GC28-0645-4
File No. S370-39

Systems

OS/VS2 TSO Terminal User's Guide

VS2 Release 3.7

Includes Selectable Units:

TSO/VTAM Level 1	VS2.03.813
System Security Support	5752-832
TSO/VTAM Level 2	5752-858

IBM

Fifth Edition (June, 1978)

This is a major revision of, and obsoletes, GC28-0645-3 and incorporates changes released in the following Selectable Unit Newsletters and System Library Supplements:

TSO/VTAM Level 1	VS2.03.813	GN28-2651 (dated May 28, 1976)
System Security Support	5752-832	GC28-0849 (dated May 27, 1977)
TSO/TCAM Level 2	5752-858	GD23-0044 (dated September 30, 1977)

This revision incorporates OS/VS2 MVS information formerly contained in **OS/MVT and OS/VS2 TSO Terminals**, GC28-6762, with supplement GD21-0001.

This edition applies to release 3.7 of OS/VS2 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 **System/370 Bibliography**, GC20-0001, 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, Building 706-2, PO Box 390, Poughkeepsie, N.Y. 12602. Comments become the property of IBM.

Preface

This publication explains how to use the TSO command language and the terminals supported by TSO. TSO commands, entered at a terminal, can be used to perform the following functions:

- Start and end a terminal session.
- Enter and manipulate data.
- Execute programs at the terminal.
- Test a program.
- Write and use command procedures.

This publication is intended as a guide for all TSO users and includes information specifically for new users. For details on how to code each command, refer to **OS/VS2 TSO Command Language Reference.**

Organization

The information in this book is divided into six sections and the Appendixes. The first three sections contain information that every TSO user must be familiar with, while Sections IV through VI describe more complex operations you may wish to perform. Each Appendix discusses one type of TSO-supported terminal. You need only refer to the Appendix that describes the terminal you are using.

Section I, "Basic Information for Using TSO," outlines introductory information for all system users. The section defines the format of TSO commands, system-provided aids, and the data set naming conventions.

Section II, "Starting and Ending a Terminal Session," provides a sample terminal session to familiarize new users with TSO. During the sample session the user logs on to the system, creates a data set, makes changes in the data, and logs off. The section goes on to discuss in more detail what was done during the sample session.

Section III, "Entering and Manipulating Data," describes how to use the EDIT command and its subcommands to create a data set and modify its contents. The chapter also discusses the TSO commands used to rename, protect, and delete data sets, and to list information about your data sets.

Section IV, "Executing a Program at a Terminal," explains how to compile, link-edit, and execute a program using TSO commands.

Section V, "Testing a Program at the Terminal," describes how the TEST command can be used to test a program for proper execution and programming errors.

Section VI, "Command Procedures," explains how to write and use command procedures to perform frequently repeated functions. The section describes the use of built-in functions, control variables, and command procedure statements, followed by examples of command procedures.

The **Appendixes** describe the operation and characteristics of the following TSO-supported terminals:

IBM 2741 Communication Terminal

IBM 1052 Printer-Keyboard

Teletype* Model 33

Teletype* Model 35

IBM 2260 Display Station

IBM 2265 Display Station

IBM 3270 Information Display System

IBM 3767 Communication Terminal

IBM 3770 Data Communication System

Each Appendix includes sections on how to start and end a terminal session, how to enter data, and how to interrupt operations from the terminal. (Additional information is also provided for some terminals.)

The examples in this publication:

- Assume that you are using an IBM 3270 Display Station and that you must press the ENTER key to enter data. For information on other types of terminals see the Appendixes.
- Show the user's input in lowercase letters and the system output in uppercase letters.

Related Publications

OS/VS2 TSO Command Language Reference, GC28-0646
OS/VS2 Access Method Services, GC26-3841
OS/VS2 JCL, GC28-0692
OS/VS Data Management Services Guide, GC26-3783
OS/VS Linkage Editor and Loader, GC26-3813
OS/VS2 Data Areas, SYB8-0606
OS/VS2 System Programming Library: TSO, GC28-0629
IBM System/370 Principles of Operation, GA22-7000
OS/VS TCAM Concepts and Applications, GC30-2049
OS/VS2 TCAM System Programmer's Guide, TCAM Level 10, GC30-2051
VTAM Concepts and Planning, GC27-6998
Introduction to VTAM, GC27-6987
OS/VS2 System Programming Library: VTAM, GC28-0668

See the following publications for more information on IBM 3767 and IBM 3770 terminals:

IBM 3767 Communication Terminal Operation Guide, GA18-2000
IBM 3770 Data Communication System: System Component, GA27-3097

*Trademark of Teletype Corporation

Contents

*Trademark of the Teletype Corporation

Figures

Summary of Amendments for GC28-0645-4 OS/VS2 Release 3.7

This publication contains information that was released in the following Selectable Unit Newsletters and System Library Supplements:

TSO/VTAM Level 1	(VS2.03.813)	GN28-2651
System Security Support	(5752-832)	GC28-0849
TSO/VTAM Level 2	(5752-858)	GD23-0044

Section VI: Command Procedures has been rewritten. Technical changes in this section have not been barred; therefore, the section should be read in its entirety. Also, any references to 2741 Communication Terminals and their use have been changed to 3270 Display Stations.

This publication also contains OS/VS2 MVS information that was formerly contained in **OS/MVT and OS/VS2 TSO Terminals**, GC28-6762, with supplement GD21-0001. This information is in Appendixes A through H. Technical changes in the Appendixes have not been barred; the Appendixes should be read in their entirety.

Miscellaneous editorial and technical changes have been made throughout this publication. Significant technical changes have been made to the following areas:

- "Positioning the Current Line Pointer" in Section III -- clarify use of the TOP subcommand of EDIT.
- "Ending the EDIT Functions" in Section III -- clarify use of the END and SAVE subcommands of EDIT.
- "Assigning Attributes to a Data Set" in Section IV -- clarify use of the ATTRIB command.

Summary of Amendments for GC28-0645-3 OS/VS2 Release 3.7

Changes have been made throughout this publication to reflect a Service Update to OS/VS2 Release 3.7. In addition pertinent technical and editorial changes have been made. All references to the ITF:BASIC and ITF:PLI Program Products have been deleted from this manual. As announced in P73-70, these program products have been withdrawn and reclassified to programming service classification "C" effective June 28, 1974.

Section I: Basic Information for Using TSO

- Line Continuation

Summary of Amendments for GC28-0645-2 OS/VS2 Release 3

The revision of this publication for Release 3 has focused on detail changes to ensure readability and the consistency of style, expression, and point of view. Additionally, the individual sections include the following new information and major changes:

Section I: Basic Information for Using TSO

- Abbreviating command and subcommand names
- Using alias names
- Coding TSO comments
- Canceling prompting sequences

Section II: Starting and Ending a Terminal Session

- Getting a TSO User Identification
- Sample terminal session for novice TSO users
- Modified logon procedures
- Print-inhibited passwords
- Sending messages for delayed perusal

Section III: Entering and Manipulating Data

- Renumbering lines with the new EDIT RENUM subcommand, and at the time a data set is saved
- Moving and copying data with the new EDIT MOVE and COPY subcommands

Section V: Testing a Program at a Terminal

- Completely rewritten to increase readability and intelligibility

Section VI: Command Procedures

- Completely new chapter to explain the entire scope of TSO command procedure capability, as expanded for Release 3

Introduction

TSO is a time sharing system that 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 it is processing work entered in the conventional manner in the computer room. Due to the speed of the system, however, you will be able to work almost as though you had exclusive use of the system.

You can 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.
- Develop programs written in assembler, FORTRAN, COBOL, PL/I, or other languages.
- Execute programs.

When you enter a command, the system performs the work requested by that command and sends messages back to your terminal. Messages tell you the status of your program and whether the system is ready to accept another command.

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

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

This manual explains how to perform various functions using the command language. The manual consists of the following sections:

Section I: Basic Information for Using TSO

Section II: Starting and Ending a Terminal Session

Section III: Entering and Manipulating Data

Section IV: Executing Programs at a Terminal

Section V: Testing a Program at a Terminal

Section VI: Command Procedures

The first three sections must be understood by all system users. Sections IV through VI describe specific functions that you may wish to perform.

This manual tells you how commands are used to perform the functions mentioned above. For details on how to enter each command, refer to **OS/VS2 TSO Command Language Reference.**

Section I: Basic Information for Using TSO

Before using TSO you should know how to use:

- A terminal
- TSO commands
- System-provided aids
- Data set naming conventions

Using a Terminal

A terminal session is relatively simple: a terminal user identifies himself to the system and then issues commands to request work from the system. As the session progresses, the terminal user has a variety of aids available, which he can use if he encounters any difficulties.

Entering Information at a Terminal

All TSO terminals have a typewriter-like keyboard. 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 appendixes. However, the examples in this book address only the use of an IBM 3270 Display Station. For details on how to use the 3270, see Appendixes E and F.

Correcting Typing Errors

If you wish to correct typing errors, you must correct them before you press the ENTER key. Move the cursor under the error and type the correct character. To replace a character with a space, move the cursor under the character and press the space bar.

Using TSO Commands

A command consists of a command name, usually followed by one or more operands. A command name is typically a familiar English word, that describes the function of the command; for instance, the RENAME command changes the name of a data set.

Nearly all TSO commands have abbreviations that you can use in place of the long English name for the function. These abbreviations save you entry time at the terminal. In general, these abbreviations are as short as possible, while still providing uniqueness among them; for example, you may enter "alloc" instead of "allocate", or "e" instead of "edit". However, for readability and clarity in this book, all the references to commands and all examples of their use appear in the long English form.

Operands provide the specific information required for the command to perform the requested operation. For instance, operands for the RENAME command identify the data set to be renamed and specify the new name:

RENAME	OLDNAME	NEWNAME
command name	operand-1 (old data-set-name)	operand-2 (new data-set-name)

Two types of operands are used with the commands: *positional* and *keyword.*

Positional Operands

Positional operands follow the command name in a prescribed sequence. In the command descriptions within this manual, the positional operands appear in lowercase characters. A typical positional operand is:

```
data-set-name
```

You must replace "data-set-name" with an actual data set name when you enter the command.

When you want to enter a positional operand that is a list of several names or values, the list must be within parentheses. The names or values must not include unmatched right parentheses.

Keyword Operands

Keywords are specific names or symbols that have a particular meaning to the system. You can include keywords in any order following the positional operands. In the command descriptions within this book, keywords appear in uppercase characters. A typical keyword is:

```
TEXT
```

In some cases you may specify values with a keyword. You must enter the value within parentheses following the keyword. The way a typical keyword with a value appears in this book is:

```
LINESIZE( integer )
```

Continuing this example, you would select the number of characters that you want to appear in a line and substitute that number for the "integer" when you enter the operand:

```
LINESIZE( 80 )
```

Note: If you enter conflicting keywords, the last keyword entered overrides the previous ones.

Abbreviating Keyword Operands

You may enter keywords spelled exactly as they are shown, or you may use an acceptable abbreviation. An acceptable abbreviation is as much of the keyword as is necessary to distinguish it from the other keywords of the command or subcommand; for instance, the LISTBC command has four keywords:

```
MAIL    NOTICES

NOMAIL  NONOTICES
```

The abbreviations are:

```
M   for MAIL (also MA and MAI)
NOM for NOMAIL (also NOMA and NOMAI)
NOT for NOTICES (also NOTI, NOTIC, and NOTICE)
NON for NONOTICES (also NONO, NONOT, NONOTI, NONOTIC,
    and NONOTICE)
```

Certain keyword operands may also have synonyms (aliases). Although these aliases may have shorter forms than the operands they replace, they are not abbreviations because their form is entirely different. They do, however, mean the same to TSO as the operands they replace.

An example of a keyword operand and its alias are respectively "file" and "ddname", either of which is permissible for use with the ALLOCATE command. Specifying ALLOCATE with either operand produces identical results. Where aliases are permissible, the syntax descriptions point them out in **OS/VS2 TSO Command Language Reference**.

Delimiters

When you type a command, you must separate the command name from the first operand by one or more blanks. You must separate operands by one or more blanks or a comma. Using a blank or a comma as a delimiter, you can type the LISTBC command like this:

```
LISTBC NOMAIL NONOTICES
```

or like this:

```
LISTBC NOMAIL,NONOTICES
```

or like this:

```
LISTBC NOMAIL    NOTICES
```

A list of items may be enclosed in parentheses and separated by blanks or commas, for example:

```
LISTDS (MYDSA MYDSB,MYDSC)
```

Subcommands

The work done by some of the commands is divided into individual operations. Each operation is defined and requested by a subcommand. To request one of the individual operations, you must first enter the command. You can then enter a subcommand to specify the particular operation that you want performed, and you can continue entering subcommands until you enter the END subcommand.

The commands that have subcommands include EDIT, OUTPUT and TEST. Some program product commands (such as PLIC) have subcommands as well.

Syntax Notation Conventions

The notation used to define the command syntax and format in this publication is described in the following paragraphs.

1. The set of symbols listed below is used to define the format but you should never type them in the actual statement. The special uses of these symbols are explained in paragraphs 4-8.

   hyphen -

   underscore __

   braces {}

   brackets []

   ellipsis ...

2. You should type uppercase letters, numbers, and the set of symbols listed below in an actual command exactly as shown in the statement command syntax.

   apostrophe '

   asterisk *

   comma ,

   equal sign =

   parentheses ()

   period .

3. Lowercase letters and symbols that appear in the command syntax represent variables for which you should substitute specific information in the actual command.

   Example: If *"name"* appears in the command syntax, you should substitute a specific value (for example, ALPHA) for the variable when you enter the command.

4. Hyphens join lowercase words and symbols to form a single variable.

   Example: If member-name appears in the command syntax, you should substitute a specific value (for example, BETA) for the variable in the actual command.

5. An underscore indicates a default option. If you select an underscored alternative, you need not specify it when you enter the command because the system provides it for you by default.

   Example: The representation

   A
   $\underline{B}$
   C

   indicates that you are to select either A or B or C; however, if you select B, you need not specify it because it is the default option.

6. Braces group related items, such as alternatives.

   Example: The representation

   ```
             ⎧A⎫
   ALPHA=( ⎨B⎬ ,D)
             ⎩C⎭
   ```

   indicates that you must choose one of the items enclosed within the braces. If you select A, the result is ALPHA=(A,D).

7. Brackets also group related items; however, everything within the brackets is optional, so that you may omit all bracketed items.

   Example: The representation

   ```
             ⎡A⎤
   ALPHA=( ⎢B⎥ ,D)
             ⎣C⎦
   ```

   indicates that you may choose one of the items enclosed within the brackets or that you may omit all of the items within the brackets. If you select only D, you may specify ALPHA=(,D).

8. An ellipsis indicates that the preceding item or group of items can be entered more than once.

   Example: The representation

   ```
   ALPHA[,BETA...]
   ```

   indicates that ALPHA can appear alone or can be followed by ,BETA any number of times in succession.

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

Line Continuation

When it is necessary to continue to the next line, use a plus sign or a minus sign as the last character of the line being worked on. A plus sign will cause leading delimiters to be removed from the continued line.

Delimiters affected by the plus sign are blanks, tabs, commas, and the /*. The plus sign will left-justify a continued line that begins with any of these delimiters.

The minus sign continues the line without regard to the delimiters present on the continued line.

Example 1

Continuation using a minus sign:

```
list (data-set-list)    /* this is a list of my -
                        active data sets */
```

Example 2

Same example using a plus sign:

```
list (data-set-list) /* this is a list of my +
active data sets */
```

Note:

All of the leading blanks in the second example were deleted and the data was left justified.

Comments

You may insert comments into your TSO command sequences at any place where a blank would ordinarily appear. Simply start the comments with a slash-asterisk sequence, like this:

```
    /* This is a comment. TSO ignores it as a command. */
```

The trailing asterisk-slash sequence is optional unless you follow the comment with a command on the same line.

Comments may continue from line to line without limit. To continue a comment to the next line, enter either a plus sign or a minus sign (hyphen) as a continuation character, and immediately press the ENTER key, like this:

```
change 20 ?br?bt /* This comment begins on a command+
line and continues on consecutive following lines-
beginning in column one or following columns */
```

TSO considers the valid comments field in a line to be in columns 1-72 for fixed-length record formats, or to the end of the line for variable-length record formats.

Using System-Provided Aids

Several aids are available for your use at the terminal:

- The attention interruption stops processing so that you can enter a command.
- The conversational messages guide you at the terminal.
- The HELP command provides information about the commands.

The Attention Interruption

The attention interruption allows you to interrupt processing at any time so that you can enter a command or subcommand. If you are executing a program and the program gets in a loop, for instance, you can use the attention interruption to halt execution. As another example, when you are having data displayed at your terminal and the data that you need has been displayed, you may use the attention interruption to stop the displaying operation instead of waiting until the entire data set has been displayed.

If, after causing an attention interruption, you want to continue with the interrupted operation, you can do so by pressing the ENTER key before typing anything else; however, input data that was being typed or output data that was being displayed at the time of the attention interruption may be lost. You can also request an attention interruption while at the command level, enter the TIME command, and then resume the interrupted operation by pressing the ENTER key.

Note: One output record from the interrupted programs may be displayed at the terminal after you enter your next command. This is normal for some programs.

You can use the TERMINAL command to specify particular operating conditions that the system is to interpret as a request for an attention interruption. More specifically, you can specify a sequence of characters that the system is to interpret as a request for an attention interruption. In addition, you can request the system to pause after a certain number of seconds of processing time has elapsed or after a certain number of lines of output have been displayed at your terminal. When the system pauses, you can enter the sequence of characters that you define as a request for an attention interruption.

Messages

The conversational messages that TSO issues to your terminal guide you through your TSO session. There are four categories of messages: mode messages, prompting messages, informational messages, and broadcast messages.

Mode Messages

A mode message tells you when the system is ready to accept a new command or subcommand. When the system is ready to accept a new command it displays:

```
READY
```

When you enter a command that has subcommands and the system is ready to accept that command's subcommands, it displays the name of the command, for example:

```
EDIT
```

You can then enter the subcommands you want to use. The EDIT message also appears after each EDIT subcommand has been processed. If the system has to display any output or other messages as a result of the previous command or EDIT subcommand, it does so before displaying 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 displays the READY messages in succession after the commands. If you enter the following commands without waiting for the intervening mode messages, your display will be:

```
READY
attrib...
allocate...
edit...
READY
READY
EDIT
```

Unless you are sure that there are no mistakes in your input, you should wait for a READY message before entering a new command. When the system detects a mistake, it 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.

