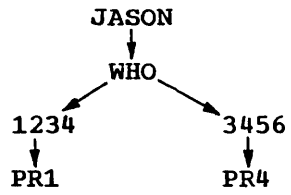
the resultant entry is:



If the procedure happens to be the only procedure associated with the account, then the account is also deleted. For example, if you now enter:

```
delete (jason who 2345) data(pr2)
```

the resultant entry is:



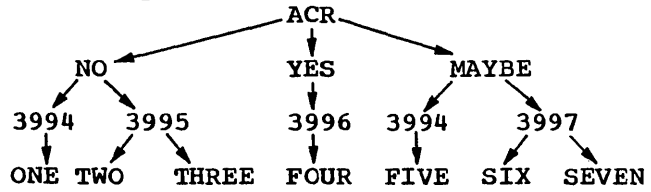
If you want to delete the same data from all user identifications, or passwords, or account numbers, replace that field with an asterisk. For example, if you want to delete password SECRET from all user identifications in the system, enter:

```
delete (*) data(secret)
```

To delete account 3994 from all passwords in the system, enter:

```
delete (* *) data (3994)
```

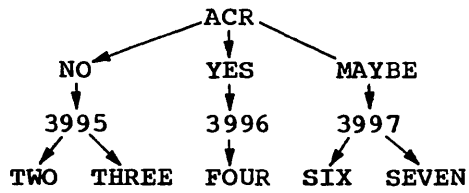
If you only want to delete account 3994 from all passwords in the following entry:



enter:

```
delete (acr *) data(3994)
```

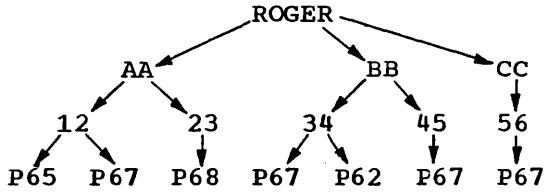
the resultant entry is:



To delete procedure P67 from all account numbers in the system, enter:

```
delete (* * *) data(p67)
```

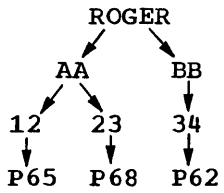
If you only want to delete procedure P67 from all accounts in the following entry:



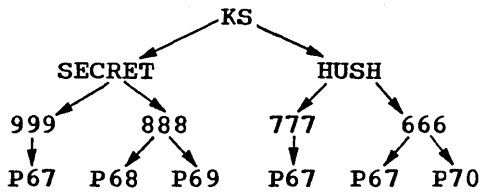
enter:

```
delete (roger * *) data(p67)
```

the resultant entry is:



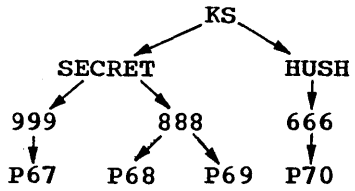
To delete procedure P67 from all accounts under password HUSH of the following entry:



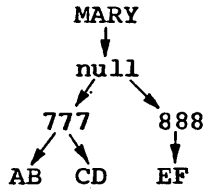
enter:

```
delete (ks hush *) data(p67)
```

the resultant entry is:



The asterisk is also used to denote a null field. For example, assume the following entry:



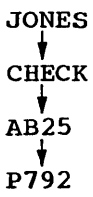
to delete procedure CD, enter:

```
delete (mary * 777) data(cd)
```

Note: You cannot delete a null field.

CHANGING DATA IN AN ENTRY

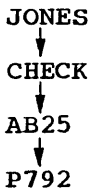
You can use the CHANGE subcommand to change any item of data in an UADS entry. For example, if you have the following entry:



and you want to change the user identification to SMITH, enter:

```
change (jones) data(smith)
```

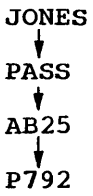
If you have the following entry:



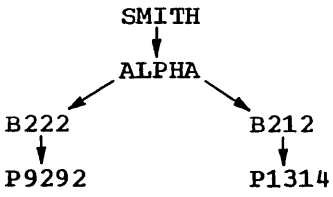
and would like to change password CHECK to PASS, enter:

```
change (jones check) data(pass)
```

The resultant entry will be:



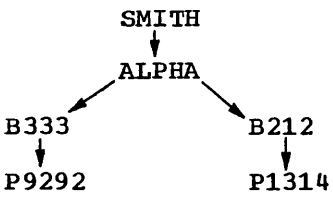
If you have the following entry:



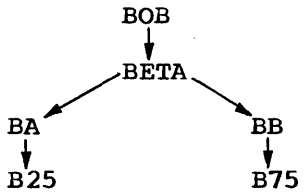
and would like to change account B222 to B333, enter:

```
change (smith alpha b222) data(b333)
```

The result will be:



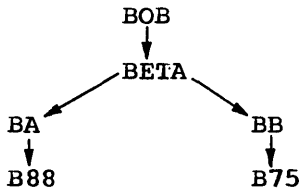
If you have the following entry:



and would like to change procedure B25 to B88, enter:

```
change (bob beta ba b25) data(b88)
```

The result will be:



In addition to changing the user identification, passwords, accounts, and procedures, you can change any user attributes. For example, if you want to authorize user JONES to use the OPERATOR command, enter:

```
change (jones) oper
```

If you want to take away the authorization to submit background jobs from user SMITH, enter:

```
change (smith) nojcl
```

DISPLAYING THE CONTENTS OF AN ENTRY

You can use the LIST subcommand to display the contents of all entries in the UADS, of one particular entry, or of parts of one entry. To display the contents of all entries, enter:

```
list (*)
```

To display the contents of entry GREEN, enter:

```
list (green)
```

If you want to display all the account numbers under password BBB of entry GREEN, enter:

```
list (green bbb)
```

If you want to display all the procedures in account 3399 of password BBB of entry GREEN, enter:

```
list (green bbb 3399)
```

DISPLAYING ALL USER IDENTIFICATIONS

You can use the LISTIDS subcommand to list all user identifications in the UADS. The contents of each entry will not be displayed. To list the user identifications, enter:

```
listids
```

ENDING OPERATION OF THE ACCOUNT COMMAND

When you want to end the ACCOUNT command, enter the END subcommand. After you enter the END subcommand you receive the READY message. You can then enter any command you choose.

Glossary

The following are definitions of words and phrases used in this publication.

abnormal end of task (ABEND): Termination of a task prior to normal completion because of an error condition.

address: The location of information in main storage.

alias: An alternate name for a particular member of a partitioned data set.

allocate: To assign a resource for use in performing a specific task.

attention interruption: An interruption of instruction execution caused by a terminal user pressing the attention key. See also "simulated attention".

attention key: A function key that is used to cause an attention interruption.

BASIC: An algebra-like language used for problem solving by engineers, scientists, and others who may not be professional programmers.

broadcast data set: A system data set containing messages and notices from the system operator, administrators, and other terminal users.

catalog:

1. noun: In the System/360 Operating System, a collection of data set indexes that are used by the control program to locate a volume containing a specific data set.
2. verb: To include the volume identification of a data set in the catalog.

cataloged data set: The quality attributed to a data set whose name and location are stored in the system catalog. A data set that is represented in an index or hierarchy of indexes which provide the means for locating the data set.

character-deletion character: A character within a line of terminal input specifying that the immediately preceding character is to be deleted from the line.