Note: Some terminals lock the keyboard after you enter a command, and therefore you cannot enter commands without waiting for the intervening READY message. Terminals that do not ordinarily lock the keyboard may occasionally do so, for example, when all buffers allocated to the terminal are used.

Prompting Messages

A prompting message tells you that required information is missing, or that you specified the information incorrectly, and asks you to supply or correct that information. For example, partitioned-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 display will look like this:

```
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 ENTER key to enter it. If the data set name is ALPHA.DATA, you would complete the prompting message as follows:

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

If you wish, you will receive prompting messages when appropriate; however, you may also use the PROFILE command to suppress prompting.

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

To request an additional level of message:

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

```
level 1  ENTER DATA SET NAME+
level 2  ENTER THE NAME OF A PARTITIONED DATA SET AND
         MEMBER THAT CONTAINS THE PROGRAM TO BE
         EXECUTED.
```

If you enter a question mark, and there are no messages to provide further detail, you receive the following message:

```
NO INFORMATION AVAILABLE
```

Canceling a Prompting Sequence

To stop any prompting sequence, enter an attention interruption. Ordinarily, the entry of the correct information to satisfy the prompting messages terminates the prompting sequence. However, you may find yourself occasionally unable to respond to the system's satisfaction, and you would prefer to get out of the prompting sequence entirely, either to try something another way, or to re-enter the correct information. Entering the attention interruption stops the prompting sequence and puts TSO back into a command or subcommand reception mode.

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 an informational message ends with a plus sign (+) you can request an additional message by entering a question mark (?) after READY. Informational messages have only one second level message, 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:

```
DEPARTMENT 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.

The HELP Command

The HELP command is available to a terminal user to provide all the information necessary to use any TSO command. The requested information is displayed at the user's terminal.

Explanations of Commands

To receive a list and a description of all the TSO commands in the HELP data set, enter the HELP command as follows:

```
help
```

Information about installation-written commands may also be in the HELP data set, if your installation's system programmer chooses to put it there.

You can also get all the information available on a specific command by entering that command name as an operand on the HELP command, as follows:

```
help call
```

If you want to know just the function of a particular command (DELETE, for instance), enter the HELP command as follows:

```
help delete function
```

If you want to know just the syntax of a particular command (TEST, for instance), enter the HELP command as follows:

```
help test syntax
```

If you want to know both the function and the operands of a particular command (EXEC, for instance), enter the HELP command as follows:

```
help exec function operands
```

Syntax Interpretation of HELP Information

The syntax notation used to present HELP information at your terminal is different from the syntax notation used in this publication. Since the HELP information resides in the SYS1.HELP data set, it is restricted to characters that can be represented at your terminal. If you want to use the HELP command, you should become familiar with the syntax interpretation by entering the HELP command as follows:

```
help help
FUNCTION -

    THE HELP COMMAND PROVIDES FUNCTION, SYNTAX, AND
    OPERAND INFORMATION ON COMMANDS. MESSAGE IDENTIFIER
    INFORMATION IS SUPPLIED WHEN AVAILABLE

SYNTAX -
    HELP'COMMAND NAME' FUNCTION SYNTAX
       OPERANDS('KEYWORD LIST') ALL/
       MSGID ('MESSAGE IDENTIFIER LIST')

    REQUIRED -   NONE
    DEFAULTS -   ALL IF FUNCTION, SYNTAX, OPERANDS, OR
                 MSGID NOT SPECIFIED.
    ALIAS    -   H
    NOTE     -   MSGID CANNOT BE SPECIFIED WITH THE
                 FUNCTION, SYNTAX, OPERANDS, OR ALL
                 KEYWORDS.
    NOTE     -   'KEYWORD LIST' IS OPTIONAL WHEN OPERANDS
                 IS USED.
    NOTE     -   IF HELP IS ENTERED WITHOUT ANY OPERANDS A
                 LIST OF AVAILABLE COMMANDS WITH A SHORT
                 DESCRIPTION OF EACH WILL BE DISPLAYED.
Syntax Interpretation -
1. USER SUPPLIED VALUES ARE IN APOSTROPHES. TWO SETS OF
    APOSTROPHES MEANS THE VALUE SHOULD BE SUPPLIED WITHIN
    A SET OF APOSTROPHES.
2. WORDS WITHOUT APOSTROPHES ARE TO BE ENTERED AS SHOWN.
3. COMMAS, PERIODS, PARENTHESES, AND ASTERISKS ARE TO BE
    ENTERED AS SHOWN.
4. EXCLUSIVE CHOICES ARE INDICATED BY SLASH (/).
5. MUTUALLY EXCLUSIVE FORMATS ARE SEPARATED BY 'OR'.
.
.
.
```

Explanations of Subcommands

You can also receive a list of all of a command's applicable subcommands. To get a list of the subcommands of EDIT, for example, you must first get the system to issue the EDIT mode message. The following simulated display shows how to enter the EDIT command to specify a an existing data set, and to receive the EDIT message:

```
    .....
    READY
    edit cmdlang old asm
    EDIT
    help
```

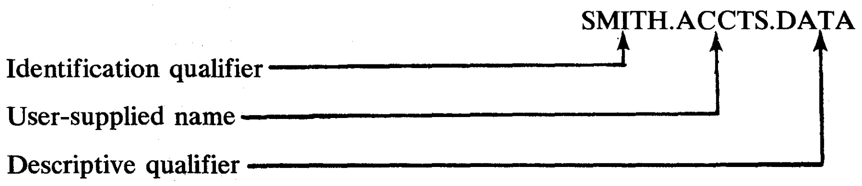
HELP entered without any operands produces a list of subcommands of EDIT.

Using Data Set Naming Conventions

The name you give a data set should follow the TSO naming conventions. A TSO data set name normally has three fields:

- Identification qualifier (used to make the data set name unique)
- User-supplied name (optional for a partitioned data set)
- Descriptive qualifier, which has meaning to the TSO commands

When you create a data set you only need to specify the user-supplied name. The system supplies values for the other two fields. The fields must be separated by periods. Each field consists of 1-8 alphameric characters and begins with an alphabetic or national ($, @, and #) character. The total length of the name, including periods, must not exceed 44 characters. A typical data set name is:



In this example, the identification qualifier is SMITH. The identification qualifier is either the user identification you specified with the LOGON command, or a qualifier you assign by using the PROFILE command.

The user-supplied name in this example is ACCTS. The user-supplied name can be either a single field or several fields separated by periods.

The descriptive qualifier in this data set name is DATA. The possible descriptive qualifiers are listed in Figure 1. The system determines what the descriptive qualifier should be from the command being processed and the contents of the data set. Figure 2 lists the default descriptive qualifiers that the system supplies when various commands are issued. The system may also determine the descriptive qualifier from the data set type operand entered on the EDIT command. See the EDIT command in the **TSO Command Language Reference** for a description of the data set type operand.

Descriptive Qualifier	Data Set Contents
ASM	Assembler input
CLIST	TSO commands and subcommands
CNTL	*JCL and SYSIN for SUBMIT command
COBOL	American National Standard COBOL statements
DATA	Uppercase text
FORT	FORTRAN IV (G1 or H) statements and free- or fixed-format Code and Go FORTRAN statements
LINKLIST	Output listing from linkage editor
LIST	Listings
LOAD	Load module
LOADLIST	Output listing from loader
OBJ	Object module
OUTLIST	*Output listing from OUTPUT command
PLI	PL/I Checkout or PL/I Optimizing compiler statements
TESTLIST	Output listing from TEST command
TEXT	Uppercase and lowercase text
VSBASIC	VSBASIC statements

*Refer to Appendix A in **OS/VS2 TSO Command Language Reference.**

Figure 1. Descriptive Qualifiers

	DESCRIPTIVE QUALIFIERS		
Command	**Input**	**Output**	**Listing**
ASM	ASM	OBJ	LIST
CALL	LOAD	—	—
COBOL	COBOL	OBJ	LIST
CONVERT	FORT	FORT	—
EXEC	CLIST	—	—
FORMAT	TEXT	—	LIST
FORT	FORT	OBJ	LIST
LINK	OBJ	LOAD	LINKLIST
	LOAD	—	—
LOADGO	OBJ	—	LOADLIST
	LOAD	—	—
OUTPUT	—	—	OUTLIST
RUN	ASM	—	—
	FORT	—	—
	COBOL	—	—
SUBMIT	CNTL	—	—
TEST	OBJ	—	TESTLIST
	LOAD	—	—

Figure 2. Descriptive Qualifiers Supplied by Default

Exceptions to Data Set Naming Conventions

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 necessary when you have to use a data set with an identification qualifier other than your own user identification. This procedure also reduces response time because it causes the system to perform fewer functions.

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 data set names that are enclosed in apostrophes.

You can refer to an existing data set by its user-supplied name and descriptive qualifier. If your data set is named:

```
SMITH.PART1.DATA
```

You may want to specify the data set name as:

```
part1.data
```

or you may specify the data set type if you are using the EDIT command:

```
edit part1 old data
```

Specifying Data Set Passwords

When referencing password-protected data sets, you must specify the password as part of the data set name or you will be prompted for it. Separate the password from the data set name by a slash (/) and optionally, by one or more standard delimiters (tab, blank, or comma).

Partitioned Data Sets

You can also create and edit partitioned data sets. A partitioned data set consists of one or more data sets called members. You can create and edit each member separately, giving each one a unique name. Enclose a member name in parentheses and append it 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 only need to use the user-supplied name and member name to refer to the member. The system appends the identification and descriptive qualifiers and moves the member name to the end to form the fully qualified name. Thus, to refer to member MEM1 you can specify:

```
part1(mem1)
```

or you might specify

```
part1.data(mem1)
```

In the second example, the system appends only the identification qualifier.

The following example uses 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 (the OLD operand is the default).

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

Data Set Types for the EDIT Command

After you specify the data set name and the NEW or OLD operand, specify the data set type. The data set type is an operand that describes the contents of the data set. The type operand is one of the sources from which the system can obtain the descriptive qualifier. (If the descriptive qualifier is a valid data set type, you may specify the descriptive qualifier as part of the data set name, rather than giving data set type: specify EDIT MYDS.DATA instead of EDIT MYDS DATA.) The valid types are:

ASM
CLIST
CNTL
COBOL
DATA
FORTGI
FORTH
GOFORT
PLI
PLIF
TEXT
VSBASIC

Note: Any user data set types, specified at system generation time, are also valid data set types.

If the system cannot determine the data set type from other sources, it prompts you for it.

Section II: Starting and Ending a Terminal Session

This section tells you how to identify yourself to TSO and describes a sample introductory terminal session that you can perform to get valuable hands-on experience and easy familiarity with using a terminal. Following the sample session are discussions of the things you did during the exercise, plus additional discussions of most commonly encountered terminal situations. These discussions include descriptions of how you can use TSO commands to:

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

Getting a TSO User Identification

The first step in becoming a TSO user is to make yourself recognizable to the system by getting a TSO user identification (userid). You will have to ask your supervisor to provide you with the exact local procedure for doing this.

Running a Sample TSO Terminal Session

After getting your userid, you are ready to use an available terminal to familiarize yourself with the mechanics of a terminal session.

Remember that this description assumes that you are using an IBM 3270 Display Station. Using a different terminal may disclose some minor operational differences, but basically, this example will work on any TSO terminal. You may wish to consult the appendixes to pinpoint the differences and clarify the procedure where necessary.

It will save you time while at the terminal to have previewed the example and determined what you should expect to do during the session, but you can run this terminal session without having done so first. It would also be useful to read the more detailed discussions about conducting a terminal session that follow this example to broaden your understanding of the points this example illustrates.

Contacting TSO

To start a TSO terminal session:

1. If power is off:

   - Pull out the POWER control knob on the left side panel of the display screen. The terminal should now be in contact with the system. If not, special procedures may be necessary. Contact your system programmer.

   - Turn the POWER control knob clockwise to brighten the image or counterclockwise to darken the image.

If power is on:

- Press the CLEAR key and then the RESET key. The cursor moves to the upper left corner of the screen and the INPUT INHIBITED light goes off.

2. If your terminal is an SDLC (synchronous data link control) 3270, before you can log on to TSO a SNA (systems network architecture) session must first be established between the terminal and TCAM. There are several ways for automatically establishing this session; your installation should define the method to be used. If your installation does not automatically establish the SNA session with TCAM, you must enter the installation-defined character string required and press the TEST REQ key. For BSC and local 3270s, perform step 4.
3. If your installation does not start the TSO session in step 2, then step 4 must be performed. Your installation should provide this information.
4. Enter the LOGON command to identify yourself to TSO. Type the word LOGON, a space, and your user identification (userid).
5. Transmit the LOGON command to TSO by pressing the ENTER key; the INPUT INHIBITED light comes on. Wait for TSO to reply to your LOGON command. TSO may display a preliminary message:

```
LOGON PROCEEDING
```

but when you are logged on, TSO displays the message:

```
READY
```

and turns the INPUT INHIBITED light off. You can now enter any command.

As part of the LOGON command (see the LOGON command in **OS/VS2 TSO Command Language Reference**), installations may also require a password, an account number, and a cataloged procedure name.

First type the word LOGON, a space, and your userid. If a password is required, type it after the userid, separating the two with a slash (/). If required, an account number and a cataloged procedure name follow, separated with spaces or commas.

Example

The userid is MYNUM. The cataloged procedure name is TRYOUT1. Type:

```
logon mynum proc(tryout1)
```

Press the ENTER key; the INPUT INHIBITED light comes on. Wait for TSO to display the message READY and to turn the INPUT INHIBITED light off. You may now enter any command.

Example

The userid is MYSEVEN. The password is APASS. The account number is AN38. Type:

```
logon myseven/apass acct (an38)
```

Press the ENTER key; the INPUT INHIBITED light comes on. Wait for TSO to display the message READY and to turn the INPUT INHIBITED light off. You may now enter any command.

Entering Data

Now you are going to use the TSO EDIT command to open a data set and enter data into it. $\boxed{E}$ means press the ENTER key.

Type: `edit roster data new` $\boxed{E}$

System:
```
INPUT
00010
```

Your EDIT command just told TSO that you want to open a new, data-type data set named "roster." TSO's response is an implicit recognition that it has opened the data set for you, and an explicit statement that it has placed the terminal in "input mode" so that you may begin entering data into "roster." The number 00010 is a "line prompt" to show you where to begin typing in your data. Now start typing this data where TSO left the cursor after the line prompt.

Type:

```
cosman      ba  csr lisbon     yes [E]
dockswell   br  nrm kingston    no [E]
reed        dj  bsr clayton    yes [E]
stone       jl  dsh rochester  yes [E]
```

At this point, your listing should look like this:

```
00010   cosman       ba  csr      lisbon        yes
00020   dockswell    br  nrm      kingston      no
00030   reed         dj  bsr      clayton       yes
00040   stone        jl  dsh      rochester     yes
00050
```

TSO prompted you for line 50 as a matter of course, because the system cannot anticipate when you are going to terminate your data entry. Press the ENTER key again to signal TSO that you are through entering data.

Type: $\boxed{E}$

System: `EDIT`

TSO has acknowledged the extra ENTER as a request for edit mode, and now you can change your data. Suppose you entered Dockswell's initials incorrectly. With the CHANGE subcommand, you can correct your error.

Type: `change 20 /br/bt` $\boxed{E}$

System: `(no response)`

Type: `list 20` $\boxed{E}$

System: `00020  DOCKSWELL  BT  NRM  KINGSTON  NO`

At this point, you have directed TSO to change Dockswell's initials in line 20 from "BR" to "BT", and then you had TSO display line 20 at the terminal.

Listing and Saving the Data Set

To display your entire data set, simply type the EDIT subcommand LIST (you are still in edit mode).

Type: `list` $\boxed{E}$

System:

```
00010        COSMAN       BA   CSR     LISBON       YES
00020        DOCKSWELL    BT   NRM     KINGSTON     NO
00030        REED         DJ   BSR     CLAYTON      YES
00040        STONE        JL   DSH     ROCHESTER    YES
END OF DATA
```

To save the data set in permanent storage, enter the EDIT SAVE subcommand.

Type: `save` $\boxed{E}$

System: `EDIT`

Ending the EDIT Function

At this point, suppose that you wish to terminate work on the data set named "roster" and move on to some other work. You can end the edit function for "roster" by entering the EDIT END subcommand.

Type: `end` $\boxed{E}$

System: `READY`

Because you previously saved "roster" in permanent storage, TSO tells you it is ready to accept a new command. However, if you had failed to save the data set before entering the END subcommand, TSO would have given you another chance to save the data set (in case you had forgotten to) by issuing the message:

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

This second chance that TSO gives you to save your data set permits you to do away with it if you wish, simply by entering END the second time.

You can also use SAVE as an operand of the END subcommand in order to save your data set and end the EDIT function in a single command line by entering:

`end save` $\boxed{E}$

In this case, TSO saves your data set before ending the EDIT session.

Modifying an Existing Data Set

At this point you can recall your data set from permanent storage for additional modification by means of the EDIT command. Suppose, however, you had several data sets in permanent storage and you wanted to refresh your memory about the names by which you stored them. The LISTCAT command provides you with this capability. When you invoke it, the command causes the (partially qualified) names of each of your data sets in permanent storage (which TSO cataloged for you when you saved them) to be listed.

Listing the Catalog

Now use LISTCAT to list the name of your one stored data set.

Type: `listcat` $\boxed{E}$

System:
```
IN-CATALOG: (catalog name)
ROSTER.DATA
READY
```

By displaying the READY message, TSO is prompting you for another command.

Recalling a Stored Data Set

Recalling the data set with the EDIT command is very similar to creating it in the first place, but now the data set is "old" rather than "new." Recall it like this:

Type: `edit roster data old` $\boxed{E}$

System: `EDIT`

With its response, TSO has told you that it has obtained a copy of your data set and it is now available for you to edit.

Deleting a Line of Data

You can delete a line of data simply by typing its line number, but TSO does not acknowledge the deletion at the terminal.

Type: `20` $\boxed{E}$

System: `(no response)`

Type: `list 20` $\boxed{E}$

System: `LINE NUMBER 20 NOT FOUND`

When you tried to list line 20 to verify that the deletion took place, TSO could not find the line because it had already deleted it.

Inserting Lines of Data

You can insert new lines of data by entering the line number and following it with the data you wish to insert. Now make a new roster entry at Dockswell's old line position.

Type: `20   henry   ra   aoh   albany   no` $\boxed{E}$

System: `(no response)`

Type: `list 20` $\boxed{E}$

System: `00020  HENRY  RA  AOH  ALBANY  NO`

In this example, TSO made the insertion and you listed the new line.

You can also insert new lines of data between existing lines in the data set by assigning line numbers that fall between the TSO-assigned numbers (which are in increments of ten, by default).

Type: `35   reed   py   jsr  clayton   yes` $\boxed{E}$

System: `(no response)`

Type: `list 30 40` $\boxed{E}$

System:

```
00030  REED   DJ    BSR   CLAYTON    YES
00035  REED   PY    JSR   CLAYTON    YES
00040  STONE  JL    DSH   ROCHESTER  YES
```

This time, you verified the line insertion by directing TSO to list a range of lines that included your insertion.

Replacing a Line of Data

You can replace an entire line of data without first having deleted it simply by entering the line number and following it with the replacement information. (This function is just like inserting a new line of data, except that you are specifying a line number where there is already some data, all of which you will lose.)

Type: `40   smith   ra  dsh  montrose   no` $\boxed{E}$

Type: `list` $\boxed{E}$

System:

```
00010   COSMAN   BA   CSR   LISBON    YES
00020   HENRY    BA   AOH   ALBANY    NO
00030   REED     DJ   BSR   CLAYTON   YES
00035   REED     PY   JSR   CLAYTON   YES
00040   SMITH    RA   DSH   MONTROSE  NO
END OF DATA
```

By listing your entire data set, you can verify that all the editing that took place since you recalled a copy from permanent storage has produced all the desired results. Note that STONE is no longer in the roster, and the other changes have also been made.

Deleting Modified Data

The simplest way to delete this session's modifications is to end the EDIT session without explicitly saving them.

Type: `end` $\boxed{E}$

System:

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

Type: `end` $\boxed{E}$

System: `READY`

Type: `listcat` $\boxed{E}$

```
System: IN-CATALOG: catalog name
        ROSTER.DATA
        READY

Type: edit roster.data old [E]

System: EDIT

Type: list [E]

System: 00010  COSMAN    BA  CSR  LISBON    YES
        00020  DOCKSWELL BT  NRM  KINGSTON  NO
        00030  REED      DJ  BSR  CLAYTON   YES
        00040  STONE     JL  DSH  ROCHESTER YES
        END OF DATA
```

The data set you just listed is the one you created initially, before you recalled it for changes. Note that TSO did not save your changes when you ended the EDIT function without entering SAVE.

Now end the EDIT function again.

```
Type: end [E]

System: READY
```

Because you used the EDIT function this time only to list the data set (you made no changes that could be saved), TSO accepted your initial END entry without prompting you for a SAVE.

Deleting the Data Set and Logging Off

You can explicitly delete one or more data sets from permanent storage by using the DELETE command. DELETE removes the indicated catalog entries and frees the permanent storage. It is a good idea to review your catalog periodically and delete data sets you no longer need.

Now review your catalog entries, delete your data set, and log off.

```
Type: listcat [E]

System: IN-CATALOG: catalog name
        ROSTER.DATA
        READY

Type: delete roster.data [E]

System: READY

Type: listcat [E]

System: ENTRY catalog name. NOT FOUND

Type: logoff [E]

System: CFT086 LOGGED OFF TSO AT 09:24:10 ON JUNE 26,
1974+
```

This command sequence has just reviewed your catalog, deleted its entry, verified the deletion, and terminated your terminal session.

Now turn off your terminal to leave it available for the next user. This concludes your sample terminal session. The remainder of this section reiterates in greater detail the techniques you just used and discusses many that you did not.

Identifying Yourself to the System

After you activate the terminal you must use the LOGON command to identify yourself to the system. You supply, as operands of LOGON, the user attributes assigned to you by your installation. Your user attributes will consist of, at the minimum, a userid. The others listed below are optional unless your installation makes them necessary, in which case, the system prompts you for them.

- User identification (required) -- the name or code by which the system knows you
- Password (required if your installation assigns you one) -- a further identification used for additional security protection
- Account number (optional) -- the account to which your terminal session is charged
- Procedure name (optional) -- the name of a series of statements that defines your job to the system
- Performance group (optional) -- the performance group you wish to use during the session

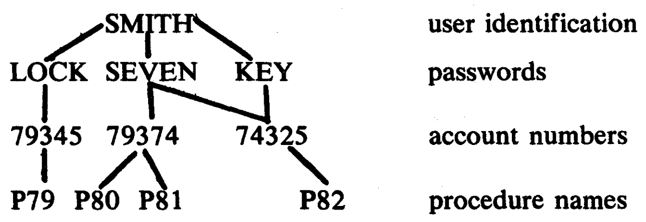
Your user attributes are in the system together with the attributes of all other terminal users. When you log on, the system compares the attributes you specify in the LOGON command to the attributes recorded in your user profile, to determine if you are an authorized user of the system.

User Attributes

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

SMITH	user identification
LOCK	password
79345	account number
P79	procedure name

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 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.

Logging On

The LOGON command tells the system your user identification, password, account number, procedure name, performance group, and whether you want the reconnect option. If you want to use procedure P81, for example, 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. Account 79345 and procedure P79 are the only account and procedure associated with password LOCK. Therefore, when you log on you only need to 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 the password KEY, you do not have to specify the account number and procedure name because there is only one account number and one procedure name associated with KEY.

Print-Inhibiting Your Password

Some terminals provide the capability to inhibit the display of data that you are entering on the keyboard. This print-inhibit feature suppresses a display of your password, thereby decreasing its exposure to people unauthorized to know it.

To return to the previous example involving Smith for a moment, initially Smith could enter:

```
logon smith
```

Note that he enters only his userid -- no slash, no password.

The system responds with a prompt for the password, which will look like this:

```
    ENTER PASSWORD FOR SMITH
```

The system puts the terminal in print-inhibit mode. Then Smith types one of his authorized passwords. Special care is necessary because the print-inhibit feature is *preventing the characters from being displayed on the screen,* thereby preventing Smith (or anyone else) from checking on the password entry. Note that, after entering the correct password, the user may be prompted for other information such as account number and procedure name.

If Smith makes a mistake in entering his password, the system prompts him to try again before logging him off as a potentially invalid user.

Defining Operational Characteristics

Operational characteristics include 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 after a specified number of seconds
- The length of the line that can be displayed at your terminal

Some of the characteristics a user profile identifies are:

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

Refer to the PROFILE and TERMINAL commands in **OS/VS2 TSO Command Language Reference** for additional information about defining terminal and user profile characteristics.

Terminal Characteristics

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 that terminal. Assume that 50 is the default for the number of lines of continuous output that are displayed before you receive an automatic interruption. You can use the TERMINAL command to request that 100 lines be displayed before you receive an interruption. When you log on for your next session at that terminal, 50 lines will again be the default, provided there has been a logoff prior to the logon. The terminal characteristics remain the same for a re-logon terminal session and assume the default values with a logoff.

Your User Profile

The system has a user profile for you and 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 a permanent part of your profile. Assume that the line-deletion character in your profile is a percent (%) sign. You could use the PROFILE command to change it to a number (#) sign for the current session and subsequent sessions. 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 and mail. Notices are messages that the system operator sends to all users. Mail consists of messages sent by the operator or another user directly to you. You can send 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 (notices) and those intended only for you (mail) are displayed at your terminal. You can use the following operands of the LOGON command to prevent display of either type of message at your terminal:

- NONOTICES suppresses display of broadcast messages intended for all terminal users.
- NOMAIL suppresses display 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. If you enter the following commands:

```
READY
profile nointercom
```

you request that all available broadcast messages (notices and mail) be displayed when you log on, but that all mail sent to you after logon 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 (notices and mail).

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 logon 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 messages available at logon 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 is still 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, except when you are in the TEST mode.

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,smith)
```

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

When you send a message to another user, he will receive it immediately if 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. 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 also send a message to a user for his later perusal, even though he is currently logged on. That user in turn may read your stored message at his convenience by using the LISTBC command. To store your message, enter:

```
send 'this is a message' user(smith) save
```

Furthermore, you can ensure that a logged-on user receives an important message, even though his terminal is busy, by entering:

```
send 'this is a message' user(smith) wait
```

This message entry causes the system to wait until Smith's terminal is no longer busy, and can accept the message. It also causes your terminal to wait until Smith receives the message.

You can send a message to only one operator at a time by identifying him with a number, for example:

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

If there is only one operator at your installation, you can omit the operand by entering:

```
send 'important message'
```

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

Displaying Session Time Used

Use the TIME command to obtain the following information:

- Cumulative CPU time (from logon)
- Cumulative session time (from logon)
- Service units used
- Local time of day
- Today's date

If you wish to enter the TIME command while executing a program or command, you must first cause an attention interruption. The TIME command has no effect upon the executing program.

If a TSO command has been executing longer than expected, you can interrupt it to check its CPU and execution time. Then, depending on your analysis of the times returned, you can either resume processing from the point of interruption, or you can cancel the processing of that command. The following example shows how a LOADGO command was interrupted, a TIME command was entered successfully, and a null line was entered to resume the processing of the LOADGO command.

```
READY

loadgo pehtest
|
READY
time
(Your time information is printed here)

READY
(Press the ENTER key)
VALID TYPES FOR DATA SET PEHTEST ARE LOAD AND OBJ
ENTER TYPE-
obj
READY   (indicates that LOADGO has completed successfully)
```

Note: If the user had decided to cancel the processing of LOADGO, he would only have had to issue another command after the third READY to cancel LOADGO.

Ending Your Terminal Session

You can end your terminal session in two ways:

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

The LOGOFF command logically disconnects your terminal from the system. If LOGOFF HOLD is specified, the terminal remains physically connected and you can enter a new LOGON command; however, terminal characteristics established by a TERMINAL command during the previous session are no longer in effect. A typical logoff follows:

```
READY
logoff
D58PEH LOGGED OFF TSO AT 15:24:47 on MAY 22, 1973+
```

The LOGON command terminates your current session and starts a new one at the same time. A typical logon follows:

```
READY
logon d58peh/d58paswd 10781525
D58PEH LOGGED OFF TSO AT 10:14:06 ON MAY 23, 1973+
D58PEH LOGON IN PROGRESS AT 10:14:40 ON MAY 23, 1973
READY
```

Note: In the case of a re-logon as shown above, the terminal characteristics of the old session are carried over into the new session.

Section III: Entering and Manipulating Data

The processing of data is an important part of almost all system applications. Therefore, you should learn how to enter data into the system and how to modify, store, and retrieve data after it has been entered. 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.

Using the EDIT Command

The EDIT command, which is used to enter 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.

Entering Data in Input Mode

In input mode, you can type a line of data and then enter it into the data set by pressing the ENTER 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. The command or subcommand is taken as data and is not executed.

You can also have the system assign a line number to each line as it is entered. Line numbers make later 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 display 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 ENTER key to enter a null line.)

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

```
EDIT
```

Entering Subcommands in Edit Mode

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 internal 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 in accordance with the subcommand you are using.

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 required. You may also use an asterisk in place of a line number in a subcommand to indicate that you wish to use the current line pointer.

Switching Modes

After you finish editing the data, you can switch to input mode in two ways:

- Entering the INPUT or INSERT subcommand.
- Entering a null line. (Press the ENTER key to enter a null line.)

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

```
INPUT
```

You can terminate the EDIT command at any time by switching to edit mode (if not already in edit mode) and entering the SAVE parameter on the END subcommand. The system then displays 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 ENTER key. You can then enter other lines after the blank line. If you fail to enter a blank and press only the ENTER key, you are entering a null line which causes EDIT to switch modes from input mode to edit mode.

Functions of EDIT Subcommands

The remainder of this chapter describes how you can use the subcommands of EDIT to:

- Identify data sets.
- 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 new or updated data set.
- Move or copy data within a data set.
- Allocate a data set.

- Submit a data set for batch execution.
- Send a message.
- End the EDIT functions.

Functions of Other Commands

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

- Rename a data set.
- Delete a data set.
- Allocate a data set.
- Free an allocated data set.
- List information about your data sets.

Note: A data set may be allocated by using the ALLOCATE command or the ALLOCATE subcommand of EDIT.

Identifying Data Sets

Use the EDIT command 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. For example, the NEW operand in the following EDIT command specifies that you are going to create a new data set named ACCTS.DATA; the system enters input mode.

```
READY
edit accts.data new
INPUT
00010
```

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

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

Creating a Data Set

You request the input mode when you enter one of the following:

- The NEW operand in the EDIT command
- The INPUT subcommand while in edit mode
- The INSERT subcommand with no operands, while in edit mode
- A null line if the system is in edit mode

The system sends you the following message:

```
INPUT
```

After this message, the system displays 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 ENTER key to enter it. The system then displays the second line number, which is 00020, and you may then enter your second line of input, and so on.

Note: A hyphen at the end of an input line indicates logical continuation of the line. In input mode, logical continuation is meaningful only if you are using the syntax checking facility. Whether or not you are syntax checking, however, the input processor will delete the hyphen from the end of the line except in a few special instances. The rules governing handling of a hyphen at the end of a line in input mode are detailed in **OS/VS2 TSO Command Language Reference.**

When you reach the end of the data you want to enter, press the ENTER key to enter a null line and the system switches to edit mode, as the following example illustrates:

```
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 preceding example, the line numbers have the standard increment of 10. If you prefer a different increment, you can use the INPUT subcommand to specify another increment. 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 one will make it easier to insert lines in your data set later on.

Placing Data into Columns

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 use the defaults provided by the system. (See the appendix to determine if your terminal supports tab settings.) If you have established logical tab settings for your data set, the system will arrange each item in its proper column whenever you press the TAB key. The maximum number of logical tab settings that can be defined is ten.

If you do not use the TABSET subcommand, the default tab settings used by the system vary with the data set type. The defaults are shown in Figure 3.

Descriptive Qualifier	Default Tab Setting Columns
ASM	10,16,31,72
CLIST	10,20,30,40,50,60
CNTL	10,20,30,40,50,60
COBOL	8,12,72
DATA	10,20,30,40,50,60
FORT	7,72
PLI	5,10,15,20,25,30,35,40,45,50
TEXT	5,10,15,20,30,40
VSBASIC	10,15,20,25,30,35,40,45,50,55
User Defined Qualifier	10,20,30,40,50,60

Figure 3. Default Tab Settings

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 on    (10 25 35)
             (null line)

INPUT
00010
```

If you prefer, you can define tab settings by entering a line containing t's in positions corresponding to desired tab settings. To establish tab settings in columns 10, 25, and 35 you can use the TABSET subcommand as follows:

```
tabset image
123456789tbbbbbbbbbbbbbbtaaaaaaaaat
```

You must fill the spaces between the t's with blanks or characters other than t. Do not use the TAB key when entering the IMAGE line, nor the backspace except as a character-delete character.

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 editing 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 depends on the last subcommand you entered. If you are editing an old data set, the current line pointer is positioned at the first line of the data set upon initial entry into edit mode. Figure 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:

```
list *
THIS IS THE LINE THE CURRENT LINE POINTER IS INDICATING
```

You can also have the system display the line at which the pointer is positioned every time the pointer changes as a result of one of the EDIT subcommands, and each time the CHANGE subcommand makes a modification to one or more lines. To do this, issue the VERIFY subcommand as follows:

```
    verify
```

The VERIFY function is in effect for an EDIT session until you enter it again with the OFF operand:

```
    verify off
```

EDIT Subcommands	Value of the Pointer at Completion of Subcommand
ALLOCATE	No change
BOTTOM	Last line (or line zero for empty data sets)
CHANGE	Last line changed
COPY	Last line copied
DELETE	Line preceding deleted line, if any, else zero
DOWN	The line n down from where you were at the start of the subcommand, or the bottom of the data set. (n is the value of the 'count' parameter.)
END	No change
EXEC	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 or replaced line, or line preceding the deleted line, if any, or zero.
LIST	Last line listed
MOVE	Last line moved
PROFILE	No change
RENUM	Same relative record
RUN	No change
SAVE	No change
SCAN	Last line referred to, if any
SEND	No change
SUBMIT	No change
TABSET	No change
TOP	Zero value
UNNUM	Same relative record
UP	The line n lines up from where you were at the start of the subcommand, or the top of the data set. (n is the value of the 'count' parameter.)
VERIFY	No change

Figure 4. How Edit Subcommands Affect the Current Line Pointer Position

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 up, relative to the pointer's current position in your data set. For example, to move the pointer so that it refers to a line located five lines before the location it currently refers to, enter:

```
up 5
```

The DOWN subcommand moves the pointer a specified number of lines down, relative to the pointer's current position in 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 is used to change the current line pointer to zero. After execution, the current line pointer precedes the first line of an unnumbered data set or the first line of a numbered data set that does not have a line number of zero. However, if the data set is numbered and contains a line number zero, the current line pointer points to the line numbered zero. 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 for which the system has to search. The special delimiter can be any character other than a number, apostrophe, semicolon, blank, tab, comma, parenthesis, asterisk, a slash followed by an asterisk, or one of the characters in the sequence you want to find. The special delimiter must be next to the first character of the sequence you want to find. Any blanks inserted between the special delimiter and the first character are considered to be part of the sequence of characters.

An alternate method for specifying the sequence of characters for FIND is quoted-string notation. With this method, the specified sequence must start and end with an apostrophe. If an apostrophe is one of the characters in the specified sequence, you must enter two apostrophes for the single apostrophe in the specified sequence. For example, to find the character sequence:

```
single 'quote'
```

using quoted-string notation, enter:

```
FIND 'single ''quote'''
```

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

```
find xplaced before entry x 7
```

or

```
find 'placed before entry ' 7
```

Notice that the same special delimiter or apostrophe 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:

```
top
find xplaced before entry
```

The following data set illustrates the examples of positioning the current line pointer. Although this data set has line numbers, they have no bearing on the examples, which depend rather on relative pointer positions.