Code and go FORTRAN: A version of FORTRAN IV modified for rapid compilation and execution of programs.

command: Under TSO, a request from a terminal for the execution of a particular program called a command processor. The command processor is in a command library under the command name. Any subsequent commands processed directly by that command processor are called subcommands. The command processor performs the function that the user requested.

command language: The set of commands, subcommands and operands, recognized by TSO.

command name: The first term in a command, usually followed by operands.

command procedure: A data set or member of a partitioned data set containing TSO commands to be performed sequentially by the EXEC command.

communication line: Any medium such as a wire or a telephone circuit, that connects a terminal with a computer.

compile: To prepare a machine language program from a computer program written in a high-level source language.

computing system: A central processing unit with main storage, input/output channels, control units, storage devices, and input/output devices connected to it.

console: The computer hardware that is used by the system operator to operate the system.

context editing: A method of editing a line data set without using line numbers. To refer to a particular line, all or part of the contents of that line are specified.

control dictionary: The external symbol dictionary and relocation dictionary, collectively, of an object or load module.

control terminal: Any terminal at which a TSO user authorized to enter commands affecting system execution is logged on.

conversational: Describing a program or a system that carries on a dialog with a terminal user, alternately accepting input and then responding to the input quickly enough for the user to maintain his train of thought.

current line pointer: A pointer maintained by the Edit command processor that indicates the line of a line data set with

which a user is currently working. A terminal user can refer to the value of the current line pointer by entering an asterisk (*) with EDIT subcommands.

data: Information used as a basis for calculation, measurement and decision.

data set:
1. A collection of data that is accessible by the system. The data set usually resides on an auxiliary storage device.

2. A telephone device used to transmit telecommunications data.

data set catalog: (See catalog).

data set name: The term or phrase used to identify a data set (see qualified name).

debug: To detect, locate, and remove mistakes from a routine.

default option: A language statement option that is selected by the operating system control program or a processing program in the absence of a selection by a user.

delimiter: A character that groups or separates words or values in a line of input.

device type: Usually, the general name for a kind of device, specified at the time the system is generated. For example, 2311 or 2400.

dump (main storage):

1. verb: To copy the contents of all or part of main storage onto an output device.

2. noun: The data resulting from (1).

3. noun: A routine that will accomplish (1).

edit mode: Under the EDIT command an entry mode that accepts successive subcommands suitable for modifying an existing line data set.

external reference: The use of a name or symbol defined in another module or program.

external symbol: A control section name, entry point name, or external reference; a symbol in the external symbol dictionary.

field, data: One or more items of information that together make up a record such as an account number or the name of a person.

file name: A name of a collection of data (the file name corresponds to the data definition name).

foreground job: For TSO, a program executed in a region devoted to time sharing operations.

function key: A terminal key, such as the attention key, that causes the transmission of a signal not associated with a character. Detection of the signal usually causes the system to perform a predefined operation for the user.

IBM System/360: A collection of computing system devices that can be connected together in many combinations to produce a wide range of unique and unified computing systems. Although the systems vary in size and performance, they share many characteristics, including a common machine language.

IBM System/360 Operating System: An application of the System/360 computing system, in the form of program and data resources, that is specifically designed for use in creating and controlling the performance of other applications. TSO is an optional facility of the Operating System.

input device: A machine used to enter data into the system.

input mode: Under the EDIT command an entry mode that accepts successive lines of input for a line data set. The lines are not checked for the presence of subcommands.

installation: A general term for a particular computing system, in the context of the overall function it serves and the individuals who manage it, operate it, apply it to problems, maintain it, and use the results it produces.

interruption: A transfer of CPU control to the control program of the Operating System. The transfer is initiated automatically by the computing system or by a problem state program through the execution of a supervisor call (SVC) instruction. The transfer of control occurs in such a way that control can later be restored to the interrupted program, or, in systems that perform more than one task at a time, to a different program.

ITF:BASIC: A conversational subset of BASIC designed for ease of use at a terminal.

ITF:PL/I: A conversational subset of PL/I designed for ease of use at a terminal.

job:

1. In the background environment, a collection of related problem programs, identified in the input stream by a JOB statement followed by one or more EXEC and DD statements.
2. In the foreground environment, the processing done on behalf of one user from LOGON to LOGOFF -- one terminal session.

Job Control Language: A high-level programming language used to code statements that control the initiation and execution of jobs.

job definitions: A series of job control statements that define a job. (See job.)

job library: A set of user-identified partitioned data sets used as the main source of load modules for a given job.

job output device: A device assigned by the operator for common use in recording output data for a series of jobs.

job (JOB) statement: A job control statement that identifies the beginning of a job. It contains information such as the name of the job, an account number, and the class and priority assigned to the job.

keyword: A command operand that consists of a specific character string (such as FORTLIB or PRINT) and optionally a parenthesized value.

language statement: A phrase that is coded by a programmer, operator, or user of a computing system. The phrase conveys information to a processor such as a language translator program, service program, or control program. A language statement may signify that an operation is to be performed or may simply contain data that is to be passed to the processing program.

language translator: Any assembler, compiler, or other routine that accepts statements in one language and produces equivalent statements in another language.

library:

1. A collection of data sets associated with a particular use and identified in a directory.
2. Any partitioned data set.

line:

1. A single line of one or more characters typed at a terminal and entered into the system.
2. A circuit, such as a telephone line, over which data is communicated.

line data set: A data set with logical records that are printable lines.

line-deletion character: A character that specifies that it and all preceding characters are to be deleted from a line of terminal input.

line number: A number associated with line in a line data set, which can be used to refer to the line.

line number editing: A mode of operation under the EDIT command in which lines to be modified are referred to by line number.

linkage editor: A program that produces a single load module from one or more object and/or load modules.

link library: A generally accessible partitioned data set which contains load modules such as those referred to by macro instructions or system facilities.

listing: A display or printout of data.

load: To place a program in main storage so that it can be executed.

loader: A program that combines the basic editing and loading functions of the linkage editor. It loads object and/or load modules into main storage for execution; however, it does not produce load modules.

load module: The output of the linkage editor; a program in a form suitable for loading into main storage for execution.

LOGOFF: The TSO command that terminates a user's terminal session.

LOGON: The TSO command that a user must enter to initiate a terminal session.

LOGON procedure: A cataloged procedure that is executed as a result of a user entering the LOGON command.

member: A partition of a partitioned data set.

message: In telecommunications, a combination of characters and symbols transmitted from one point to another on a network.

message text: A part of a teleprocessing message consisting of the information that is routed to a user at a terminal or to a program in a central system that is to process it (not including line control characters).

module: The input to, or output from, a single execution of an assembler, compiler, or linkage editor; a source, object, or load module; hence, a program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading.

name: A one to eight character alphameric term that identifies a data set, a command or control statement, a program, or a cataloged procedure. The first character of the name must be alphabetic.

national characters: The characters #, $, and @.

object module: The output of a single execution of an assembler or compiler; the output constitutes input to the linkage or loader. An object module consists of one or more control sections in relocatable, though not executable, form and an associated control dictionary.

object module library: A partitioned data set that is used to store object modules.

object program: A program that has been compiled or assembled by a language translator. (See object module.)

operand: In the TSO command language, information entered with a command name to define the data on which a command processor operates and to control the execution of the command processor. Some operands are positional, identified by their sequence in the command input line; others are identified by keywords.

operating system: An application of a computing system, in the form of organized collections of programs and data, that is specifically designed for use in creating and controlling the performance of other applications. (See IBM System/360 Operating System.)

operator: A member of a data processing installation who is responsible for directing the overall operation of a computing system.

output class: Any one of up to 36 different output data classes, defined at an installation, to which output data can be assigned.

output device: A machine (such as a printer, terminal, or tape drive) that will accept the output from the system.

partitioned data set: A data set that is stored in direct access storage and can be cataloged like any other data set. A partitioned data set is often called a program library. It is divided into independent partitions called members, each of which normally contains a program or part of a program, in the form of one or more sequential blocks. Each program library contains a built-in directory (or index) that the control program can use to locate a program in the library. Each member has a unique name listed in a directory at the beginning of the data set. Members can be added or deleted as needed. Records within members are organized sequentially.

password: A one-to-eight character symbol assigned to a user that he can be required to supply at LOGON. The password is confidential, as opposed to the user identification. Users can also assign passwords to data sets.