```
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
```

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.
- Move or copy lines within a data set.
- Renumber lines of a data set.
- Remove line numbers from a data set.

The descriptions of these functions follow.

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
```

To delete the line indicated by the current line pointer, enter:

```
*
```

You can also use the DELETE subcommand to perform the same function:

```
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, to delete all the lines between and including lines 15 and 75, enter:

```
delete 15 75
```

To delete 12 lines starting with the line indicated by the current line pointer, enter:

```
delete * 12
```

To delete all the lines in your data set, use the TOP and DELETE subcommands in combination, specifying for DELETE a number of lines greater than the number of lines in your data set.

```
top
delete * 99999999
```

After the system deletes the lines you requested, the current line pointer points at the line before the first deleted line.

Inserting Data in a Data Set

To insert only one line of data in a line-numbered data set, you do not need a subcommand; indicate only the line number. In order to prevent overlaying an existing line of data, refer to an unused line number. (That is, the new line number should fall between two existing line numbers in the data set.) Thus, if you want to insert "RECORDED DAILY IN CENTRAL" as line 22, between lines 20 and 30, enter:

```
22 recorded daily in central
```

The characters you want to enter must be separated from the line number by a single blank or a comma. Any additional blanks or commas are considered to be part of the input data. You may optionally use the tab key to separate characters from the line number. (See the appendix to determine if your terminal supports tab settings.) In this case all blanks, including the first, resulting from the tab will be part of your input data.

The number of blanks resulting from the tab is determined by the logical tab setting. The logical tab setting results from the TABSET subcommand or the default tab setting.

To insert one line of data after the current line, use the INSERT subcommand with the insert-data operand:

```
list *
TAKE ME OUT
insert to the ballgame
```

The rules for separating inserted data from the subcommand name are the same as for separating data from line numbers.

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

Both subcommands have similar functions. However, the INPUT subcommand starts adding lines immediately after the last line of the data set or the last line affected by the last input activity. The INSERT subcommand inserts data following the location pointed to by the current line pointer.

Assume that you have the following data set:

```
A. CARSON       DEPT A72
T. DANIELS      DEPT 792
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.

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. 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 line for E. FARRELL and before the line for C. LEVI, your display would look like the following:

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

Note: After you enter the INSERT or the INPUT subcommand, EDIT switches to input mode.

If your data set has line numbers, you may use either the INPUT or INSERT 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. 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 increase this number by two in the following lines, your display would look like this:

```
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
```

Another way to insert three lines between lines 20 and 30 is to use the INSERT subcommand, as follows:

```
EDIT
top
down 2
insert
INPUT
00021        2795    $0.79
00022        3241    $2.81
00023        4152    $1.79
00024        (null line)
EDIT
```

Note: INSERT automatically increases the line numbers by one. The updated data set would look like this:

```
00010         1932    $1.50
00020         2579    $1.39
00021         2795    $0.79
00022         3241    $2.81
00023         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 paragraph entitled "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 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 a single blank or a comma. The system considers any additional blanks or commas to be part of the input data. You may optionally use the tab key to separate characters from the line number or asterisk. (See the appendix to determine if your terminal supports tab settings.) In this case all blanks, including the first, resulting from the tab will be part of your input data. The number of blanks resulting from the tab is determined by the logical tab setting. The logical tab setting results from the TABSET subcommand or the default tab setting.

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. 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 2
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 display should look like this:

```
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 like this:

```
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. 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 display should look like this:

```
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 like this:

```
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 before 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, apostrophe, parenthesis, slash followed by an asterisk, 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 system considers the blanks as significant parts of the characters to be replaced. The special delimiter need not appear at the end of the replacement characters unless other parameters are to follow.

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 also have the system search for a sequence of characters in a range of lines rather than in only 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 also 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. You must use the special delimiter to terminate the replacement string before type "all":

```
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 display 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 ENTER key. Assume that you want to change the characters "TUESDAY" in the following line:

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

Your display will look as follows:

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

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

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

In either of the two examples above, the cursor would have stopped at the space where "TUESDAY" originally began, and would have remained there, awaiting your change to the line. You could, for example, type "wednesday." and then press the ENTER key. Your display (in the latter example) would then look like this:

```
change   * /tuesday
00015    PARTS DELIVERIES ARE MADE ON wednesday.
```

Another way to do the same thing is to request that the system display a specified number of 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 26 characters of the line "PARTS DELIVERIES ARE MADE ON TUESDAY" be displayed:

```
change 15 26
00015 PARTS DELIVERIES ARE MADE
```

You can have the system display the first several characters of a range of lines. This is particularly useful when you want to change a column in a table. 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 16
00010   ENROLLMENT DATES
00012   P. JONES    MAY 15
00014   A. SMITH    MAY 31
00016   J. DOE      JUNE 7
00018   B. GREEN    JUNE 9
```

For each line displayed, the cursor stops at column 17 and you can enter new data or press the ENTER key to delete the data that was there.

If you want to change the data in the last column and the current line pointer is at line 10, enter:

```
change  * 5 16
00010   ENROLLMENT DATES
00012   P. JONES    MAY 15
00014   A. SMITH    MAY 31
00016   J. DOE      JUNE 7
00018   B. GREEN    JUNE 9
```

You can insert a sequence of characters at the beginning of the line. For example, if line 15 of your data set is as follows:

```
00015 EMPLOYEE ABSENTEEISM
```

enter:

```
change 15 //weekly report of /
```

to obtain:

```
00015 WEEKLY REPORT OF EMPLOYEE ABSENTEEISM
```

You can also delete a sequence of characters using the CHANGE subcommand. To delete WEEKLY from line 15 above, enter:

```
change 15 /weekly//
```

to obtain:

```
00015 REPORT OF EMPLOYEE ABSENTEEISM
```

Quoted-String Notation

In the previous examples of the CHANGE subcommand of EDIT, special-delimiter notation specified the character sequences. You may, however, use an alternate form of notation, the quoted-string notation. General rules for quoted-string notation are:

- Begin and end each sequence with an apostrophe. (The system will ignore the apostrophes in its operations on your character sequence.)
- Separate character sequences with a blank.
- Specify two apostrophes in place of one whenever you wish to include an apostrophe within a character sequence.

For example, to replace WEEKLY with DAILY in the current line, you can use the special-delimiter notation:

```
change * /weekly/daily/
```

or the quoted-string notation:

```
change * 'weekly' 'daily'
```

To delete DAILY from the current line, you can use:

```
change * 'daily' ''
```

instead of:

```
change * /daily//
```

To insert WEEKLY at the beginning of line 15, you can use:

```
change 15 '' 'weekly'
    or
change 15 //weekly/
```

To replace characters after TUESDAY'S in line 30 of your data set, you can use the special-delimiter notation:

```
00030 THIS IS TUESDAY'S CHILD
change 30 /tuesday's/
00030 THIS IS monday's child
```

or the quoted-string notation:

```
00030 THIS IS TUESDAY'S CHILD
change 30 'tuesday''s'
00030 THIS IS monday's child
```

Renumbering Lines of Data

The RENUM subcommand of EDIT assigns line numbers to a data set without line numbers, or renumbers 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. The first line will be assigned the number 10 and subsequent line numbers will be increased 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 the following:

```
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. Suppose that starting at line 23 you wish the new line number to be 25 and the increment to be 5. Enter:

```
renum 25 5 23
```

The preceding example shows renumbering of all lines following a given line. You may want to limit the renumbering to a range of lines. You must specify the new line number (greater than the line prior to the old line number), the increment to be used, the old line number (first line to be renumbered), and the end line number (last line to be renumbered). For example, if you want to renumber lines 25 through 50, assigning line number 40 to the first renumbered line and using an increment of 2, enter:

```
renum 40 2 25 50
```

If you use the RENUM subcommand to renumber your data set, the renumber increment that you specify is used when you enter the INPUT subcommand the next time during the edit session. Thus, if the following sequence occurred:

```
list
00010   LINE 1 OF DATA
00020   LINE 2 OF DATA
00030   LINE 3 OF DATA
END OF DATA
renum   3  3
input
INPUT
00012   line 4 of data
00015   line 5 of data
00018   (null line)
EDIT
```

your data set would look like this:

```
00003   LINE 1 OF DATA
00006   LINE 2 OF DATA
00009   LINE 3 OF DATA
00012   LINE 4 OF DATA
00015   LINE 5 OF DATA
```

If you want to override the existing line number increment, use the increment operand on the INPUT subcommand.

The RENUM subcommand capability is also available as an operand of the SAVE subcommand. This means that you can make any of these line number changes as you store a new data set or save updates to an old one, using only a single command line at the terminal. For additional information about this capability, refer to the section "Storing a New Data Set."

Removing Line Numbers

You can use the UNNUM subcommand of EDIT to remove line numbers from a numbered data set. To remove the numbers from all the lines in a data set, enter:

```
unnum
```

If the following sequence of terminal activity occurred:

```
list
00010 LINE 1 OF DATA
00020 LINE 2 OF DATA
00030 LINE 3 OF DATA
END OF DATA
unnum
```

your data set would look like this:

```
LINE 1 OF DATA
LINE 2 OF DATA
LINE 3 OF DATA
```

The UNNUM subcommand capability is also available as an operand of the SAVE subcommand. This means that you can remove a data set's line numbers as you save it, using only a single command line. Additional information about this capability is in the section "Storing a New Data Set."

Listing the Contents of a Data Set

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

```
list
```

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 line numbers, you can use the current line pointer and the number of lines to be listed. 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
```

To list the line pointed at by the current line pointer, enter:

```
list *
```

You can use the SNUM operand of LIST 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.) 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
```

The LIST subcommand uses a standard listing format. If you list a non-line-numbered data set, or a line-numbered data set using the SNUM operand (to suppress line numbers), the lines displayed will consist of only the data portion of the records. To list a non-line-numbered data set your display will look like this:

```
list
LINE 1 OF DATA
LINE 2 OF DATA
LINE 3 OF DATA
END OF DATA
```

If you list a line-numbered data set, the system will suppress up to three leading zeros in each line number, and separate the line number from the data with a blank. The line number is displayed to the left of the data. For example, data with an 8-digit line number would display:

```
list
00010   LINE 1 OF DATA
00020   LINE 2 OF DATA
00030   LINE 3 OF DATA
END OF DATA
```

If you are editing a line-numbered COBOL data set, with a six-character sequence (line number) field, either one or three leading zeros will be suppressed, depending on the command. For the INPUT subcommand, one leading zero is suppressed; for the LIST subcommand three leading zeros are suppressed, as follows:

```
edit a new cobol
INPUT
00010   identification division
00020   program-id.  calc.
00030   environment division
00040   (null line)
EDIT
list
010 IDENTIFICATION DIVISION
020 PROGRAM-ID.  CALC.
030 ENVIRONMENT DIVISION
END OF DATA
```

Moving or Copying Data within a Data Set

You can move or copy any part or all of your data set's content to any other place within the data set. Copying the data implies that when the operation is complete, identical data will be in two or more places within the data set. Moving implies that there will be only a single copy of the data, but that it will be in a new place.

There are two ways to specify the data you wish to move or copy:

- By line number
- By string identification

If you specify your data by line number, the beginning line number in your subcommand is the line in your data set where the move or copy operation is to begin. If you specify your data by string identification, TSO must search for the string of characters to find the line at which to begin moving or copying data.

Specifying Data by Line Number

Suppose you are working with the text data set in Figure 5.

```
00010   This data set is a sample to
00020   show you some things about the
00030   way the EDIT command's MOVE
00040   and COPY subcommands work.
00050   It contains eleven lines.
00060
00070       The first five lines are
00080   for moving and copying. The
00090   sixth line is a blank line to show
00100   a convenient place to move data,
00110   and these last lines complete this thought.
```

Figure 5. Sample Text Data Set for Illustrating the EDIT MOVE/COPY Function

Consider the following sample subcommand for moving a range of line-numbered data:

```
move    10  50  60  incr(2)
```

In this command, the data to move begins with line 10 and ends with line 50, the place to move this data begins at line 60, and once the data is in its new place, its line numbers should be in increments of two. (The default increment is 10.)

The results of entering this subcommand appear in Figure 6.

Because the sample data set already has a line numbered 60, the subcommand apparently conflicts with the space available in the data set. TSO resolves this conflict by adding the increment (in this case 2) to line 60, the line specified for the beginning of the new data, and then it inserts the new lines at the specified increment. In this case, the five moved lines of data begin at line 62 and extend through line 70. To keep from overlaying the original line 70, TSO renumbers that line by adding 1. In this case the newly moved-in line 70 displaces the original line at that position to line 71.

```
00060
00062   This data set is a sample to
00064   show you some things about the
00066   way the EDIT commands's MOVE
00068   and COPY subcommands work.
00070   It contains eleven lines.
00071       The first five lines are
00080   for moving and copying. The
00090   sixth line is a null line to show
00100   a convenient place to move data,
00110   and these last lines complete this thought.
```

Figure 6. Sample Text Data Set after a Move Operation

Note that the beginning line of the data set is now 60, (the blank line) because the first five data lines are now in positions 62-70. This illustrates that after a move operation, a data set retains its original number of lines, although the line numbering is different.

Lines 10-50 could also be moved using the COPY subcommand; however, the first data set would retain lines 10-50 as well as adding lines 62-70 as copies of lines 10-50.

You can move or copy a whole data set to precede or follow itself, simply by specifying its entire range of line numbers. Consider the sample data set again, with this subcommand:

```
copy    10  110  120  incr(10)
```

This subcommand replicates the original data set, starting at line 120. You could also change the increment of the copied data set at the same time.

Consider the same data set with this subcommand:

```
move    10  110  120  incr(10)
```

This subcommand effectively renumbers the data set. Its new line numbers are 120-230, and the original line numbers of 10-110 are no longer in the data set (although they are available for use for future assignments of data). The RENUM subcommand provides a simpler way to achieve this result, however.

Specifying Data by Character String Identification

Refer to the sample data set in Figure 5 again and note that line 10 contains the character string "is a sample". To move the first five lines as in the first example, you could enter:

```
top
move    'is a sample'  5   60  incr(2)
```

The MOVE subcommand's operands mean:

is a sample	-	the data to move includes this unique string of characters in its first line; TSO uses the character string to find the line containing it. This is the beginning line to be moved.
5	-	the data to move consists of the five lines beginning with the line that contains the requested character string
60	-	the place to move the five lines of data begins at line 60.
incr(2)	-	the moved lines will be in increments of two.

The results of this subcommand are identical to those in the previous MOVE example, as Figure 6 shows:

1. The data set now begins with line 60, a null line.
2. The original first five lines now occupy line numbers 62, 64, 66, 68, and 70.
3. The original line numbers 70-110 are now 71-110.
4. The data set's total line count remains the same.

While character string identification of the lines you wish to move or copy works as just explained for numbered data sets, its primary use is in editing unnumbered data sets. For editing data sets without line numbers, the use of this "finding" scheme and relative line counting capability is necessary.

Storing a New Data Set

The data set you create or change remains in 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, the system discards your newly created data set, or your changes. If you want the system to make your new data set permanent, 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, or the SAVE keyword on the END subcommand.

In the following sequence you create a data set named RECORDS and ask the system to store it as a permanent data set. This example uses the SAVE keyword on the END subcommand.:

```
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
end save
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:

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

When you save your data set, you may also make line numbering changes by adding operands to the SAVE subcommand. You can make any of the changes outlined in "Renumbering Lines of Data" by using RENUM as an operand of SAVE, or you may remove the line numbers entirely by using UNNUM.

Suppose in the example above you wish to have TSO store the four-line data set with line numbers beginning with two in increments of three. Instead of the SAVE subcommand shown, enter:

```
save * renum (2 3)
```

The asterisk tells TSO to save and renumber the data set currently being edited.

The next time you list your data set, it will look like this:

```
00002   project 21 7/10-8/25    a. jones
00005   project 23 7/10-9/12    p. smith
00008   project 39 8/1-9/15     r. brown
00011   project 42 8/15-9/21    s. green
```

To delete the line numbers as you save the data set, enter:

```
save *  unnum
```

Note that causing an attention interrupt during SAVE processing when either the RENUM or UNNUM operands were specified may cause undesirable results. It is possible to terminate the entire subcommand or just the RENUM/UNNUM function, depending on the point in execution at which the attention interrupt was entered.

Creating an Updated Copy of a Data Set

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
EDIT
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 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
00040   PROJECT 42  8/15-9/21   S. GREEN
```

Saving Updates to a Data Set

You can use the SAVE subcommand whenever you are using the EDIT command: you can create a data set and save it, 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   (null line)
EDIT
save
EDIT
30            c. alden         2241
input   52 2
INPUT
00052         l. davis         4119
00054         j. egan          6835
00056         e. foster        1384
00058   (null line)
EDIT
list

00010         TELEPHONE LISTING - SALES DEPT
00020         J. ADAMS         1291
00030         C. ALDEN         2241
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
save    * renum
EDIT
end
READY
```

Ending the Edit Functions

Use the END subcommand to terminate the operation of the EDIT command. You can enter the SAVE or NOSAVE operands on the END subcommand to indicate whether or not you want to save the modified data set. Note that causing an attention interrupt during the execution of an END SAVE subcommand may cause undesirable results. It is possible to terminate the SAVE portion of the operation but the END function may continue. Nothing would be saved even though the EDIT mode message may be produced. In this case, subsequent subcommands are ignored because EDIT has completed its function.

If you have made changes to your data set and have not entered the SAVE subcommand or the SAVE/NOSAVE operand on the END subcommand, the system issues a message to respond to this message by entering the word SAVE, if you want to save the modified data set or END, if you do not want to save the modifications. (You cannot enter any operands with SAVE or END at this time.)

After ending EDIT, you will receive the READY message. You can then enter another command.

Renaming a Data Set

The RENAME command allows you to:

- Change the name of a non-VSAM data set. (The Access Method Services ALTER command changes the name of a VSAM data set or a non-VSAM data set in a VSAM catalog. For additional information about ALTER, refer to **OS/VS2 Access Method Services.**)
- Change the name of a member of a partitioned data set.
- Assign an alias to a member of a partitioned data set.
- Rename common qualifiers.

If your LOGON user identification is SMITH and you have a data set named SMITH.RECPT.DATA that you want to change 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
```

Notice 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 under that name. However, if you have two data sets under the same user-supplied name, SMITH.RECPT.DATA and SMITH.RECPT.TEXT, you must specify either RECPT.DATA or 'SMITH.RECPT.DATA' in the RENAME command. 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 either 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 at a time.