PL/I: A high-level programming language that has features of both COBOL and FORTRAN, plus additional features.

priority: A rank assigned to a task that determines its precedence in receiving system resources.

private library: A partitioned data set other than the link library or the job library.

processor: A program performing some fixed function on input, such as a compiler or the linkage editor.

profile (user): The set of characteristics that describe the user to the system.

program: A logically self-contained sequence of instructions that can be executed by a computing system to attain a specific result.

program library: A partitioned data set containing programs in load module form for general or assorted applications.

prompting: A system function that helps a terminal user by requesting him to supply operands necessary to continue processing.

qualified name: A data set name that is composed of two or more names separated by periods. (For example, MOORE.SALES.JUNE.)

record: One or more data fields that represent an organized body of related data, such as all of the basic accounting

information concerning a single sales transaction.

receive interruption: The interruption of a transmission to a terminal by a higher priority transmission from the terminal. Also called a "break".

region: An area of main storage allocated to a job step and assigned a unique storage protection key. Time sharing jobs share regions. Each job occupies a region briefly, then is swapped out to auxiliary storage and another job is swapped into the vacated main storage area for execution. The jobs are swapped in and out until they are completed.

resource: Any facility of the system required by a job or task, including main storage input/output devices, the central processing unit, data sets, and control and processing programs.

return code: A number placed in a designated register (the "return code register") at the completion of a program. The number is established by user convention and may be used to influence the execution of succeeding programs or, in the case of an abnormal end of task (ABEND), it may simply be printed for programmer analysis.

session time: The elapsed real time from LOGON to LOGOFF.

service program: A processing program, such as the linkage editor, sort/merge program, or a utility program that performs specific services for a user of the program.

simulated attention: A function that allows terminals without attention keys to interrupt processing. The terminal is queried (for a specified character string meaning "attention") after a specified number of minutes of uninterrupted execution or after a specified number of lines of consecutive output.

source language: The input to a language translator; for example, FORTRAN, COBOL, PL/I.

source module: A series of language statements that represent the input to a language translator.

source module library: A partitioned data set that is used to store and retrieve a source module.

source program: A program written in a source language.

statement: A phrase consisting of words or terms of a programming language.

storage dump: A recording of the contents of main or auxiliary storage so that it can be examined by a programmer or operator. (See also "dump".)

subcommand: For TSO, a subcommand is a request for a particular operation to be performed, the particular operation falling within the scope of work requested by the command to which the subcommand applies.

symbol: A unique word, composed of as many as eight alphameric characters and beginning with an alphabetic character, which is used to identify an address, module, etc.

syntax checker: A program that tests source statements in a programming language for violations of that language's syntax.

SYSIN: A system input stream. Also, a name used as the data definition name of a data set in the input stream.

SYSOUT: A system output stream. Also, an indicator used in data definition statements to signify that a data set is to be written on a system output unit.

System catalog: (See catalog)

system output device: An output device shared by all jobs.

system generation: The process of using one operating system to assemble and link together into a coherent whole all the required, alternative, and optional parts that form a new operating system.

system library: A program library in auxiliary storage in which the various parts of an operating system are stored.

system programmer:

1. A programmer who is assigned to plan, generate, maintain, extend, and control the use of an operating system with the aim of improving the overall productivity of an installation.

2. A programmer who designs programming systems and other applications.

terminal: A device resembling a typewriter that is used to communicate with the system.

terminal job: A foreground job; a session from LOGON to LOGOFF. Also used to refer to the time sharing region assigned to a user and associated system control blocks.

terminal user: See "user".

time sharing: A method of using a computing system that allows a number of users to execute programs concurrently and to interact with them during execution.

user: Under TSO, anyone with an entry in the User Attribute Data Set; anyone eligible to log on.

user attributes: A set of parameters in the User Attribute Data Set (UADS). The parameters describe the user to the system: whether he is authorized to use the ACCOUNT command, what size main storage region he is to be assigned, etc.

User Attribute Data Set (UADS): A partitioned data set with a member for each authorized system user. Each member contains the appropriate user identifications, passwords, account numbers, LOGON procedure names, and user characteristics defining the user's profile.

user identification: A one to seven character symbol identifying each system user.

utility programs: Service programs that assist the user in organizing and maintaining data.

verification: An operation under the EDIT command in which all subcommands are acknowledged and any text changes are displayed as they are made.

Index

Indexes to systems reference library manuals are consolidated in the publication IBM System/360 Operating System: Systems Reference Library Master Index, Order No. GC28-6644. For additional information about any subject listed below, refer to other publications listed for the same subject in the Master Index.

GC28-6763-0

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

READER'S COMMENT FORM

IBM System/360 Operating System:
Time Sharing Option
Terminal User's Guide

Order No. GC28-6763-0

Please use this form to express your opinion of this publication. We are interested in your comments about its technical accuracy, organization, and completeness. All suggestions and comments become the property of IBM.

Please do not use this form to request technical information or additional copies of publications. All such requests should be directed to your IBM representative or to the IBM Branch Office serving your locality.

- Please indicate your occupation: _____
- How did you use this publication?
  - ☐ Frequently for reference in my work.
  - ☐ As an introduction to the subject.
  - ☐ As a textbook in a course.
  - ☐ For specific information on one or two subjects.
- Comments (Please include page numbers and give examples.):

- Thank you for your comments. No postage necessary if mailed in the U.S.A.

YOUR COMMENTS, PLEASE . . .

This manual is part of a library that serves as a reference source for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Note: Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Cut Along Line

Fold Fold

FIRST CLASS
PERMIT NO. 81
POUGHKEEPSIE, N.Y.

BUSINESS REPLY MAIL

NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY ...

IBM Corporation
P.O. Box 390
Poughkeepsie, N.Y. 12602

Attention: Programming Systems Publications
Department D58

Fold Fold

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

IBM **Technical Newsletter**

File Number	S360-36
Re: Order No.	GC28-6763-0
This Newsletter No.	GN28-2483
Date	June 1, 1971
Previous Newsletter Nos.	None

IBM SYSTEM/360 OPERATING SYSTEM:
TIME SHARING OPTION
TERMINAL USER'S GUIDE

This Technical Newsletter, a part of release 20.1 of IBM System/360 Operating System, provides replacement pages for the subject publication. These replacement pages remain in effect for subsequent releases unless specifically altered. Pages to be inserted and/or removed are:

Cover,2	49,50,50.1
3,4,4.1	55,56
9,10	67,68
17,18	79,80
21-23	83,84
27-32	

A change to the text or a small change to an illustration is indicated by a vertical line to the left of the change; a changed or added illustration is denoted by the symbol • to the left of the caption.

Summary of Amendments

This Technical Newsletter includes the addition of a new keyword for the PROFILE command, a subcommand for TEST, and editorial changes.

Note: Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Programming Systems Publications, P.O. Box 390, Poughkeepsie, N.Y. 12602

PRINTED IN U.S.A.

File No. S360-36
Order No. GC28-6763-0

OS

Systems Reference Library

IBM System/360 Operating System:

Time Sharing Option

Terminal User's Guide

The Time Sharing Option (TSO) of the IBM System/360 Operating System lets you use the facilities of a computer from a terminal. You define your work to the system through the TSO Command Language. This publication explains to all users of TSO how to use the TSO Command Language to perform the following functions:

- Start and end a terminal session.
- Enter and manipulate data.
- Program at the terminal.
- Test a program.
- Write and use command procedures.
- Control a system with TSO.

After becoming familiar with the information presented in this manual, you may use IBM System/360 Operating System: Time Sharing Option, Command Language Reference, GC28-6732 for review and reference.

Preface

This publication describes how to use the TSO Command Language to all TSO terminal users. The commands can be used to perform the following functions:

- Start and end a terminal session.
- Enter and manipulate data.
- Program at the terminal.
- Test a program.
- Write and use command procedures.
- Control a system with TSO.

This publication tells you what commands to use to perform these functions. For details on how to code each command, refer to the publication IBM System/360 Operating System: Time Sharing Option, Command Language Reference, GC28-6732.

Before reading this manual you should be aware of three facts:

- Program Products are not discussed in this manual.
- All examples in this manual show the user's input in lowercase letters and the system output in uppercase letters.
- All examples in this manual assume that you are using an IBM 2741 Communications Terminal, and that you must press the RETURN key to enter data. For information on your type of terminal refer to the publication IBM System/360 Operating System: Time Sharing Option, Terminals, GC28-6762.

First Edition (March, 1971)

This edition with the addition of Technical Newsletter GN28-2483, applies to release 20.1, of IBM System/360 Operating System, and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Changes are continually made to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/360 SRL Newsletter, Order No. GN20-0360, for the editions that are applicable and current.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Programming Systems Publications, Department D58, PO Box 390, Poughkeepsie, N.Y. 12602

Contents

Illustrations

Figures

Tables

Summary of Major Changes

Release 20.1

Item	Description	Areas Affected
CTLX keyword for PROFILE command	Keyword added.	18
COPY subcommand of TEST	Subcommand added.	10

When you use a command to request work, the command establishes the scope of the work to the system. For some commands, the scope of the work encompasses several operations that you can identify separately. After entering the command, you may specify one of the separately identifiable operations by entering a subcommand. A subcommand, like a command, is a request for work; however, the work requested by a subcommand is a particular operation within the scope of work established by a command.

The commands and subcommands recognized by TSO form the TSO command language. The command language is designed to be easy to use. The command names and subcommand names are typically familiar English words, often verbs, that describe the work to be done. The number of command names and subcommand names that you must learn has been kept to a minimum. (Your installation can add its own commands to perform functions not provided by the TSO command language.)

Besides entering the name of the command or subcommand, you are often required to specify additional information to pinpoint the function you want performed. You define the additional information with operands (words or numbers that accompany the command names and subcommand names.) Most of the operands have default values that are used by the system if you choose to omit the operand from the command or subcommand. However, some operands do not have default values. If you fail to provide a required operand for which there is no default, the system sends you a prompting message asking you to supply the operands. The publication, IBM System/360 Operating System: Time Sharing Option, Command Language Reference shows all operands for each command, and indicates the default values where applicable, and describes how to enter the commands.

You can abbreviate many of the command names, subcommand names and operands. Together, the defaults and abbreviations decrease the amount of typing required. (The abbreviations and their use are discussed in the publication IBM System/360 Operating System: Time Sharing Option, Command Language Reference.)

Table 1 lists the commands and their subcommands in alphabetical order.

Table 1. TSO Commands and Subcommands, Including Abbreviations

COMMAND (abbreviation) SUBCOMMAND (abbreviation)	COMMAND (abbreviation) SUBCOMMAND (abbreviation)
ACCOUNT	LISTDS (LISTD)
ADD (A)	LOADGO (LOAD)
CHANGE (C)	LOGOFF
DELETE (D)	LOGON
END	*MERGE
HELP (H)	OPERATOR (OPER)
LIST (L)	CANCEL (C)
LISTIDS (LISTI)	DISPLAY (D)
ALLOCATE (ALLOC)	END
*ASM	HELP (H)
*CALC	MODIFY (F)
*DELETE	MONITOR (MN)
*END	SEND
*HELP	START(S)
*SAVE	STOP (P)
CALL	OUTPUT (OUT)
CANCEL	CONTINUE (CONT)
*COBOL (COB)	END
*CONVERT (CON)	HELP (H)
*COPY	SAVE (S)
DELETE (D)	PROFILE (PROF)
EDIT (E)	PROTECT (PROT)
BOTTOM (B)	RENAME (REN)
CHANGE (C)	RUN (R)
DELETE (D)	SEND (SE)
DOWN	STATUS (ST)
END	SUBMIT (SUB)
FIND (F)	TERMINAL (TERM)
*FORMAT (FORM)	TEST (T)
HELP (H)	Assign (=)
INPUT (I)	AT
INSERT (IN)	CALL
LIST (L)	COPY (C)
*MERGE (M)	DELETE (D)
PROFILE (PROF)	DROP
RENUM (REN)	END
RUN (R)	EQUATE (EQ)
SAVE (S)	FREEMAIN (FREE)
SCAN (SC)	GETMAIN (GET)
TABSET (TAB)	GO
TOP	HELP (H)
UP	LIST (L)
VERIFY (V)	LISTDCB
EXEC (EX)	LISTDEB
*FORMAT (FORM)	LISTMAP
*FORT	LISTPSW
FREE	LISTTCB
HELP (H)	LOAD
LINK	OFF
*LIST (L)	QUALIFY (Q)
LISTALC (LISTA)	RUN (R)
LISTBC (LISTB)	WHERE (W)
LISTCAT (LISTC)	TIME (TI)
	**END
	**WHEN

*Available as program products
**For use in command procedures

Starting and Ending a Terminal Session

This section describes the commands you can use to:

- Identify yourself to the system.
- Define operational characteristics of your session.
- Receive and send broadcast messages.
- Display session time used.
- End your terminal session.

Identifying Yourself to the System

The first thing you must do to start your terminal session is to turn on the power according to instructions provided by your installation. In many cases, you will find an instruction sheet such as the one shown in Figure 1 attached to the terminal. In the example shown in Figure 1, instructions 1 through 8 must be followed to turn on the power and to establish the connection with the system. If there is no instruction sheet attached to the terminal, consult the publication, IBM System/360 Operating System: Time Sharing Option, Terminals.

After you turn on the power you must use the LOGON command to identify yourself to the system. You must supply, as operands of LOGON, the user attributes assigned to you by your installation. Your user attributes are:

1. User identification (required) -- The name or code by which you are known to the system.

2. Password (required if your installation assigns you one) -- A further identification used for additional security protection.

3. Account number (optional) -- The account to which your terminal session is charged.

4. Procedure name (optional) -- The name of series of statements that defines your job to the system.

TERMINAL #7

(Available 9:00 a.m. - 3:00 p.m.
For additional time call A. Jones ext. 1234)

1. Turn ON/OFF switch to ON.

2. Make sure the COM/LCL switch is set to COM.

3. Remove handset from telephone (data set).

4. Press TALK button on telephone.

5. Dial ext. 5555, 5556, or 5557.

6. Wait for a high pitched tone. When you hear this tone you are in contact with the computer. If you get a busy signal or no answer, hang up and repeat from 3 trying another extension.

7. Push the DATA button on the telephone. If DATA button light goes off at any point during session, repeat from (3).

8. Replace handset on the cradle.

9. Enter LOGON command:

   logon___/___ acct(___) proc(___) size(___) notices nonotices mail nomail

   userid password account procedure nnnn

10. The default TERMINAL command is:

    terminal nolines noseconds noinput break notimeout linesize (120)

    If you want to change any of the following defaults use this TERMINAL command:

    terminal lines( ) seconds( ) input( ) linesize( )

11. If you want to change your user profile, use the PROFILE command:

	char( )	line( )	prompt	intercom	pause	msgid
profile	char(bs)	line(attn)	noprompt	nointercom	nopause	nomsgid
	nochar	line(ctlx)				
		noline				

    The following operands are recommended for this terminal:
    char(bs) and line(attn)

Note: Please turn ON/OFF switch to OFF after you enter LOGOFF.

Figure 1. Sample Instruction Sheet for a Terminal

Your user attributes are recorded in the system together with the attributes of all other terminal users. When you log on, the system compares the attributes you specify with the LOGON command to the recorded attributes of each user to determine if you are an authorized user of the system.

RECEIVING BROADCAST MESSAGES

You can use three commands to control which broadcast messages you receive:

LOGON, PROFILE, and LISTBC

When you log on, broadcast messages sent to all users (notices) and those broadcast messages intended for you (mail) are displayed at your terminal. You can use the following operands of the LOGON command to prevent printing either type of messages at your terminal:

- NONOTICES suppresses printing of broadcast messages intended for all terminal users.
- NOMAIL suppresses printing of broadcast messages intended specifically for you.

For example, if you enter:

```
logon smith acct(72411) nomail
```

You will not receive mail but you will receive all notices that are available at the time.

NONOTICES and NOMAIL suppress those broadcast messages outstanding at the time you log on. You will automatically receive any broadcast messages issued after you log on. You cannot stop the operator from sending you notices, but you can specify that you do not want to receive any mail by using the NOINTERCOM operand of the PROFILE command. For example, if you enter the following commands:

```
logon jones/cloud proc(ab)
READY
profile nointercom
```

you request that all broadcast messages available at the time be displayed, but that all mail sent to you after you log on be suppressed throughout your session. (Note that NOINTERCOM can be a default of your user profile, and therefore you may not have to specify it with the PROFILE command.)

At any time during your session you can use the LISTBC command to request that either all available notices for users, or all your mail (or both) be displayed. If you enter:

```
listbc
```

you will get all broadcast messages. If you enter:

```
listbc nomail
```

you will get only notices. If you enter:

```
listbc nonotices
```

you will get only your mail.

The notices you get are both the notices available at the time you logged on and those issued throughout your session. This enables you to see what notices were available at log on time if you specified NONOTICES in your LOGON command. (The system operator can delete notices at any time. Consequently you will get only those notices he has not deleted.)

Mail messages sent directly to you are automatically deleted by the system after you receive them. Therefore the mail you get when you use the LISTBC command are those available at log on time if you specified NOMAIL in your LOGON command, and those suppressed as a result of the NOINTERCOM operand of the PROFILE command. After you use the LISTBC command to see your mail, the NOINTERCOM operand will again be in effect.

If there are no messages available when you use the LISTBC command you will receive the following message:

```
NO BROADCAST MESSAGES
```

If you want to cancel the effect of the NOINTERCOM operand, enter:

```
profile intercom
```

You will receive any mail issued after you enter this command. To obtain your mail messages issued before you entered INTERCOM, use the LISTBC command.

SENDING MESSAGES

You can use the SEND command to send mail messages to another terminal user or to a system operator. The SEND command can be used at any time after you log on.

You can send a mail message to another user only if you know his user identification. For example, the command:

```
send 'do not use procedure 245 until notified' user(jones,dept4)
```

will send the message enclosed in quotes to the two users whose identifications are JONES and DEPT4.

When you send a message to another user, he will receive it immediately provided that he is logged on and is accepting messages. If he is not logged on or is not accepting messages, you are notified and your message is deleted. For example, assume that SMITH is not logged on, JONES is not accepting messages, and CLARK is both logged on and accepting messages. When you send the following message:

```
send 'this is a message' user(smith,jones,clark)
```

SMITH and JONES do not receive the message, you are notified, and the message is deleted. CLARK receives the message.

You can request the system to save your message until the user you sent it to logs on or decides to accept messages, by using the LOGON operand of the SEND command. For example, if you enter:

```
send 'this is a message' user(smith,jones,clark) logon
```

SMITH will receive your message when he logs on, JONES will receive it when he uses the LISTBC command, and CLARK will receive it immediately.

You can send a message to only one operator at a time. With the SEND command, you can identify an operator by a number. For example,

```
send 'important message' operator(7)
```

If there is only one operator at your installation, you can omit the number. For example,

```
send 'important message' operator
```

If there are several operators and you omit the number, your message is sent to the master console.

Displaying Session Time Used

You can use the TIME command to find out how much time you have used during the current session. If you enter:

```
time
```

the system sends you a message telling you how long you have been using the terminal since you logged on.

If you are executing a program, you can use the TIME command to find out how long the program has been running. You must first enter an attention interruption and then enter the TIME command. The system then sends you a message telling you how long a program has been running. If you want to continue processing the program, press the RETURN key and the program continues. If you want to stop processing the program, enter another attention interruption and wait for the READY message before you enter another command.

Ending Your Terminal Session

You can end your terminal session in either of two ways:

- By entering the LOGOFF command to end the session.
- By entering the LOGON command to start a new session.

The LOGOFF command:

- Displays your user identification.
- Displays the length of time you have been using the terminal, and the time of day and date your session ended.
- Disconnects your terminal from the system.

The LOGON command terminates your current session and starts a new session at the same time. LOGON must be specified as described in "Identifying Yourself to the System".

the following EDIT command specifies that you are going to create a new data set named ACCTS.DATA. After you enter the command the system enters input mode.

```
READY
edit accts.data new
INPUT
```

In the following example, the OLD operand of the EDIT command specifies that you want to edit an existing data set named PARTS.TEXT. After you enter the command, the system enters edit mode.

```
READY
edit parts.text old
EDIT
```

As you can see, the NEW operand specifies that you are going to create a data set, and the OLD operand specifies that the data set already exists.

The name you give a data set should follow certain conventions. A data set name has three fields.

1. Identification qualifier.
2. User-supplied name.
3. Descriptive qualifier.

The fields must be separated by periods. The total length of the name, including periods, must not exceed 44 characters. For example, a typical data set name is:

SMITH.ACCTS.DATA

Identification qualifier

User-supplied name

Descriptive qualifier

When you create a data set you need only specify the user-supplied name. The system supplies values for the other two fields. The identification qualifier is the user identification you specified with the LOGON command. The descriptive qualifier must be one of those listed in Table 2. The system obtains it from operands you specify in the EDIT command. If you do not supply it in another operand, the system prompts you for a descriptive qualifier. If you prefer, you can specify the descriptive qualifier as part of a data set name, for example,

```
PARTS.DATA
```

You may specify a fully qualified name (a name with all three qualifiers) by enclosing it in apostrophes. For example,

```
'JONES.PROG1.ASM'
```

This is a useful procedure when you have to use a data set with an identification qualifier other than your own user identification.

Table 2. Descriptive Qualifiers

Descriptive Qualifier	Data Set Contents
ASM	Assembler (F) input
BASIC	ITF: BASIC statements
FORT	FORTRAN IV (E, G, G1 or H) statements and free- or fixed-format FORTRAN statements
IPLI	ITF:PL/I statements
PLI	PL/I (F) statements
COBOL	American National Standard COBOL statements
TEXT	Uppercase and lowercase text
DATA	Uppercase text
CNTL	JCL and SYSIN for SUBMIT command
CLIST	TSO commands
STEX	STATIC external data from ITF:PLI
OBJ	Object module
LIST	Listings
LOAD	Load module
LINKLIST	Output listing from linkage editor
LOADLIST	Output listing from loader
TESTLIST	Output listing from TEST command
OUTLIST	Output listing from OUTPUT command

Any name that does not conform to the naming conventions must be enclosed in apostrophes. For example, if you have a data set named RECORDS, with no identification or descriptive qualifiers, enter

```
'records'
```

The system will not append the identification and descriptive qualifiers to such a name.

You can refer to an existing data set by its user-supplied name. In some cases, you may also have to include the descriptive qualifier. For example, if two of your data sets were named:

```
SMITH.PART1.ASM
SMITH.PART1.DATA
```

and you want to refer to the latter, you must specify:

```
part1.data
```

You can also create and edit partitioned data sets. A partitioned data set consists of one or more data sets called members. Each member can be created and edited separately and each has a name. The member name is enclosed in parentheses and appended to the right of the fully qualified data set name. For example, the fully qualified name of member MEM1 of the SMITH.PART1.DATA data set is:

```
SMITH.PART1.DATA(MEM1)
```

You need only use the user-supplied name and member name to refer to the member. The system appends the identification and descriptive qualifiers. For example, to refer to member MEM1 you can specify:

```
part1(mem1)
```

In the following example you use the EDIT command to create member ONE of a partitioned data set named JONES.T42.DATA. The second EDIT command, creates member TWO of JONES.T42.DATA. Note that the NEW operand must be specified in both cases. The third EDIT command, specifies that changes are to be made to member ONE.

```
READY
edit t42.data(one) new
INPUT
.
.
.
READY
edit t42.data(two) new
INPUT
.
.
.
READY
edit t42.data(one) old
EDIT
.
.
.
```

After you specify the data set name and the NEW or OLD operand, you should specify the data set type. The data set type is an operand that describes the purpose for which the data set is to be or was created. The type operand is one of the sources from which the system can obtain the descriptive qualifier. The valid types are:

```
ASM
COBOL
GOFORT
FORTE
FORTG
FORTGI
FORTH
PLI
PLIF
IPLI
BASIC
DATA
TEXT
CLIST
CNTL
```

You do not have to specify the type operand if you specify it as the descriptive qualifier. For example, the following two commands have the same effect:

```
edit ab75 new asm
edit ab75.asm new
```

If the system cannot find the data set type from other sources, you are prompted for it.

If you do not want your data set to have line numbers, use the NONUM operand. For example,

```
edit ab75 new asm nonum
```

Do not specify NONUM for the BASIC, IPLI, and GOFORT data set types, because they must always have line numbers.

Except for one case, lines of input are translated to uppercase letters by the system. If you want the system to retain your input in the same form as you enter it (uppercase and lowercase), code the ASIS operand. For example,

```
edit records new data asis
```

Creating a Data Set

You usually create a data set when EDIT is in input mode. You request input mode when you enter one of the following:

- The NEW operand in the EDIT command.
- The INPUT subcommand while you are in edit mode.

After you enter the EDIT command with the NEW operand the system sends you the following message:

```
INPUT
```

After this message is printed the system prints the first line number of your data set unless you specified NONUM in the EDIT command. The first line number printed is 00010. Type the first line of input to the right of the line number and press the RETURN key to enter it. The system then prints the second line number, which is 00020, and you may then enter your second line of input, and so on. When you reach the end of the data you want to enter, press the RETURN key without entering anything (a null line) and the system switches to edit mode. The following example illustrates the points just discussed:

```
READY
edit accts new data
INPUT
00010     #23942      5     @2.75     acme inc
00020     #32135     21     @3.90     bbb corp
00030     #32174     12     @1.80     alpha inds
00040     #49213     35     @7.95     xyz dist
00050     #52221     50     @2.35     beta mfg
00060  (null line)
EDIT
```

In the example, the line numbers have the standard increment of 10. If you prefer a different increment, you can use the INPUT subcommand to create the data set. To do this you must first request a switch to edit mode by entering a null line after you receive the INPUT message. Then enter the INPUT subcommand specifying the number of the first line and the size of the increment. After entering the INPUT subcommand the system switches to input mode and prompts you with the first line number. For example, to start with line 5 and use increments of 5, you could use the following sequence:

```
READY
edit accts new data
INPUT
00010  (null line)
EDIT
input 5 5
INPUT
00005     #23942      5     @2.75     acme inc
00010     #32135     21     @3.90     bbb corp
00015     #32174     12     @1.80     alpha inds
00020     #49213     35     @7.95     xyz dist
00025     #52221     50     @2.35     beta mfg
00030  (null line)
EDIT
```

You can create the same data set in edit mode. However, you must enter the line numbers you wish to use.

```
READY
edit accts new data
INPUT
00010  (null line)
EDIT
5         #23942      5     @2.75     acme inc
10        #32135     21     @3.90     bbb corp
15        #32174     12     @1.80     alpha inds
20        #49213     35     @7.95     xyz dist
25        #52221     50     @2.35     beta mfg
```

Note: Requesting an increment larger than 1, makes it easier for you to insert lines in your data set later on. (See the section "Updating a Data Set" for instructions on how to insert lines in your data set.)

Placing Data into Columns

You can use the TAB key of your terminal to align your data in columns, just as you would with an ordinary typewriter. However, this mechanical tab setting is not recognized by the system which interprets each striking of the TAB key as a space. For example, if you enter the following three lines and align them with the TAB key, they appear at the terminal as follows:

```
39427        abcde        49211        72669        ab4
22           fghijkl      441          123456       72de
987654       mnop         2            31           xyz
```

but they are received by the system as follows:

```
39427 ABCDE 49211 72669 AB4
22 FGHIJKL 441 123456 72DE
987654 MNOP 2 31 XYZ
```

If you want the system to place your data into columns, you must establish logical tab settings with the TABSET subcommand of the EDIT command or else use the defaults provided by the system. If you have established logical tab settings for your data set, the system will receive each item in its proper column whenever you press the TAB key. The mechanical tab settings in your terminal need not correspond to the logical tab settings. For example, assume that the logical tab settings for the data set are columns 10, 20, and 30, while the mechanical tab settings in the terminal are columns 5, 10 and 15. When you type in the following three lines using the TAB key:

```
abc def  ghi  jkl
mno pqr  stu  vwx
yz0 123  456  789
                        column 15
                        column 10
                        column  5
                        column  1
```

they are received by the system as follows:

```
ABC         DEF         GHI         JKL
MNO         PQR         STU         VWX
YZ0         123         456         789
                                            column 30
                                            column 20
                                            column 10
                                            column  1
```

You may find it convenient to make the mechanical tab settings coincide with the logical tab settings.

The default tab settings used by the system vary with the data set type. They are shown in Table 3.

Table 3. Default Tab Settings

Data Set Type	Default Tab Setting Columns
COBOL	8,12,72
PLI	5,10,15,20,25,30,35,40,45,50
PLIF	5,10,15,20,25,30,35,40,45,50
FORT(ALL)	7,72
ASM	10,16,31,72
TEXT	5,10,15,20,30,40
DATA	10,20,30,40,50,60
CLIST	10,20,30,40,50,60
CNTL	10,20,30,40,50,60
IPLI	5,10,15,20,25,30,35,40,45,50
BASIC	10,20,30,40,50,60

If you want to change the default settings or other settings you previously established, or nullify all tabs, you must use the TABSET subcommand. If you want to change the default settings, you will probably do so before you create the data set. That means you must request edit mode after you enter the EDIT command, then enter the TABSET subcommand and return to the input mode to create the data set. For example, if you want to create a TEXT data set with the logical tabs at columns 10, 25, and 35, you can use the following sequence:

To delete the entire group, place an asterisk in the position where the names do not match. (The asterisk cannot replace the user identification.) For example, to delete the first group use the following:

```
READY
delete *.data
READY
```

To delete the second group use the following:

```
READY
delete weather.*
READY
```

Establishing Passwords for a Data Set

Use the PROTECT command to establish passwords for your data set. Passwords prevent unauthorized persons from reading (listing) or writing (making changes to) your data set. Whenever anyone attempts to use a password-protected data set, the system requests a password unless the data set is protected with the same password that was entered in the logon procedure. The system allows two chances to provide the correct password. If your terminal has the "print-inhibit" feature, the system disengages the printing mechanism at your terminal while you enter the password in response. However, the "print-inhibit" feature is not used if the prompting is for a new password you are adding to the data set.

The PROTECT command also specifies what the person who knows the password can do to the data set; that is, whether he is allowed to read it, or write in it, or both. You can require a password for both reading and writing; or just for reading and not writing. You can also assign one password for reading and a different one for writing. The operands that control the type of operations are:

PWREAD -- you must specify a password before you can read from the data set.

PWWRITE -- you must specify a password before you can write in the data set.

NOPWREAD -- you can read from the data set without specifying a password.

NOWRITE -- you cannot write into the data set (with this password).

There are three valid combinations of operands:

PWREAD PWWRITE -- the password is required for either reading or writing your data set.

PWREAD NOWRITE -- the password is required for reading. Writing is not allowed with this password.

NOPWREAD PWWRITE -- you can read without a password. The password allows you to both read and write the data set.

If you specify only one operand you get two values by default. They are:

Operand	Default Values
PWREAD	PWREAD PWWRITE
NOPWREAD	NOPWREAD PWWRITE
PWWRITE	NOPWREAD PWWRITE
NOWRITE	PWREAD NOWRITE

The type of password operand, the number of times the password is used, and optional security information that you can specify are recorded in the PASSWORD data set of the operating system.

The following example adds the password HUSH for reading and writing the JONES.SECRET.DATA data set:

```
READY
protect secret add(hush) pwread
READY
```

The following example adds another password, WHUSH, to the same data set. This password can be used only for reading the data set:

```
READY
protect secret/hush add(whush) nowrite
READY
```

Note how you must use the password in subsequent commands once you have established it.

You can replace a password. For example, to replace the password SESAME for HUSH in the JONES.SECRET.DATA data set, enter

```
READY
protect secret/hush replace(hush,sesame)
READY
```

Note that when you are replacing a password you do not have to specify the function of the password.

You can also delete a password. For example, if you no longer require the WHUSH password for reading the data set, enter

```
READY
protect secret/sesame delete(whush)
READY
```

You can use the DATA operand to specify optional security information to be recorded in the system. For example, when you establish the password AB#72 for the SMITH.SALES.TEXT data set, you can also specify other information:

```
READY
protect sales add(ab#72) data(password changes on monday)
READY
```

To find out what the optional information is, the type of operation allowed, and the number of times the password has been used, use the LIST operand. For example,

```
protect sales list(ab#72)
```

Note:

1. Data sets which are permanently allocated as part of the LOGON procedure or by use of the ALLOCATE command cannot be accessed by the PROTECT command. These data sets should be freed by using the FREE command prior to issuing the PROTECT command.

2. When a protected data set is renamed or deleted you should update the password data set to reflect the change. This prevents your having insufficient space for future entries.

Programming at the Terminal

You can use the TSO facilities to compile, link edit, and execute (or compile and load) your source program at the terminal. TSO also allows you to use any other program, such as utilities, at the terminal. That is, instead of taking your job to the computer room to run it directly under the operating system, you can use the TSO facilities to enter it through your terminal. These facilities reduce your job turnaround time because you get immediate results at the terminal.

You can also use the terminal to submit your job for processing at the computer in the conventional manner. That is, you submit your job through the terminal but do not want to get immediate results at the terminal. The results are sent to you from the computer room after your job is executed or you may obtain them at the terminal at a later time. Jobs submitted in this manner are called background jobs.

Most compilers or assemblers that can be used under the operating system can be used from your TSO terminal. They can be used to obtain results at the terminal, or for background jobs. In addition to these programs, your installation may have one or more of the special TSO Program Product compilers and other TSO programs for your use at the terminal. They are:

- Interactive Terminal Facility (ITF): PL/I -- A problem-solving language processor.
- Interactive Terminal Facility (ITF): BASIC -- A problem-solving language processor.
- Code and Go FORTRAN -- A FORTRAN compiler designed for a very fast compile-execute sequence at the terminal.
- FORTRAN IV (G1) -- A version of the FORTRAN IV (G) compile modified for the terminal environment.
- TSO FORTRAN Prompter -- An initialization routine to prompt you for options and invoke the FORTRAN IV (G1) Processor.
- FORTRAN IV Library (Mod I) -- Execution-time routines for use with either Code-and-Go FORTRAN or FORTRAN IV (G1).
- Full American National Standard COBOL Version 3 -- A version of the American National Standard COBOL modified for the terminal environment.
- TSO COBOL Prompter -- An initialization routine to prompt you for options and invoke the full American National Standard COBOL Version 3 Processor.
- TSO Assembler Prompter -- An initialization routine to prompt you for options and invoke the Assembler (F).

If your installation has one or more of the TSO Program Products, it will provide you with documentation that explains how to use them. This section explains how to use the programs normally available under the operating system. The following paragraphs describe how you can:

Create a program
Compile your program
Link edit a compiled program

Execute a program
Load a program
Process background jobs

It is assumed that you are familiar with a programming language and with the information in the Guide to Writing a Terminal Monitor Program or a Command Processor or Terminal User's Guide that corresponds to that language. The options and data set requirements of the compilers, linkage editor, and loader are summarized in the publication, IBM System/360 Operating System: Job Control Language User's Guide, GC28-6703.

Creating a Program

Before your source program is compiled you must introduce it to the system. You do so with the EDIT command, as described in the section, "Entering and Manipulating Data".

When you enter the EDIT command you must specify the type operand or give a descriptive qualifier to the data set name. The type (or descriptive qualifier) tells the system which programming language you are using. If you are writing a program and JCL statements to be submitted as a background job, use CNTL as the type or descriptive qualifier.

The EDIT command allows you to specify certain options for your source program. You can use the SCAN operand to request syntax checking when the data set type is GOFORT, FORTE, FORTG, FORTGI, FORTH, BASIC, PLIF, PLI, or IPLI. You can use the LINE operand specify the length of the input line for PL/I source programs. The length of the input line for the Assembler, FORTRAN, and COBOL is 80 characters.

After you create your source program you must use the SAVE subcommand to save the data set before you end the EDIT command. Your source program is now ready for compilation.

The example in Figure 4 shows the creation of an assembler source program.

```
READY
edit   prog1   new   asm
INPUT
  .
  .
  .    source program
  .
  .
EDIT
save
SAVED
end
READY
```

Figure 4. Creating an assembler source program

```
READY
edit  backpgm  new  cntl  nonum
INPUT
//smith3     job    7924,smith,msglevel=(1,1)
//step1      exec   pgm=iepck100,parm=(deck,maps,list)
//syslib     dd     dsname=sys1.coblib,disp=shr
//sysut1     dd     unit=2311,space=(trk,(50,10))
//sysut2     dd     unit=2400
//sysut3     dd     unit=2400
//sysprint   dd     sysout=a
//syspunch   dd     dsname=comp.cobol,disp=(,catlg),unit=2400
//sysin      dd     *
  .   source statements
  .
  .
  .
  .
/*
//step2      exec   pgm=loader,parm=(map,let,call)
//syslib     dd     dsname=sys1.coblib,disp=shr
//syslout    dd     sysout=a
//syslin     dd     dsname=*.step1.syspunch
//master     dd     dsname=order,disp=old
//print      dd     sysout=a
//input      dd     *
  .
  .
  .   input data
  .
  .
/*
//
(null line)
EDIT
save
SAVED
end
READY
submit  backpgm  nonotify
READY
```

Figure 8. Submitting a Program as a Background Job

DISPLAYING THE STATUS OF BACKGROUND JOBS

Any time after you submit a background job you can use the STATUS command to have its status displayed. The display will tell you whether the job is awaiting execution, is currently executing, or has executed. For example, if you want to display the status of SMITH23, enter:

```
READY
status smith23
```

If you want to know the status of all the jobs that begin with your user identification, enter the STATUS command without operands:

```
READY
status
```

CANCELLING BACKGROUND JOBS

You can use the CANCEL command to cancel execution of a background job. If the job has already been executed, the CANCEL command has no effect.

For example, if you want to cancel job JONESAB, enter:

```
READY
cancel jonesab
```

After you enter the CANCEL command, the system will send you a message telling you that the jobs specified have been cancelled.

CONTROLLING THE OUTPUT OF A BACKGROUND JOB

You can use the OUTPUT command to:

- Direct the JCL statements and system messages (MSGCLASS) and system output data sets (SYSOUT) produced by a background job to your terminal.
- Direct the MSGCLASS and SYSOUT output from a background job to a specific data set.
- Change an output class used in a background job.
- Delete the output data sets (SYSOUT) or the system messages (MSGCLASS) for background jobs.

If you use the NOTIFY operand of the SUBMIT command and you have elected to receive messages, you will receive messages. If you are not receiving messages, the message is placed in the Broadcast data set. You can then use the OUTPUT command to control the output produced by the job on the MSGCLASS and SYSOUT classes before the system processes them.

For example, assume that job GREEN 67 produces output on classes A, B, D, G, and M. If you want the output on classes G and M listed at the terminal, enter:

```
READY
output  green67  class(g m)  print(*)
```

If you want the output of class B to be listed in the GREEN.KEEP.OUTLIST data set, enter:

```
READY
output  green67  class(b)  print(keep)
```

If you want to change the output in class A to class C, enter:

```
READY
output  green67  class(a)  noprint(c)
```

Controlling a System with TSO

Two commands are used to control TSO: OPERATOR and ACCOUNT. The OPERATOR command is used to regulate the operation of the system from a terminal. The ACCOUNT command is used to maintain the list of authorized users of the system.

You must have authorization from your installation to use either the OPERATOR or the ACCOUNT command. This authorization is recorded in the system with your user attributes. Use of the OPERATOR command is restricted to terminals that have the transmit-interrupt capability.

The Operator Command

The OPERATOR command, through its subcommands, allows you to perform the following functions:

- Monitor terminal activity (MONITOR and STOP subcommands).
- Display TSO information (DISPLAY subcommand).
- Cancel a terminal session or a background job (CANCEL subcommand).
- Send messages to terminal users (SEND subcommand).
- Modify time sharing parameters (MODIFY subcommand).
- End operation of the OPERATOR command (END subcommand).

Note: The attention interruption will not halt the output from Operating System commands, such as DISPLAY ACTIVE.

You must first enter the command and then the subcommand you wish to use. For example, use the following sequence to enter the MONITOR subcommand:

```
READY
operator
OPERATOR
monitor...
```

For further information on system operator commands and procedures refer to the publications, IBM System/360 Operating System: Time Sharing Option, Command Language Reference, and IBM System/360 Operating System: Operator's Procedures, GC28-6692.

MONITORING TERMINAL ACTIVITY

The MONITOR subcommand lets you keep track of the users of the system and of any background jobs submitted with the SUBMIT command.

If you want to be notified whenever a terminal session starts or ends, enter the SESS operand of the MONITOR subcommand. For example, after using the following sequence:

```
READY
operator
OPERATOR
monitor sess
```

you will receive messages, such as the following, interspersed with other messages and input at your terminal:

```
IEF125I JONES LOGGED ON
          .
          .
          .
IEF125I SMITH LOGGED ON
          .
          .
          .
IEF126I JONES LOGGED OFF
          .
          .
          .
IEF125I BROWN LOGGED ON
          .
          .
          .
IEF126I BROWN LOGGED OFF
          .
          .
          .
IEF126I SMITH LOGGED OFF
          .
          .
          .
```

The message informing you that a user logged on, consists of his user identification, for example,

```
JONES LOGGED ON
```

The message informing you that a user's session has ended (logged off) consists of the user identification and the words "LOGGED OFF", for example,

```
JONES LOGGED OFF
```

You can also request the time at which the session starts and ends as part of the message. You do this by entering SESS,T with the MONITOR subcommand. For example, if you enter:

```
monitor sess,t
```

the message informing you that JONES logged on may appear as follows:

```
IEF125I JONES LOGGED ON TIME = 1.35.05
```

The LOGON time is shown in hours, minutes and seconds.

If you want the name of each background job submitted during a terminal session displayed when the job starts and ends, you must enter another MONITOR subcommand. For example, after using the following sequence:

```
OPERATOR
monitor jobnames
```

you will start receiving messages, such as the following, interspersed with other messages and input at your terminal:

DISPLAY also lets you obtain a listing of messages from background jobs that are awaiting reply from an operator. To obtain such a listing enter:

```
display r
```

If you want to know the time of day and the date, enter:

```
display t
```

CANCELLING A SESSION OR BACKGROUND JOB

You can use the CANCEL subcommand of the OPERATOR command to cancel a terminal session or a background job submitted by a terminal user. To cancel a session enter the U=user identification operand in the CANCEL subcommand. For example, if you want to cancel the session of user SMITH, enter:

```
cancel u=smith
```

SMITH will be presented with information that notifies him of the end of his session.

To cancel a background job, enter its jobname in the CANCEL subcommand. For example, if you want to cancel job AB999, enter:

```
cancel ab999
```

You can also request that when the job is cancelled a dump be taken of any step of that job currently being executed, for example,

```
cancel ab999,dump
```

In addition to the dump, you can request that all input and output for the job be cancelled. For example,

```
cancel ab999,dump,all
```

SENDING MESSAGES TO TERMINAL USERS

You can use the SEND subcommand to send broadcast messages (notices) to all users or to individual users. For example, if you want to send the message TSO NOT AVAILABLE ON TUESDAY 9/29 to all users, enter:

```
send 'tso not available on tuesday 9/29'
```

If you only want users SMITH and JONES to receive the message, enter:

```
send 'tso not available on tuesday 9/29',user=(smith,jones)
```

SMITH and JONES will receive the message only if they are logged on. If you want to make sure that they receive the message when they log on, enter:

```
send 'tso not available on tuesday 9/29',user=(smith,jones),logon
```

When the LOGON operand is specified and Smith and Jones are already logged on and accepting messages, the message will be sent to them immediately. If Smith and Jones are not logged on or are not accepting messages, the message will be put in the Broadcast data set. It will be issued to the specified user only after he enters either LISTBC or another LOGON command.

Messages that you send to all users are given a number and are retained by the system. If you want to receive a list of all retained messages, enter

```
send list
```

If you want to delete a given message, enter its number in the SEND subcommand. For example, if you want to delete message number three enter:

```
send 3
```

If you want to list a given message without deleting it, enter the LIST operand. For example

```
send 3,list
```

MODIFYING TIME SHARING PARAMETERS

You can use the MODIFY subcommand to change the time sharing parameters specified during system generation or specified by the system operator with the START command. For information on this subcommand refer to the publication, IBM System/360 Operating System: Time Sharing Option, Command Language Reference, and IBM System/360 Operating System: Operator's Procedures.

ENDING OPERATION OF THE OPERATOR COMMAND

Whenever you want to end the OPERATOR command, enter the END subcommand. After you enter the END subcommand you receive the READY message. You can then enter any command you choose.

The Account Command

The user attributes of each authorized user of TSO are recorded in the User Attribute Data Set (UADS). There is an entry in the UADS for each user. Each entry contains:

1. A single user identification.
2. One or more passwords, or a single null field, associated with the user identification.
3. One or more account numbers, or a single null field, associated with each password.
4. One or more procedure names associated with each account number. Each procedure name identifies a LOGON procedure that is invoked when the user begins a terminal session by entering the LOGON command.
5. The main storage region size requirements for each procedure.
6. The name of the group of devices that the user is allowed to use when he does not request specific devices.
7. The authority to use, or a restriction against using, the ACCOUNT command.

READER'S COMMENT FORM

IBM System/360 Operating System:
Time Sharing Option
Terminal User's Guide

Order No. GC28-6763-0

Please use this form to express your opinion of this publication. We are interested in your comments about its technical accuracy, organization, and completeness. All suggestions and comments become the property of IBM.

Please do not use this form to request technical information or additional copies of publications. All such requests should be directed to your IBM representative or to the IBM Branch Office serving your locality.

- Please indicate your occupation: _____
- How did you use this publication?
  - ☐ Frequently for reference in my work.
  - ☐ As an introduction to the subject.
  - ☐ As a textbook in a course.
  - ☐ For specific information on one or two subjects.
- Comments (Please include page numbers and give examples.):

- Thank you for your comments. No postage necessary if mailed in the U.S.A.

GC28-6763-0

YOUR COMMENTS, PLEASE . . .

This manual is part of a library that serves as a reference source for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Note: Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Fold Fold

FIRST CLASS
PERMIT NO. 81
POUGHKEEPSIE, N.Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY ...

IBM Corporation
P.O. Box 390
Poughkeepsie, N.Y. 12602

Attention: Programming Systems Publications
Department D58

Fold Fold

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

Cut Along Line

Printed in U.S.A. GC28-6763-0