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

Renaming a Member of a Partitioned Data Set

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

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

Assigning an Alias to a Member

Use the ALIAS operand to indicate that the new member name is an alias and not a replacement. To assign the alias DAILY to member INPUT of

SMITH.AB79.DATA, use any of the following:

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

After entering this command the member can be referred to as either SMITH.AB79.DATA(INPUT) or SMITH.AB79.DATA(DAILY).

Renaming Common Qualifiers

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. 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." If your logon identifier is JONES, you can then enter the RENAME command as follows:

```
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
```

Listing Information about Your Data Sets

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

LISTALC lists the data sets presently allocated to you. Other information is available about these data sets depending on the parameters you specify.

LISTCAT lists the names of all cataloged data sets that have your user identification as the high level qualifier. The names of the catalogs containing these data sets will also be displayed. Using the optional LISTCAT subparameters will give you information about VSAM data sets much like LISTDS gives for non-VSAM data sets. Detailed explanations of LISTCAT definition and output are in the publication, **OS/VS2 Access Method Services.**

LISTDS gives you information on specific data sets that are currently cataloged, allocated, or both. The information you receive, which is described in detail in **OS/VS2 JCL**, 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

For more information on the LISTALC and LISTDS commands refer to **OS/VS2 TSO Command Language Reference**. LISTCAT is discussed in **OS/VS2 Access Method Services.**

Protecting Your Data Sets

The PROTECT command protects only non-VSAM data sets; TSO issues an error message if you attempt to protect a VSAM data set. To protect VSAM data sets, use the Access Method Services ALTER and DEFINE commands. Discussions of these commands are in **OS/VS2 Access Method Services**.

Deleting a Data Set

Use the Access Method Services DELETE command to delete one or more data sets, or one or more members of a partitioned data set. DELETE is discussed in **OS/VS2 Access Method Services**.

Section IV: Executing Programs at a Terminal

You can use the TSO commands to compile, link-edit, and execute (or compile and load) your source program at the terminal. TSO also allows you to use other programs, such as utilities, at the terminal. That is, instead of taking your job to the computing room to run it, you can use the TSO commands to enter it through your terminal. These commands reduce your job turnaround time because you get immediate results at the terminal. Since TSO commands are designed to operate on cataloged data sets, you should catalog the background data sets created for use with TSO in the foreground.

You can also use the terminal to submit your job for processing at the computer in the conventional manner. That is, you can forego the immediate results at the terminal and either have the results sent to you from the computer room after your job is executed or obtain them at the terminal later. Jobs submitted in this manner are called batch jobs.

Most compilers or assemblers that are available under OS/VS2 are also available from your TSO terminal, for either foreground or background jobs. In addition to these programs, your installation may have one or more of the special TSO programs for your use at the terminal. Some of these programs are:

- 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) compiler modified for the terminal environment
- TSO FORTRAN Prompter -- an initialization routine to prompt you for options and invoke the FORTRAN IV (G1) Processor
- FORTRAN Interactive Debug -- a tool for dynamic debugging of FORTRAN programs (used in conjunction with Code and Go FORTRAN or FORTRAN G1)
- 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 or Version 4 --versions of the American National Standard COBOL compilers with extensions 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 or 4 Processor
- COBOL Interactive Debug -- a tool for dynamic debugging of COBOL programs (used in conjunction with ANS COBOL Version 4)
- TSO Assembler Prompter -- an initialization routine to prompt you for options and invoke the Assembler
- PL/I Optimizing compiler and PL/I Checkout compiler -- both compilers include the PL/I Prompter, which is an initialization routine that prompts you for options and invokes the compiler

If your installation has the PL/I Optimizing compiler or the PL/I Checkout compiler, you can compile and execute PL/I programs under

TSO. These compilers are program products, and each includes the PL/I Prompter, which is an initialization routine that checks compiler options, allocates data sets required by the compiler, and then invokes the compiler.

If your installation has one or more of the TSO program product PL/I compilers, it will provide you with documentation that explains how to use them. This section explains how to use only the programs that are standard, as part of OS/VS2. The following paragraphs describe how you can:

- Allocate a data set.
- Assign data set attributes.
- Free an allocated data set.
- Create a program.
- Compile your program.
- Link-edit a compiled program.
- Execute a program.
- Load a program.

It is assumed that you are familiar with a programming language. The options and data set requirements of the compilers, linkage editor, and loader are summarized in the programmer's guide for the compiler you are using.

Allocating a Data Set

There are three reasons for allocating data sets with the ALLOCATE command or the ALLOCATE subcommand of EDIT:

- To allocate data sets required by the program or compiler you intend to invoke
- To allocate a data set for which special characteristics have been defined with the ATTRIB command
- To allocate data sets required by the linkage editor or loader when you use the CALL command

You should identify the data set requirements for any program that you intend to invoke. In some cases, compilers have prompters that allocate the required data sets for you. The documentation for a program or compiler specifies data set requirements. A data set with unique characteristics assigned by the ATTRIB command may be allocated with the USING operand of the ALLOCATE command.

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 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 and the ALLOCATE subcommand of EDIT.

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 only need to 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. (For additional information about defining LOGON procedures refer to **OS/VS2 System Programming Library: TSO.**)

When you define a data set with the ALLOCATE command or subcommand, it remains allocated until you use the FREE command or subcommand to free it, or until you log off. You may allocate a data set to the terminal by using an asterisk (*) as the data set name.

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 or subcommand.

You can list the data sets allocated to you with the LISTALC command (described in "Listing the Names of Your Data Sets").

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. 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 every time you log on, you can have your installation allocate them in the form of fully defined data sets in the LOGON procedure or you can build a command procedure containing your ALLOCATE commands and execute that procedure as soon as you are logged on. In either case you do not have to type the same ALLOCATE commands every time you log on. (For information about writing the LOGON procedure, refer to **OS/VS2 System Programming Library: TSO.**)

The example in Figure 7 illustrates the use of the ALLOCATE command for allocating the data sets required for an execution of the Assembler. The assembler requires eight data sets for this compilation. They are:

SYSLIB	The macro library (usually 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 entered with the EDIT command and defined to the assembler with the ALLOCATE command.

```
.
.
.
READY
allocate dataset('sys1.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 7. Allocating Data Sets for the Assembler

Assigning Attributes to a Data Set

TSO data set characteristics are called attributes. Generally, you do not have to be concerned with attributes because TSO assigns them automatically. In some instances, however, you may want to allocate a data set with attributes different from those assigned automatically. The ATTRIB command or subcommand provides a way for you to do this.

The ATTRIB command dynamically assigns DCB and other parameters, such as retention and expiration dates, to a data set. This allows you to run existing programs that are dependent upon JCL for specifying certain DCB parameters.

Use the ATTRIB command to build a list of the attributes that you want to assign to a data set. Then use the ALLOCATE command, specifying the name of the attribute list as the value for the USING (attr-list-name) operand. TSO then assigns the attributes in the list to the data set when it is allocated.

You may also assign attributes by specifying the filename of a previously allocated data set as the value for the USING operand on the ALLOCATE command.

You can refer to the attribute list any number of times during your terminal session. When you finish using an attribute list, use the FREE command to delete it from the system.

The attribute list is a null file allocation that other commands can use and modify. Therefore, it is advisable to allocate data sets requiring the attribute list before before issuing any subsequent commands such as LINK or RUN. These commands may cause additional null file allocations.

The operands of the ATTRIB command or subcommand (as discussed in **OS/VS2 TSO Command Language Reference**) correspond to data control block (DCB) and other parameters discussed in the following publications:

- **OS/VS2 JCL**
- **OS/VS2 Data Management Services Guide**

Note: Not all DCB parameters can be specified via ATTRIB.

You should understand the purpose of DCB parameters as presented in these publications before using the ATTRIB command.

The example in Figure 8 illustrates the use of the ATTRIB command. In this example, the attributes are the logical record length, the block size, and the expiration date.

```
attr dcbparms lrecl(24) blksize(96) expdt(72111)
READY
alloc da('attr.show') using(dcbparms) new bl(80) sp(1,1) vol(231400)
READY
free attrlist(dcbparms)
```

Figure 8. Assigning Attributes to a Data Set

Freeing an Allocated Data Set

To release any data sets allocated to you use the FREE command or subcommand. You can also use this command to change the output class of a SYSOUT data set, or to release attribute lists created by the ATTRIB command.

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.

The following example frees the data sets allocated in Figure 7. The output class of the SYSPUNCH and SYSPRINT data sets is changed to B.

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

Creating a Program

To create your source program use 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, FORTGI, FORTH, PLIF, or PLI. You can use the LINE or the LRECL operands to 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 either the SAVE subcommand or the SAVE keyword on the END subcommand to save the data set before you end the EDIT command. Your source program is now ready for compilation.

The example in Figure 9 shows the creation of an assembler source program.

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

Figure 9. 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 invoke the compiler you must use ALLOCATE commands to allocate all the data sets required for compilation. The data sets required by your compiler are described in that program product's user's guide publication.

You must give the data set name of your compiler in the CALL command by which you invoke it.

In addition to the compiler's data set name, you can enter the compiler options you desire in the CALL command. These options are those specified with the PARM parameter of the EXEC statement in JCL. 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 be displayed 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 10 shows the commands required to create a COBOL source program, allocate the eight data sets required for compilation, call the COBOL compiler, and free all allocated data sets except the one that contains the compiled program (object module). This procedure assumes that 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
end save
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 dataset(prog2.cobol) file(sysin) old
READY
call 'sys1.linklib(ikfcbl00)' 'map load nodeck flagw'
.
.
.
.   COBOL listings and messages
.
.
.
READY
free file(syslib,sysut1,sysut2,sysut3,sysut4,sysprint,sysin)
READY
```

Figure 10. COBOL Compilation

Link-Editing a Compiled Program

The LINK command makes the services of the linkage editor available to you. 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.

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:

```
link prog2
```

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:

```
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 goes into member TEMPNAME of the SMITH.LM.LOAD data set.

```
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 goes into member TEMPNAME of CARTER.TRAJ.LOAD.

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

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

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

The system prompts 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 you can specify it with one of the following operands:

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

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 system searches the libraries 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.PLILIB, and DEPT39.LIB.SUBRT2 are to be searched to resolve external references.

```
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, 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:

```
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 be placed in the BILL.LIST.LINKLIST data set.

```
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, it has the same user-supplied name as the object module. For example, if you want the listing to go in the BILL.PRGM.LINKLIST data set, enter:

```
link prgm map print
```

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

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

Error messages appear 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.

```
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 the 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. If you want to execute a program that resides in DEPTB.PROGS.DAILY(NUM3), enter:

```
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 LOGON identifier is "JONES" and if your program resides in JONES.LIB.LOAD(MEM2), enter:

```
call lib(mem2)
```

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

```
call lib
```

You can pass parameters to your program if you wrote it in assembler. These are the parameters you would specify with the PARM parameter of the EXEC statement in JCL. For example, if you want to pass the parameters OPTION1 and OPTION5 to a program that resides in JONES.ASMPG.LOAD(MEM3), enter:

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

Figure 11 shows the commands for link-editing and executing the COBOL program created and compiled in Figure 10. In Figure 10, the commands placed the compiled program (object module) 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 sets are deleted. (The other data sets, allocated in Figure 10, are automatically deleted because they were not given a data set name when allocated.) This procedure assumes that you are using your user identification as part of the data set names.

If your program has an error termination, you can use the facilities of the TEST command to debug your program.

```
READY
link prog2 print(*) map
.
.
.       linkage editor messages and listings
.
.
READY
edit input.data new
INPUT
.
.
.       input data
.
.
EDIT
end save
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 11. Link-Editing and Executing a Program

Loading a Program

The LOADGO command makes the services of the loader available to you. 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, without needing to produce a load module. For complete information on the loader, refer to the publication, **OS/VS Linkage Editor and Loader.**

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:

```
loadgo prog3
```

To load the object modules in JONES.PROG3.OBJ, JONES.COB.OBJ and the load module in JONES.COB.LOAD(TWO), enter:

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

You can also pass parameters to your program if you wrote it in assembler. These are the parameters you would specify with the PARM parameter of the EXEC statement in JCL. For example, if you want to pass the parameters OPTION1 and OPTION5 to a compiled program that resides in JONES.ASMPG.OBJ, enter:

```
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 if so, you can specify it with one of the following operands:

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

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

The following example shows how to load the object module in JONES.PRG.OBJ. The libraries SYS1.PLILIB, and DEPT39.LIB.SUBRT2 are to be searched to resolve external references.

```
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 in JCL. For example, to use the LET and EP(MAIN) options when the object module in JONES.CIR.OBJ is loaded, enter:

```
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 JONES.PRGM.OBJ is to be loaded. The listing produced by the MAP option is to be placed in the JONES.LISTING.LOADLIST data set.

```
loadgo prgm map print(listing)
```

Note: 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 JONES.PRGM.LOADLIST data set, enter:

```
loadgo prgm map print
```

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

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

Error messages appear 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,

```
    loadgo prgm map print noterm
```

Figure 12 shows the commands for loading the COBOL program created and compiled in Figure 10. The loading operation shown in Figure 12 is the equivalent of the link-editing and execution shown in Figure 11. The same data sets required for execution of your program in Figure 11 are also necessary in this example. The object module resides in PROG2.OBJ. The loader does not produce a load module, and therefore only PROG2.COBOL, PROG2.OBJ, and INPUT.DATA are deleted at the end. This procedure assumes that you are using your user identification as part of the data set names.

```
READY
edit input.data new
INPUT
.
.
.       input data
.
.
.
EDIT
end save
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 12. Loading a Program

Section V: Testing a Program at a Terminal

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 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, you can still use the TEST command to obtain 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.) Refer to **OS/VS2 TSO Command Language Reference** for a complete description of the facilities of the TEST command.

If you are not an assembler language 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 abnormally terminated. If you press the ENTER key after the error message and "READY", the system will take a dump. Your other choices are to enter any command, or to enter the word 'TEST' with no operands. Your system programmer 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.

```
MYPROG ENDED DUE TO ERROR +
?
SYSTEM ABEND CODE 0C1
READY
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 display pertinent information about your program. For example, he could ask you to perform procedures similar to the following examples.

Example 1

```
MYPROG ENDED DUE TO ERROR +
READY
test
TEST
listpsw
XRXXXTIE  KEY CMWP  SPM  CC  PROG MASK  INSTR ADDR
00000111  8   1101  0    00  0000       00067AB8
TEST
where 67ab8.
67AB8. LOCATED AT +38 IN (load-module name.csectname)
    UNDER TCB LOCATED AT 660D0.
TEST
list 67ab8.-32n length(32)
```

Begin testing by entering the TEST command, and then 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. You can then enter the last five characters of the PSW that is listed with the WHERE subcommand, and the system will provide the location and the program name in which the ABEND occurred. When you enter LIST in the proceeding manner, the system displays the 32 bytes of instructions prior to the ABEND.

At this time you may list all the registers in the following manner to aid you in solving the problem.

```
list 0r:15r
```

Example 2

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

```
at +0:+200 (go)
at +32
at +8c
at +10a
go +0
```

In this case, TEST sets breakpoints at every instruction in your program between relative addresses 0 and 200 (inclusive), stopping at the first invalid instruction encountered. Breakpoints set at relative address 32, 8C, and 10A override the previous settings. The last GO causes the program to resume execution from the beginning (assuming the first address contains a valid instruction). Before execution of the instruction at any of the breakpoint locations, a message appears at the terminal. If the location is other than 32, 8C, or 10A, execution continues because of the GO subcommand in the subcommand list of the first AT. Before 32, 8C or 10A are executed, the associated AT subcommand causes control to return to the terminal so that you can enter any TEST subcommands before continuing execution.

Example 3

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

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

The values specified would be assigned to register 0, register 1, and register 2.

Example 4

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

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

If you want to display storage and find the absolute address associated with the relative address, enter:

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

Example 5

To list an area of storage greater than 256 bytes, you must use the MULTIPLE keyword of the LIST subcommand. For example, to find a module name that is a DC within the instructions of a module, enter:

```
list a0680. c l(256) m(4)
```

(List the storage beginning at location A0680, translate into printable characters, for length 4 x 256.)

When To Use TEST

There are two basic situations in which you might want to use the TEST command:

- To TEST a currently active program.
- To TEST a program not currently being executed.

You may want to TEST a currently executing program either because it is terminating abnormally, or because you want to check through the current environment to see that the program is executing properly.

If a program is terminating abnormally, you will receive a diagnostic message from the terminal monitor program (TMP) and then a READY message. If you respond with anything but TEST, the system's ABEND routine abnormally terminates your program. If, however, you issue the TEST command (and supply no program name), the TEST command processor gets control, and you can use the TEST subcommands to debug the defective program.

If you just want to look at the current environment of an executing program that is not terminating abnormally, enter an attention interruption. The currently active program remains attached and the TMP responds to your interruption by issuing a READY message. When you issue the TEST command (no program name), the currently active program remains in storage under the control of the TEST command processor. You can then use the TEST subcommands to examine the current environment.

Note: In the case of both the ABEND and the attention interruption, you should not enter a program name following the TEST command. If you do, you will lose the current in-storage copy of the program, whereupon TEST loads a new copy.

Testing a program not currently being executed requires that you enter a program name along with the TEST command. (There are other optional operands of the TEST command, but they are not necessary for this example.) The TEST command processor gets control and loads a copy of the named program. The program can be a newly written TMP, a command processor, or an application program.

Programs to be tested in this manner must be link-edited members of partitioned data sets, or object modules in sequential or partitioned data sets, loadable by the system's loader program. Testing object modules requires a special keyword; refer to **OS/VS2 TSO Command Language Reference**.

While your program is under the control of TEST, you can execute it, one instruction at a time, investigate or alter its environment at any time, change instructions or register contents, force entry into various subroutines, and perform other debugging operations. Note that when testing code that is executing in an abend recovery environment, results are unpredictable and TEST may have to terminate unexpectedly.

The following discussion primarily addresses the debugging of newly written code.

For additional discussion of the TEST command and its operands, see **OS/VS2 TSO Command Language Reference.** The TEST subcommands are listed in Figure 13.

Subcommand Name	Function
= (Assignment)	Assigns values to one or more locations.
AT	Establishes breakpoints at specified locations.
CALL	Initiates execution of a program at a specified address.
COPY	Moves data fields or addresses.
DELETE	Deletes a load module.
DROP	Removes symbolic addresses from the symbol table.
END	Terminates all functions of the TEST command.
EQUATE	Adds symbolic address to the symbol table.
FREEMAIN	Frees a specified number of bytes of real storage.
GETMAIN	Acquires a specified number of bytes of real storage for use by the program being processed.
GO	Restarts a program at the point of interruption or at a specified address.
HELP	Obtain the syntax and function of the TEST subcommands.
LIST	Displays the contents of specified areas of real storage or the contents of specified registers.
LISTDCB	Lists the contents of a data control block (DCB). You must specify the address of the DCB.
LISTDEB	Lists the contents of a data extent block (DEB). You must specify the address of the DEB.
LISTMAP	Displays a storage map of any real storage assigned to a program.
LISTPSW	Displays the program status word (PSW). You may specify the address of any PSW.
LISTTCB	Lists the contents of the task control block (TCB). You may specify the address of any TCB.
LOAD	Loads a program into real storage for execution.
OFF	Removes breakpoints.
QUALIFY	Establishes the starting or base location for symbolic or relative addresses; resolves external symbols within load modules.
RUN	Voids all breakpoints so that a program can execute to termination.
WHERE	Displays the absolute address of a symbol or entrypoint, and its relative location within the CSECT.

Figure 13. The TEST Subcommands

Addressing Restrictions

The TEST command processor can resolve internal and external symbolic addresses only if these addresses are available to TEST. Within certain limitations, symbolic addresses are available for both object modules (processed by the loader) and load modules (fetched by contents supervision). To ensure availability of symbols, use the EQUATE subcommand of TEST to define the symbols you intend to use.

External symbols, such as CSECT names, can be available for both object modules and load modules. Object modules require that the Loader have enough real storage to build in-storage composite external symbol dictionary entries. LOAD modules must have been processed by the linkage editor with the TEST parameter specified, or must have been fetched to main storage by the TEST command or its LOAD subcommand.

Internal symbols are available only for load modules. You can refer to most internal symbols in load modules if you specified the TEST parameter during both assembly and link-editing. Certain internal symbols, however, are not available. These include the names on EQU, DSECT, LTORG, and ORG assembler statements, and the symbolic names contained in system routines that operate in zero protection key.

Symbolic addresses normally cannot be obtained for modules fetched from data sets that have been concatenated to SYS1.LINKLIB by use of a link library list in a member of SYS1.PARMLIB. However, if the TEST command processor brings these modules into real storage (with the LOAD subcommand, or as an operand on the TEST command), then the symbolic addresses within these modules are available to TEST.

If the necessary conditions for symbol processing are not met, you can use absolute, relative, or register addressing, but you cannot refer to symbols, unless you have previously defined them with the EQUATE subcommand of TEST.

Executing a Program under the Control of TEST

Any program, that is a link-edited member of a partitioned data set or an object module in a sequential or partitioned data set can be executed under the control of the TEST command processor. Note that TEST is not an authorized program and cannot be used to test an authorized program. For additional information on authorizèd program execution, see **OS/VS2 System Programming Library: TSO.**

Issue the command TEST followed by the program name and those operands of the TEST command that either define the program or are necessary to its operation. These operands may consist of parameters necessary to the operation of the program under test, the keyword LOAD or OBJECT (depending upon whether the program is a load or an object module), and the keyword CP or NOCP (depending upon whether or not the program to be tested is a command processor).

Any parameters that you specify in the TEST command are passed to the named program as a standard operating system parameter list; that is, when the program under test receives control, register one contains a pointer to a list of addresses that point to the parameters.

If the program to be tested is a command processor, include the keyword CP (the default is NOCP). This causes the test routine to create a command processor parameter list (CPPL), whose address it places into register 1 before loading the program.

Establishing and Removing Breakpoints within a Program

To establish breakpoints within the program under test, use the AT subcommand. Then issue the GO subcommand to begin execution of the program. To begin executing a newly-loaded program, enter the subcommand GO; no address is necessary.

When the TEST command processor encounters the executing program's breakpoints, processing temporarily halts, and the message "AT address" appears at the terminal. You can then examine the executing program, its registers, and data areas to see whether it has been executing properly.

There are two ways to do this:

- Specify a list of subcommands when you issue the AT subcommand.
- Wait for the TEST command processor to return control to you at the terminal each time it encounters a breakpoint.

When the TEST command processor encounters a breakpoint, it issues each of the specified subcommands as though you had entered each of them at the terminal at that time. The subcommands execute and display the results of their execution at the terminal. If you specify GO as the last subcommand, control automatically returns to the program under test at the point of interruption. Otherwise, control returns to you at the terminal after the last TEST subcommand completes execution.

If you determine from the information displayed by the subcommands that your program has executed correctly up to that breakpoint, then issue the GO subcommand to return control to your program. Your program can then resume execution at the point of interruption and continue, either to another breakpoint or to its normal conclusion.

If you choose to establish breakpoints with the AT subcommand without specifying additional subcommands with it, TEST passes control directly to your terminal at each breakpoint and awaits your entry of the additional subcommands. This procedure permits you to check on the progress of your program's execution, one TEST subcommand at a time.

To remove all previously established breakpoints, issue the OFF subcommand without an address operand. To remove only specific breakpoints, or specific ranges of breakpoints, enter their appropriate addresses, or ranges of addresses, as operands of the OFF subcommand.

Displaying Selected Areas of Storage

Use the various LIST subcommands to display the contents of a specified area of real storage, registers, or various control blocks at your terminal, or to write this information to a data set. There are six variations of the LIST subcommand:

LIST
LISTMAP
LISTTCB
LISTDEB
LISTDCB
LISTPSW

LIST: The LIST subcommand displays areas of storage or the contents of registers. The address required as an operand of the LIST subcommand can be one address, a list of addresses, or a range of addresses. You may specify the address as a symbolic address if there is a symbol table that contains the requested symbolic address. If there is no symbolic address (the program was not link-edited or did not have a symbol table), you can use the EQUATE subcommand to create a symbolic address for any location within the program, or you can specify the address as a relative address, an absolute address, or as a register containing an address.

If you use the LIST subcommand to list information found at an address specified by a symbol contained in a symbol table, the listed information appears in the character type and the length specified in the symbol table. You can, however, override the attributes contained in the symbol table by including attribute operands on the LIST subcommand.

Use the LIST subcommand at any point during the execution of your program (use the AT subcommand or an attention interruption to stop the execution of the program) to determine whether data areas and registers contain proper data. If the data displayed is not what it should be, use the TEST subcommands to determine why the data is not as expected, or to modify the data in real storage and continue execution of the program.

LISTMAP: The LISTMAP subcommand displays a map of all real storage assigned to the program under test. Some of the information in this map is:

- Region size
- Task control block (TCB) address
- Program name, length, and location in real storage
- Active request blocks (RBs), RB types, and the names of the programs associated with each of the RBs

LISTTCB: The LISTTCB subcommand displays the entire task control block (TCB) of the program under test, or any fields of that TCB. The displayed information is formatted, and each field is identified according to the field names contained in **OS/VS2 Data Areas**.

If you want to display the TCB for the program under test, enter the subcommand LISTTCB with no address. If you want to display another TCB on the TCB queue, you must include the address of the TCB as an operand of the LISTTCB subcommand.

LISTDEB: The LISTDEB subcommand displays the basic section and any direct access sections of any valid data extent block (DEB), or any fields of that DEB. The displayed information is formatted according to the field names of the DEB as contained in **OS/VS2 Data Areas**.

The LISTDEB subcommand requires the address of a DEB as an operand.

LISTDCB: The LISTDCB subcommand displays the contents of a data control block (DCB). The information displayed is formatted, and each field is identified according to the field names of the DCB contained in **OS/VS2 Data Areas**.

The LISTDCB subcommand requires the address of a DCB as an operand. If you have created the DCB within the program under test, use the address of the DCB macro instruction used to create the DCB. You can also obtain the address of the DCB from the DEBDCBAD field of the DEB displayed with the LISTDEB subcommand.

LISTPSW: The LISTPSW subcommand displays the current program status word (PSW) or any of the PSWs. If you issue the subcommand LISTPSW with no address following the subcommand, the current PSW is displayed at your terminal. If you want to display any of the other PSWs at your terminal, supply the address of the PSW you want to see as an operand of the LISTPSW subcommand. A list of the permanent real storage locations of all PSWs can be found in the **IBM System/370 Principles of Operation**.

Changing Instructions, Data Areas, or Register Contents

Once you have listed those areas of real storage that help you determine what has occurred in your program, you can use the assignment function of the TEST command to make changes within the real storage copy of the code, or to change the contents of data areas or registers.

Enter the address at which you want the new data entered, a code indicating the data type, and the new data you want entered at that address. The address must conform to the address restrictions already discussed. The new data must be within single quotes. The data type codes are listed in **OS/VS2 TSO Command Language Reference.**

One problem that can arise during a debugging session occurs when you want to replace a section of the program under test but the replacement code is longer than the section to be replaced. If you type in the beginning address of the section to be replaced, followed by a portion of code longer than the segment to be replaced, you will overlay some functional code. You can solve this problem with the GETMAIN subcommand of TEST.

1. Issue the GETMAIN to obtain a work area in which to build your replacement segment of code. (The GETMAIN subcommand displays at your terminal the address of the beginning of the real storage area it got for you.)
2. Use the assignment of values function of the TEST command to place a branch instruction to the getmained area at the address in your program that begins the code you want to replace.
3. Use the assignment function again, this time to build your newly-written code segment in the getmained area.

You can then use the GO subcommand to restart your program at some point before the branch. Your program will execute through the branch instruction, into the new instructions, and branch back into your original code. Later, you can use the LIST subcommand to display the newly written code in a form useful to you, enter it into your program with the TSO EDIT command, and reassemble your module.

Forcing Execution of Program Subroutines

You may need to test your module's response to return codes set by other modules or, possibly, by unwritten code. To do this type of testing, you can use the following procedure:

1. Use the AT subcommand to insert a breakpoint in your program at the point where it passes control to the unwritten code.
2. Use the assignment function of TEST to set register 15 to an expected return code.
3. Use the GO subcommand to restart your program executing at the point where it would ordinarily get back control.

By using this procedure, you can test your program's response to each possible return code.

Using TEST after a Program ABEND

If a program running under TSO terminates abnormally (ABEND), a diagnostic message containing the ABEND code appears at the terminal, ABEND processing halts, and control returns to either the TMP or TEST. If the program was running under the control of the TEST command processor, control returns to TEST and you can immediately begin to use the TEST subcommands to determine the cause of the error. If the program was not running under TEST, control returns to the TMP. You can then enter the TEST command (without a program name), to place the abnormally terminating program under control of the TEST command processor.

Issue the WHERE subcommand to determine where the interruption occurred. The WHERE subcommand displays the current instruction address at the terminal. If you then enter WHERE followed by that instruction address, WHERE responds by displaying the program name, the CSECT name, the offset of the current instruction address within the CSECT, and the address of the abnormally terminating task's TCB. The instruction address and the information returned by the WHERE subcommand pinpoint the point of error.

The LIST subcommand displays the instructions leading up to the error condition, and the data areas and registers used in those instructions. This information should be sufficient to determine the cause of the error.

Determining Data Set Information

If you want to investigate the condition of any of your data sets, perform the following operations:

1. Use the LISTTCB subcommand to display the TCB for the terminating task.
2. Use the contents of the TCBDEB field as an operand of the LISTDEB subcommand to gain access to the data extent block queue.
3. Use the contents of the DEBDCBAD field in each of the DEBs in the DEB queue, or the addresses of any DCB macro instructions coded within your program, as an operand of the LISTDCB macro instruction, to list the data control blocks.

These control blocks contain the addresses of other control blocks useful in the debugging process.

Section VI: Using Command Procedures

Command procedures are executable sequences of TSO commands, subcommands, and command procedure statements. The entire TSO command language is available to command procedures. Additionally, command procedure statements, symbolic substitution facilities, control variables, and built-in functions give command procedures capabilities similar to those of high-level languages.

The following topics describe how to create and store a command procedure and how to invoke it. There is also an overview of the command procedure facilities.

Creating a Command Procedure

To create a command procedure, use the EDIT command to put the commands, subcommands, and command procedure statements into a data set. This CLIST (command list) data set may be either sequential or partitioned. A sequential CLIST data set consists of only one command procedure, while a partitioned data set may contain more than one command procedure. When a partitioned data set consists entirely of command procedures it is called a command procedure library.

The following sample EDIT session shows how to create a sequential CLIST data set that contains the command procedure named "listpgm" This command procedure loads and passes control to a program called "weekly" after allocating three data sets that "weekly" requires.

```
edit listpgm clist new

INPUT
00010 allocate dataset( input) file( indata) old
00020 allocate dataset( output) block( 100) space( 300,10) +
00030 file( outdata)
00040 allocate dataset( list) file( print)
00050 call weekly
00060   (null line)

EDIT

end save
```

As a result of the END subcommand, TSO stores your data set with the fully-qualified name of:

```
    userid.LISTPGM.CLIST
```

The CLIST qualifier identifies the data set as a command procedure.

To create a command procedure library, use EDIT to make the command procedure a member of a partitioned data set. For example, to create a partitioned data set named "clistlib" and make the command procedure named "listpgm" a member of "clistlib", invoke EDIT as follows:

```
    edit clistlib( listpgm) clist new
```

Command procedures are stored and cataloged like any other TSO data sets. Your installation may allocate a partitioned data set to be used as a central command procedure library. Or you may create your own command procedure library by making your command procedures members of a partitioned data set.

How to Invoke a Command Procedure

To invoke a command procedure, use the EXEC command or EXEC subcommand of EDIT. (Be aware that command procedures executing under EDIT are in the subcommand environment and, therefore, can execute only EDIT subcommands and command procedure statements -- not other commands.)

Use of EXEC (either command or subcommand) can be in one of two forms:

- Explicit form of EXEC -- enter the word EXEC and the command procedure's name.
- Implicit form of EXEC -- enter only the command procedure's name. This will work only for command procedures that are members of a partitioned data set allocated to a file named SYSPROC.

Using the Explicit Form of EXEC

You can use the explicit form of EXEC to invoke any command procedure. To do so, enter EXEC followed by the command procedure's name. The command procedure name can be either the name of a sequential data set or the name of a member of a partitioned data set. For example, to execute a command procedure named "listpgm", which is a sequential CLIST data set, enter:

```
exec listpgm
```

This command causes TSO to execute the procedure named "userid.listpgm.clist".

If the command procedure is a member of a partitioned data set, you must enter both the data set name and the member name. For example, if "listpgm" is a member of a partitioned data set named "clistlib", enter:

```
exec clistlib(listpgm)
```

This command causes TSO to execute the procedure named "listpgm", which is a member of the partitioned data set named 'userid.CLISTLIB.CLIST.'

Using the Implicit Form of EXEC

You can use the implicit form of EXEC to invoke command procedures that are members of a partitioned data set allocated with a filename of SYSPROC. To do so, enter just the procedure name.

Before you implicitly invoke a command procedure, you may want to ensure that the partitioned data set has a filename of SYSPROC. Assume the command procedure name is "listpgm" and it is a member of a data set named "clistlib". Enter:

```
listalc status
```

This command lists all the names of data sets that you have allocated with the ALLOCATE command, that were allocated in your LOGON procedure, and that were temporarily allocated by command processors. If "clistlib" does not appear in this list with a file name of SYSPROC, allocate it by entering:

```
allocate dataset(clistlib.clist) file(sysproc) shr
```

Allocating the file with a shared disposition allows others to use the command procedure library concurrently.

You can now invoke the command procedure. Enter:

```
listpgm
```

Shortening TSO's Search Time

Ordinarily, when you invoke a command procedure implicitly, TSO searches several libraries before it searches the SYSPROC file. It does this to determine first if the name you entered is a TSO command. You can have TSO search only the SYSPROC file by prefixing a percent sign (%) to the command procedure name. For example, enter:

```
%listpgm
```

Concatenating Data Sets to SYSPROC

You may concatenate data sets and allocate them to the file SYSPROC to create a command procedure library that spans several data sets. This is useful if your installation has defined a command procedure library to which you would like to add a library of your own command procedures. You could similarly concatenate your library to one or more of someone else's libraries or concatenate them all to the installation's library.

To concatenate data sets with the ALLOCATE command, enter their names (separated by delimiters) in the order in which you wish TSO to concatenate them. The concatenation order establishes the order by which TSO will search the data sets to find specified command procedures. Therefore, you should specify the data set you expect to use most frequently first.

The blocksizes of data sets can affect their concatenation order. Where blocksizes vary, the system requires that you specify the dataset with the largest blocksize first. If, for example, you wish to concatenate your private library to the installation's library so that your library is the first in the concatenation order, make sure that your library's blocksize is at least as large as that of the installation's library. To determine the blocksize of the installation's library, display its data set attributes with the LISTDS command.

For example, assume your userid is D58JSG1 and you wish to concatenate the command procedure libraries whose fully-qualified data set names are as follows:

```
D58DEW1.CLISTLIB.CLIST
D58JSG1.CMPRCLIB.CLIST
D95MRT1.PROFLIB.CLIST
```

The following ALLOCATE command concatenated them in the order listed and allocates them to file SYSPROC so that command procedures from any of them may be invoked implicitly:

```
allocate file(sysproc) dataset('d58dew1.clistlib.clist
  cmprclib.clist 'd95mrt1.proflib.clist') shr
```

The hyphen at the end of the first line indicates that more command information follows on the next line. The single quotes are necessary because the names are fully-qualified names. The disposition of SHR permits concurrent use of the allocated data sets by other users.

Command Procedure Facilities

A command procedure can be basic, consisting only of a series of TSO commands, or complex, using some or all of the command procedure facilities. These facilities include built-in functions, which perform immediate evaluations of character strings; control variables, which contain information pertinent to the currently executing command procedure; and command procedure statements, which give the writer of command procedures the capabilities of a high-level language. These facilities are described in detail in following topics.

The descriptions of these facilities often refer to expressions, symbolic variables, and labels. You should understand these terms, as used in command procedures, before you attempt to use the command procedure facilities. These terms are defined in the following topics.

Terminology

Throughout this section on command procedures, certain terms are used extensively. The next topics define expressions, operators, symbolic variables, and labels, as used in command procedures.

Operators and Expressions

Operators are used in command procedures to specify operations to be performed on terms in an expression. Expressions are used as parameters on some command procedure statements.

Operators are in three categories:

- Arithmetic operators, which specify fixed-point arithmetic operations to be performed on numeric operands. These operators connect whole numbers, character strings, symbolic variables, control variables, and built-in functions to form simple expressions.
- Comparative operators, which specify comparison functions to be performed between two simple expressions, and thereby form comparative expressions. Comparative expressions are often used to determine conditional branching within a command procedure. Note that, to compare character strings, TSO uses the standard EBCDIC collating sequence.
- Logical operators, which specify a logical connection between two comparative expressions, and thereby form logical expressions. Logical expressions are ofter used to determine conditional branching within a command procedure.

When using expressions in command procedures, you should be aware of the valid range which a numeric variable can have. This range is -2, 147, 483, 648 (the maximum negative number) to +2, 147, 483, 647 (the maximum positive number). This range covers values from a minus two to the 31st power through a plus two to the 31st power minus one.

If a number outside the valid range is entered directly in a command procedure statement, an error code is issued and evaluation of the statement terminates. If the result of any arithmetic calculation (even an intermediate result) is outside the valid range, an error code is issued and, for any statement other than SET, evaluation of the statement terminates. For a SET statement, the error code can be ignored and evaluation of the statement continues if NOFLUSH has been specified in the command procedure.

Figure 14 lists the operators in the three categories and shows how to enter them.

	For the function:	Enter:
Arithmetic	Addition Subtraction Multiplication Division Exponentiation Remainder	+ - * / **(see Note 1) //
Comparative	Equal Not equal Less than Greater than Less than or equal Greater than or equal Not greater than Not less than	= or EQ ¬ = or NE < or LT > or GT < = or LE > = or GE ¬> or NG ¬< or NL
Logical	And Or	& & or AND | | or OR
Note 1: Negative exponents are handled as exponents of zero.		

Figure 14. Arithmetic, Comparative, and Logical Operators

Symbolic Variables

The term "symbolic variable" refers to any character string in a command procedure for which different values may be substituted at different times. Symbolic variables add flexibility to command procedures by symbolizing real values that can change during execution of a command procedure.

Symbolic variables can be used on TSO commands and subcommands, on certain command procedure statements, as file names for file input/output processing, and as global parameters. TSO has predefined a set of special-purpose symbolic variables; these are control variables and built-in functions (see "Built-In Functions" and "Control Variables" in this section). The writer of the command procedure can also define his own symbolic variables; before he can use them, though, they must appear on certain command procedure statements (see PROC, READ, READDVAL, GLOBAL, SET, and OPENFILE statements).

A symbolic variable consists of an ampersand (&) followed by a maximum of 31 alphameric characters, the first of which is alphabetic. The real values to be substituted for a symbolic variable are supplied by the invoker of the command procedure, by the writer of the command procedure, or by the system.

TSO scans each line in a command procedure and replaces the symbolic variables with their real values in a process called symbolic substitution. The real value substituted for a symbolic variable may actually be another symbolic variable (nested symbolic variables). If there are nested symbolic variables, the line is scanned more than once until all symbolic variables are resolved.

The use of double ampersands requires special processing by the symbolic substitution routine. Each pair of ampersands is replaced by a single ampersand. This substitution takes place only after all other symbolic substitution in a line is complete.

You can concatenate symbolic variables to modify existing variable names. For example, symbolic variable &N can be set to a value of 1 and concatenated to a symbolic variable &DSN:

```
&DSN&N
```

As the value of &DSN changes during command procedure processing, you could have the following real values:

```
ALPHA1
BETA1
KAPPA1
```

You could then reset the value of &DSN and increase &N. You could then have the following real values:

```
ALPHA2
BETA2
KAPPA2
```

This is a useful technique when portions of several character strings are identical.

Concatenating symbolic variables and character string requires a period as a delimiter when the symbolic variable name precedes the character string. For example:

```
&DSN.1
```

No delimiter is required when the character string precedes the symbolic variable. For example;

```
ALPHA&N
```

Labeling within Command Procedures

Labels are names by which you can identify particular TSO commands, TSO subcommands, or command procedure statements for branching purposes. Labels must be one to eight alphameric characters, the first being alphabetic, and must be unique names within the command procedure.

To use a label, enter it first in a line, then follow it immediately with a colon, one or more delimiters (blanks, commas, or tabs), and the command, subcommand, or command procedure statement. The following examples show two valid labels:

```
target: alloc dataset (input) file(indata) old
here1234: set &a = &a+1
```

Note the following restrictions on the use of labels:

- Labels cannot be used on PROC statements or on ENDDATA statements.
- TSO never lists labels even though you may have specified the CONLIST option on the CONTROL statement.

Built-In Functions

Built-in functions provide the ability to perform certain evaluations of expressions and character strings. To request a built-in function, specify the appropriate symbolic variable with an expression or character string on a command procedure statement. TSO evaluates the expression first, if necessary, and then performs the requested function. The symbolic variable is then replaced by the result of performing the built-in function. Note that the expression is normally another symbolic variable which may have been set previously in the command procedure.

Figure 15 lists the available built-in functions and a brief explanation of their use. They are explained in more detail in the following topics.

Symbolic Variable	Use
&DATATYPE(expression)	To determine if an expression is entirely numeric.
&EVAL(expression)	To determine the result of an arithmetic expression.
&LENGTH(expression)	To determine the number of characters in the result of an evaluated expression.
&STR(string)	To use a string of characters as a real value.
&SUBSTR(expression [:expression],string)	To use a portion of a character string as a real value.

Figure 15. Built-In Functions

Example 3 in the following topic, "Command Procedure Examples," shows the use of the built-in functions &DATATYPE, &LENGTH, &STR, and &SUBSTR in a command procedure.

Determining an Expression's Type

Entering the &DATATYPE symbolic variable causes the associated built-in function to determine whether or not the evaluated expression is entirely numeric. The expression to be evaluated appears within the parentheses after the symbolic variable:

```
&datatype( expression)
```

After evaluating the expression, TSO replaces this variable with either:

- CHAR -- The evaluated expression contains at least one non-numeric character.
- NUM -- The evaluated expression is entirely numeric.

The following examples show the evaluations of various expressions:

Built-In Function Expression	Resulting Evaluation
&datatype(alphabet)	CHAR
&datatype(1234)	NUM
&datatype(sys1.proclib)	CHAR
&datatype(3*2/4)	NUM
&datatype(A1234)	CHAR

Evaluating an Arithmetic Expression Immediately

Entering the &EVAL symbolic variable causes the associated built-in function to evaluate the indicated arithmetic expression. To indicate that immediate evaluation is required within a statement, enter:

```
&eval(expression)
```

The expression may be any valid arithmetic expression. TSO evaluates it and places the result in the space represented by &EVAL(expression) within the statement. During execution of the statement, therefore, a numeric value replaces &EVAL(expression). For example, consider the following:

```
&eval(3+5-2)
```

The result of evaluating the arithmetic expression within the parentheses is six. The value "6", therefore, replaces the entire string "&eval(3+5-2)" during execution of the statement in which the string appears.

Determining an Expression's Length

Entering the &LENGTH symbolic variable causes the associated built-in function to determine the number of characters in the evaluation of an indicated expression. To indicate the expression you want evaluated, put it in parentheses following &LENGTH, as follows:

```
&length(expression)
```

The result after performing the built-in function is a numeral representing the number of characters constituting the evaluated expression.

For example, the numeral replacing &LENGTH in the following except:

```
&length(1 + 1)
```

is "1" because, although the evaluation of one plus one is two, that answer is only one character.

If there are leading zeroes in the expression, they are ignored when evaluating &LENGTH. The numeral replacing &LENGTH in the following example is 2:

```
&LENGTH(00045)
```

Defining a Character String for Symbolic Substitution

Entering the &STR symbolic variable causes the associated built-in function to use the indicated character string as a real value for symbolic substitution. To use the &STR built-in function, enter:

```
&str(string)
```

The string within the parentheses may be any valid expression. Within this parenthesized expression, nested built-in functions and symbolic substitution take place. After these operations, however, no further evaluation is done.

Consider the following:

```
&str( 1+1 )
```

Because the &STR built-in function inhibits arithmetic evaluation, the result of using this built-in function is the character substitution "1+1", not "2".

Assume that the value of &ONE is 1 and consider this second example:

```
&str( &one + &one )
```

The value substituted for this expression is also "1 + 1".

The &STR symbolic variable serves as a "mask" for a syntactically misleading or invalid expression, thereby permitting the procedure writer to code his intended expression so that unintentional results will not occur.

Defining a Substring for Symbolic Substitution

Entering the &SUBSTR symbolic variable causes the associated built-in function to use a part of the indicated string of characters as a real value during symbolic substitution. When using &SUBSTR, enter the substring and character string information in parentheses following &SUBSTR. To designate the substring, indicate numerically where in the character string the substring of characters is to start and end. This substring can be only one character or can be the entire character string.

The syntax of the &SUBSTR built-in function is:

```
&substr( start-expression:end-expression,
character-string )
```

The information in parentheses may be characters or symbolic variables in any combination, as long as the final values of "start-expression" and "end-expression" are numeric values.

Assume you designate the alphabet as a character string and wish to select a range of letters from it. The most direct way to do this is as follows:

```
&substr( 2:8,abcdefghijklmnopqrstuvwxyz )
```

This entry specifies that you wish to use the letters B, C, D, E, F, G, and H from the alphabet for symbolic substitution.

You could enter the same information with symbolic variables. Assume that you have assigned the following values to symbolic variables:

- &alphabet contains abcdefghijklmnopqrstuvwxyz
- &range contains 2:8
- &testcond contains abcdefg

If you compared &substr(&range,&alphabet) to &testcond, the result would show that they are not equal.

Control Variables

Control variables represent information relevant to the current command procedure environment and user. Command procedures can access the information represented by these variables by including the appropriate symbolic variable in a command procedure statement. TSO replaces the symbolic variable with the requested information.

A command procedure may explicitly set four of the control variables. The contents of all the other control variables are the result of TSO's monitoring of the current command procedure environment. An attempt by a command procedure to change their contents results in an error.

The control variables can be divided into three categories: user-oriented variables, variables related to the current command procedure, and variables related to the system environment. Figure 16 summarizes the control variables in each category, the information each variable represents, and whether the information can be modified by the command procedure.

Category	Symbolic Variable	Information Represented by the Variable	Can be Modified by the Procedure
User-oriented	& SYSUID	Current user's identification	No
	& SYSPROC	LOGON procedure name	No
	& SYSPREF	Data set name prefix	No
Related to the current command procedure	& LASTCC	Most recent return code	Yes
	& MAXCC	Highest return code	Yes
	& SYSICMD	Implicit execution member name	No
	& SYSSCAN	Symbolic substitution rescan limit	Yes
	& SYSDLM	Terminal delimiter	No
	& SYSDVAL	Terminal parameters	Yes
	& SYSNEST	Nested procedure indicator	No
	& SYSPCMD	Current primary command name	No
	& SYSSCMD	Current subcommand name	No
Related to the system environment	& SYSTIME	Current time	No
	& SYSDATE	Current date	No

Figure 16. Control Variables

Example 2 in the following topic "Command Procedure Examples" shows the use of several control variables in a command procedure named "D95MRT2.CLIST(PROFILE)".

User-Oriented Control Variables

The user-oriented control variables represent information directly related to the TSO user who invoked the command procedure. This information is the user's identification, the LOGON procedure name, and the data set name prefix.

&SYSUID -- User's Identification

The &SYSUID control variable contains the userid associated with the current terminal session. TSO maintains this variable; a command procedure can access this variable but cannot modify it.

&SYSPROC -- LOGON Procedure Name

The &SYSPROC control variable contains the procedure name specified when the current command procedure user logged on to TSO. TSO maintains this variable; a command procedure can access this variable but cannot modify it.

&SYSPREF -- Data Set Name Prefix

The &SYSPREF control variable contains the current data set name prefix, which the user sets by using the PROFILE command. TSO maintains this variable; a command procedure can access this variable but cannot modify it.

Control Variables Related to the Current Command Procedure

The control variables related to the current command procedure represent information that pertains to the command procedure currently being executed. This information includes return codes, command names, and terminal input.

&LASTCC -- Most Recent Return Code

The &LASTCC control variable contains the return code from the last TSO command, TSO subcommand, command procedure statement, or nested command procedure executed. The command procedure can modify this control variable.

The return codes from command procedure statements and TSO commands are listed in **OS/VS2 TSO Command Language Reference.**

&MAXCC -- Highest Return Code

The &MAXCC control variable contains the highest return code from a TSO command, subcommand, or command procedure statement in the currently executing command procedure, or from a nested command procedure. The command procedure can modify this control variable.

&SYSICMD -- Implicit Execution Member Name

The &SYSICMD control variable contains the name by which the user *implicitly* invoked the currently executing command procedure. If the user invoked the command procedure explicitly, this variable has a blank value. TSO maintains this variable; a command procedure can access this variable but cannot modify it.

The writer of the command procedure can use this variable to determine by which of several alias names the user invoked the procedure. Each alias could indicate a different path within a general-purpose command procedure that provides several related functions.

&SYSSCAN -- Symbolic Substitution Rescan Limit

The &SYSSCAN control variable contains a value limiting the number of times that the TSO symbolic substitution routine may scan each line in order to substitute values for all the symbolic variables that the line may contain. The default scan limit is 16. A command procedure may modify this variable, specifying a value from 0 to 2^{31}. A zero limit inhibits all scans, preventing any substitution of values for symbolic variables.

&SYSDLM -- Terminal Delimiter

The &SYSDLM control variable contains a number that identifies by position (first, second, third, etc.) the TERMIN delimiter string entered by a terminal user to return control to the command procedure. TSO maintains this variable; a command procedure can access this variable but cannot modify it.

This variable can be used to determine what action should be taken when the terminal user returns control to the command procedure, based on the user's choice of the TERMIN delimiter.

&SYSDVAL -- Terminal Parameters

The &SYSDVAL control variable contains either:

- Any parameters the terminal user entered, besides the delimiter, when he returned control to the command procedure after a TERMIN statement
- The terminal user's response after a READ statement requests terminal input

A command procedure may modify the contents of &SYSDVAL. The initial value of &SYSDVAL is blank, and it remains blank if the user does not specify any parameter information after the TERMIN delimiter, or if the user's response to a READ statement is a blank line.

&SYSNEST -- Nested Procedure Indicator

The &SYSNEST control variable contains 'YES' if the currently executing command procedure is a nested procedure, or 'NO' if it is not. (A nested procedure is one that was invoked by another command procedure rather than by the terminal user.) TSO maintains this control variable; the command procedure can access this variable but cannot modify it.

&SYSPCMD -- Current Primary Command Name

The &SYSPCMD control variable contains the name of the TSO command that the command procedure has most recently executed. The initial value of &SYSPCMD is EXEC or, if EXEC was issued as a subcommand of EDIT, EDIT. TSO maintains this control variable; the command procedure can access this variable but cannot modify it.

This variable can be used, for example, as an identifier in a message written from an error exit routine.

&SYSSCMD -- Current Subcommand Name

The &SYSSCMD control variable contains the name of the TSO subcommand that the command procedure has most recently executed. The initial value of &SYSSCMD is blank, if the EXEC command was issued, or EXEC, if the EXEC subcommand of EDIT was issued. TSO maintains this control variable; the command procedure can access this variable but cannot modify it.

The &SYSSCMD and &SYSPCMD control variables are correlational. For example, the contents of &SYSPCMD could be "EDIT" and the contents of &SYSSCMD could be "END". Then the procedure might execute the CALL command. At this point, the contents of &SYSPCMD become "CALL" and the contents of &SYSSCMD become blank. The &SYSSCMD variable is useful as an identifier in a message written from an error exit routine.

Control Variables Related to the System Environment

Control variables related to the system environment represent information that is dependent on the current TSO environment. This information is the time of day and the date.

&SYSDATE -- Current Date

The &SYSDATE control variable contains the current date in the format mm/dd/yy, where mm is month, dd is day, and yy is year. TSO maintains this variable; a command procedure can access this variable but cannot modify it.

&SYSTIME -- Current Time

The &SYSTIME control variable contains the current time of day in the format hh:mm:ss, where hh is hours, mm is minutes, and ss is seconds. TSO maintains this variable; a command procedure can access this variable but cannot modify it.

Command Procedure Statements

Command procedure statements supplement the TSO command language and can be used in both the command and subcommand environments. There are four categories of command procedure statements:

- Control statements, which influence execution by naming functions to be performed, setting processing options, redirecting control, and altering execution sequence.
- Assignment statements, which assign values to variables.
- Conditional statements, which establish and test conditions in the sequence of commands and statements to determine the logical flow of execution.
- File access statements, which open, access, and close QSAM data sets to give command procedures I/O capability.

Figure 17 summarizes these categories of statements.

	Type of Statement:		
Control	**Assignment**	**Conditional**	**File Access**
ATTN	READ	IF-THEN-ELSE	OPENFILE
CONTROL	READDVAL	DO-WHILE-END	GETFILE
DATA-ENDDATA	SET	(WHEN TSO command)	PUTFILE
ERROR			CLOSFILE
EXIT			
GLOBAL			
GOTO			
PROC			
RETURN			
TERMIN			
WRITE			
WRITENR			

Figure 17. Summary of Command Procedure Statement Categories

The following topics describe the functions that are performed by the various command procedure statements:

- Establishing initial parameters
- Establishing processing options
- Assigning values to symbolic variables
- Controlling execution flow
- Communicating with the terminal user
- Performing file input and output
- Executing nested command procedures
- Establishing exit routines

Establishing Initial Parameters

Most command procedures use symbolic variables. There are two ways that real values can be assigned to symbolic variables:

- The writer of the command procedure can assign real values to symbolic variables by using command procedure statements.
- The writer of the command procedure can require the invoker of the procedure to supply real values for certain symbolic variables when he invokes the procedure.

Parameters on the PROC statement identify the symbolic variables for which the invoker must supply real values. These values cannot be predetermined by the writer of the command procedure; the invoker must determine the values before he invokes the procedure. Also, the invoker can change these values to get different results each time he invokes the command procedure. For example, these values could be data set names that are to be used within the command procedure or indications of optional functions to be performed, such as listing the commands at the terminal.

There are two types of parameters that can be specified on the PROC statement:

- Positional parameters, for which the invoker must supply real values. These must appear in the same order as they appear on the PROC statement.
- Keyword parameters, which are optional and, if specified, can be in any order after the positional parameters.

Use of the PROC Statement

You need to code a PROC statement in a command procedure only if the procedure requires the invoker to supply information. If the PROC statement is coded, it must be the first statement in the procedure. The format of the PROC statement is:

```
proc n [posit1 ... positn] [keyword [ ( [ subparameter ] )]]
```

The n denotes the number of positional parameters coded on the PROC statement. This number cannot exceed eight digits. If there are no positional parameters, n must be 0.

Posit1 ... positn indicates the positional parameters for which the invoker must supply real values. TSO prompts him for any values he does not enter. Each positional parameter name cannot exceed 252 alphameric characters and the first character must be alphabetic.

Keyword indicates an optional symbolic variable for which the invoker may supply a real value. The keyword name cannot exceed 31 alphameric characters and the first character must be alphabetic. Keyword parameters can have a subparameter associated with them. The subparameters can have a default value to be used if the invoker does not supply a value.

Figure 18 lists the types of parameters that can be specified on a PROC statement and, for each type, how the writer of the command procedure would code it on the PROC statement, how the invoker of the procedure would specify it in the value-list on the EXEC command, and what the results would be.

Parameter type	Specified by writer of procedure on PROC statement	Specified by invoker of procedure in EXEC value-list	Result
positional	proc 1 dsname	mylib.asm	Symbolic variable &dsname has a real value of "mylib.asm".
		nothing specified	User is prompted; assume he then enters "cmdproc.clist"; symbolic variable &dsname then has a real value of "cmdproc.clist".
keyword without subparameter	proc 0 list	list	Symbolic variable &list has a real value of "list".
		nothing specified	Symbolic variable &list has a real value of blanks.
keyword with subparameter and no default	proc 0 list()	list	User is prompted; assume he then enters "all"; symbolic variable &list then has a real value of "all".
		list(all)	Symbolic variable &list has a real value of "all".
		nothing specified	Symbolic variable &list has a real value of blanks.
keyword with subparameter and default	proc 0 list(none)	list	User is prompted; assume he then enters "all"; symbolic variable &list then has a real value of "all".
		list(some)	Symbolic variable &list has a real value of "some".
		nothing specified	Symbolic variable &list has a real value of "none", the default.

Figure 18. Results of Entering Positional and Keyword Parameters

Note that use of quoted-string notation affects the way a user passes a dataset name to a command procedure when the substituted data set name is to be enclosed in single quotes. If a keyword with value is being substituted, the invoker of the command procedure must:

- Enclose the dataset name in three single quotes on an implicit EXEC command.
- Enclose the dataset name in six single quotes on an explicit EXEC command.

For example, assume a command procedure named EXAM has these statements:

```
PROC 0 LIB( )
ALLOC DA( &LIB. ) SHR
END
```

If the invoker of the command procedure wanted to substitute 'SYS1.TSO.CLIST' for LIB, then he would code on an explicit EXEC command:

```
EXEC EXAM 'LIB(''''''SYS1.TSO.CLIST'''''')'
```

On an implicit EXEC command he would code:

```
EXAM LIB('''SYS1.TSO.CLIST''')
```

By following this procedure, the ALLOC command in the command procedure would then be:

```
ALLOC DA('SYS1.TSO.CLIST') SHR
```

Establishing Processing Options

The CONTROL statement establishes basic processing options to be in effect during the execution of a command procedure. These options include whether prompting can be done while the procedure is executing, what type of displays are desired while the procedure is executing, and whether informational messages are to be displayed at the terminal. Because the CONTROL statement can set options for an entire procedure, it is commonly among the first executable statements in a procedure. You may, however, reset the options further along in a command procedure by writing a new CONTROL statement with different operands.

The format for the CONTROL statement is as follows:

```
control [optionl ... optionN]
```

All but two of the CONTROL statement's options occur in mutually-exclusive pairs. When selecting options, you should choose only one from the pair. The options, described in the following topics, are:

```
MSG/NOMSG
PROMPT/NOPROMPT
LIST/NOLIST
CONLIST/NOCONLIST
SYMLIST/NOSYMLIST
FLUSH/NOFLUSH
MAIN
END(string)
```

Command procedures without CONTROL statements will execute with these predefined options: MSG, NOLIST, NOPROMPT, NOCONLIST, NOSYMLIST, and FLUSH. The user may preset the PROMPT and LIST options by entering them as keywords on the EXEC command with which he invokes the procedure; for example:

```
exec myproc.clist prompt list
```

There are, however, no default operands on the CONTROL statement. Entering CONTROL with no operands causes TSO to display only those options already in effect because they are the predefined set or because of a previous CONTROL statement with operands.

Note: TSO's symbolic substitution routine does not scan the CONTROL statement. As a result, CONTROL options may not be in the form of symbolic variables.

Setting the Message Option

You can request that informational messages from commands or statements in the procedure be displayed or suppressed. To request message display, code:

```
control msg
```

To suppress this display, code:

```
control nomsg
```

TSO has predefined the MSG option for command procedures. If you want your command procedure to display informational messages, it is unnecessary for you to specify the MSG option unless you are overriding a previous NOMSG specification in the same procedure. Note that this option has no effect on error messages.

Setting the Prompt Option

You can permit a command procedure to prompt the terminal user for input, or you can prohibit the procedure from doing so, if it ordinarily has prompting capability. To permit prompting, code:

```
control prompt
```

To prohibit prompting, code:

```
control noprompt
```

TSO provides no predefinition for the PROMPT option. You must explicitly set the prompting environment for any command procedure with prompting capability by including a CONTROL PROMPT statement as shown above. A user can request the PROMPT option by specifying the PROMPT keyword on the EXEC command when he invokes your procedure.

Setting the Display Options

You can permit or suppress the display at your terminal of three different categories: TSO commands and subcommands after symbolic substitution but before execution; command procedure statements after symbolic substitution but before execution; TSO commands and subcommands and command procedure statements before symbolic substitution.

To permit the display of TSO commands and subcommands after symbolic substitution, code:

```
control list
```

To suppress their display, code:

```
control nolist
```

TSO provides no predefinition for the LIST option. If you want your command procedure to display its commands and subcommands at the terminal prior to execution, it is necessary to include a CONTROL LIST statement as shown above. A user of a command procedure can request the LIST option by specifying the LIST keyword on the EXEC command when he invokes the procedure.

To permit the display of command procedure statements at the terminal after symbolic substitution but before execution, code:

```
control conlist
```

To suppress their display, code:

```
control noconlist
```

TSO has predefined the NOCONLIST option for command procedures. It is unnecessary for you to explicitly suppress the display of command procedure statements unless you are overriding a previous CONLIST specification in the same procedure.

To permit the display at your terminal of the TSO commands and subcommands and command procedure statements before symbolic substitution, code:

```
control symlist
```

To suppress their display, code:

```
control nosymlist
```

TSO has predefined the NOSYMLIST option for command procedures. It is unnecessary for you to explicitly suppress the display of your command procedure's executable statements unless you are overriding a previous SYMLIST specification in the same procedure.

Setting the Input Stack Flushing Options

The input stack is a queue that indicates the source of TSO's next input. This source can be a terminal or command procedure. When an execution error occurs, TSO purges this queue (flushes the stack) and gets its next input from the terminal.

You should prevent TSO from flushing the stack if your command procedure has an error exit that processes non-zero return codes. In this case, you would not want to flush the stack so that your command procedure could continue processing.

There are two ways to control stack flushing: by specifically requesting or suppressing stack flushing or by designating a command procedure as the main procedure in the invoker's TSO environment.

To specifically prohibit stack flushing, code:

```
control noflush
```

To request stack flushing, code:

```
control flush
```

TSO has predefined the FLUSH option for command procedures. You must use the CONTROL NOFLUSH (or CONTROL MAIN as described below) statement in any command procedure that has an error exit. However, for command procedures where you wish TSO to perform stack flushing normally, it is unnecessary to explicitly specify the FLUSH option except to override a previous CONTROL statement that specified the NOFLUSH option.

You can prevent your command procedure from being deleted by stack flushing requests from the system by designating it as the main procedure in the invoker's TSO environment. The MAIN option provides the same function as the NOFLUSH option previously described and also prevents the attention exit in TSO's terminal monitor program from deleting the command procedure. To designate the MAIN option, code:

```
control main
```

Because the MAIN option performs the same function as the NOFLUSH option, TSO ignores either FLUSH or NOFLUSH on a CONTROL statement that also specifies the MAIN option.

Substituting a String for an END Delimiter

You can specify a character string to be used in place of the normal "END" statement to denote the conclusion of DO-groups. This provides a way for TSO to distinguish between END commands or subcommands you may wish to place within DO-groups and the end of the DO-group itself. To substitute another delimiter for END, code:

```
control end(string)
```

The string within the parentheses may be one to four alphameric characters with the first character alphabetic. For example, you could code:

```
control end(stop)
```

In this case, you could use "STOP" to denote the end of DO-groups in a procedure, until overridden by a subsequent CONTROL statement.

Assigning Values to Symbolic Variables

The SET statement assigns values to symbolic variables. The values may be alphabetic or numeric. The SET statement is useful in any situation where you must assign some specific value to a symbolic variable initially or where you need to change some already established value.

A simple use of the SET statement is in a loop of instructions. You can use SET to set the loop control counter initially and also to increase the counter for each execution of the loop.

To use the SET statement, enter it in the form:

```
set symbolic-variable-name = expression
```

The symbolic variable may be a parameter from a PROC statement, a control variable, or a previously undefined symbolic variable. In any case, you may enter the symbolic variable with or without its leading ampersand.

Assigning a Quantity to a Symbolic Variable

To assign a quantity to a symbolic variable, set the variable equal to a number; for example:

```
set n=5
```

After this statement executes, the variable &N will have the value of 5 until you change it, regardless of whether or not it had some previous value.

Another way to assign a numeric value to a symbolic variable is to set the variable equal to some other variable that already has a numeric value. Consider this sequence:

```
set n = 5
set a = &n
```

The second SET statement assigns the value of 5 to &A from the variable &N, whose value you previously set to 5.

You can also set a variable to the result of an arithmetic expression; for example:

```
set b = 4 + 5
```

The variable &B is assigned the value of 9. The arithmetic expression itself could also incorporate variables; for example:

```
set n = 5
set a = &n
set b = 4+5
set c = &b * ( &a * &n)
```

To resolve the fourth expression (that is, &b * (&a * &n)), TSO uses the values assigned to the symbolic variables &B, &A, and &N and performs the indicated arithmetic operations. In this case, variable &C is assigned the value 255.

The expressions on SET statements may also contain built-in functions. (See the preceding topic "Built-In Functions.") Assume an arithmetic expression uses the length of a particular character string as one of its factors. The representation of this character string is symbolic so that it can apply to different character strings (of differing lengths) during the course of command procedure execution. The &LENGTH built-in function is performed on the symbolic variable &STRING in the following SET statement:

```
set price = 5 * &length( &string)
```

The variable &PRICE is assigned a value that is five times the length of the string represented by the variable &STRING.

Assigning a Character String to a Symbolic Variable

To assign a character string to a symbolic variable, substitute the characters directly; for example:

```
set alphabet = abcdefghijklmnopqrstuvwxyz
```

The variable &ALPHABET is assigned a character string value of the letters A through Z.

You may also use symbolic values that already represent character strings to make such assignments:

```
set string = harry
set arnold = &string
```

As a result of these two SET statements, the character string value represented by the variable &ARNOLD is HARRY.

You can concatenate character string values by entering them successively without intervening delimiters or operators. Concatenated strings may also consist of symbolic variables representing character strings. Note the following examples:

```
set combo = alfred
    or
set name1 = al
set name2 = fred
set combo = &name1&name2
```

The resulting value of &COMBO in either case is ALFRED.

Controlling Execution Flow

The execution sequence in a command procedure is controlled by unconditional branching and by conditional statements and commands. Unconditional branching is implemented by GOTO statements. Conditional statements and commands are DO-groups, IF-THEN-ELSE sequences, and WHEN commands. The following topics describe how to use these types of unconditional and conditional execution control techniques.

Unconditional Branching

The GOTO statement causes an unconditional branch within a command procedure. Code the statement in one of the following forms:

```
goto label-name
goto symbolic-variable
```

The label-name must be a valid label within the command procedure. The symbolic-variable is one whose value, after symbolic substitution, is a label within the command procedure.

To use the GOTO statement with a label name, code GOTO followed by one or more delimiters and a label that identifies a command, subcommand, or statement within the procedure; for example:

```
goto thatspot
```

Assuming that the procedure uses a label called THATSPOT, before or after the GOTO statement, TSO branches to the command, subcommand, or statement designated by THATSPOT when the GOTO statement is executed. After the branch, execution continues with the instruction identified by the label.

The target of the GOTO statement may also be a symbolic variable that the procedure has resolved as a label name by symbolic substitution. For example, assume a variable called &ALIAS has been assigned a value of THATSPOT, which is a label in the procedure. You enter:

```
goto &alias
```

When the command procedure executes this GOTO statement, TSO branches to the label THATSPOT, as in the previous example.

Conditional Statements and Commands

Conditional execution is controlled by the following techniques:

- DO-groups -- sets of related instructions
- DO-WHILE-END sequences -- sets of related instructions that execute repeatedly as long as a specified condition exists
- IF-THEN-ELSE sequences -- sets of related instructions that execute only under certain conditions
- WHEN command -- a TSO command that causes action to be taken for certain system return codes

DO-Groups and the DO-WHILE-END Sequence

DO-groups consist of commands, subcommands, and command procedure statements placed between DO and END statements. These groups are to be executed consecutively or not at all, depending on the result of a decision coded previously in the command procedure. Using indentation for instructions between DO and END provides an easily readable structure; for example:

```
target: do
          allocate dataset( &input) file( indata) old
          allocate dataset( &list) file( print)
        end
```

The END statement is required. For every DO statement, there must be a corresponding END statement to denote the end of the DO-group. DO-WHILE-END sequences are DO-groups that execute repeatedly. The number of repetitions depends on conditions established by WHILE. WHILE, therefore, is a loop control operand. Consider the following execution loop:

```
set &counter = 10
do while &counter gt 0
  .
  .
  .
 set counter = &counter - 1
end
```

The variable &COUNTER is a loop counter initially set to a value of 10. WHILE causes a test of the value of this counter each time the command procedure begins to execute the DO-WHILE-END sequence. As long as the value of &COUNTER is greater than zero (the test condition is true), the procedure executes the sequence, whose last instruction decreases the counter's value by one. When the counter's value reaches zero, the test condition is false, the command procedure bypasses the sequence, and resumes processing at the instruction following the END statement.

Entering END Commands or Subcommands within DO-Groups: There are two ways to use END commands or subcommands within DO-groups without denoting the end of the group to TSO.

One way is to use a control option to establish a substitute character string (like "stop") that TSO can recognize as the end of a DO-group rather than the normal string "end". A description of this control option is in the preceding description of "Establishing Processing Options".

Another method permits the conclusion of DO-groups with the normal END statement. To use it, place any necessary END commands that are in DO-groups between DATA and ENDDATA statements; for example:

```
set counter = 10
do while &counter gt 0
  .
  . (command procedure statements)
  .
  data
    edit 'd58dew1.datapak.clist' old
    .
      (EDIT subcommands)
    .
    end
  enddata
  .
  . (more command procedure statements)
  .
end
```

Only TSO commands and subcommands can appear within the DATA and ENDDATA statements. If a command procedure statement is included, TSO attempts to execute it as a TSO command and the result is an error. If a command procedure statement that has the same name as a TSO command or subcommand (for example, END) is included within the DATA and ENDDATA statements, it is executed as a TSO command or subcommand.

Results of Branching into a DO-group: A branch to a labeled statement within a DO-group (via a GOTO statement) results in the execution of the labeled statement and all remaining commands, subcommands, and statements in the group (that is, up to the END statement that denotes the end of the DO-group).

If the DO-group has a WHILE condition in effect, the command procedure checks the condition when execution reaches the END statement and re-executes the entire DO-group each time the WHILE condition is true.

For example, consider this segment:

```
.
.
set n = 1
set ans = 0
.
.
goto mid
.
.
.
do while &n le 2
  mid: set ans = &ans + 2 * &n
       set n = &n +1
end
.
.
```

The result of executing the GOTO statement is as follows:

1. &ANS is set to 2; that is, (0 + 2 * 1).
2. &N is increased from 1 to 2.
3. The END statement is reached, and since &N satisfies the WHILE condition, the DO-group is re-executed.
4. &ANS is set to 6; that is, (2 + 2 * 2).
5. &N is increased from 2 to 3.
6. The END statement is reached and, since the WHILE condition is now false, the DO-group is not executed again.

The IF-THEN-ELSE Sequence

The IF-THEN-ELSE sequence tests a condition or set of conditions, then determines the path for further execution based on the results of the test. The form for using the sequence is:

```
if logical-expression then action
 [else  [action]]
```

The "action" may be any executable command or statement, or a DO-group. A logical expression consists of comparative expressions connected by logical operators.

For every IF statement, there must be a THEN on the same line or a continuation line. The ELSE action, however, is optional. If ELSE is specified, it cannot be on the same line as IF-THEN.

Figures 19 - 21 show representations of the three primary ways of writing IF statements. Figure 19 shows an IF-THEN-ELSE sequence followed by additional instructions outside the sequence.

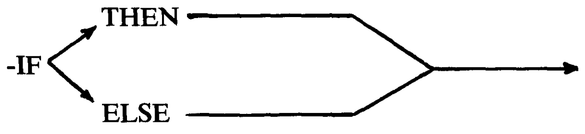


Figure 19. Divergent-Convergent IF-THEN-ELSE Sequence

This IF-THEN-ELSE sequence would be coded as follows:

```
.
.
.
if &a + &b ge &c - &d then do
  .
  . (an action represented by a DO-group)
  .
end
else do
  .
  . (another DO-group, providing ELSE-action)
  .
end
.
. (command procedure instructions following the
    IF-THEN-ELSE sequence)
.
```

In this example, the IF statement tests whether the sum of &A and &B is greater than or equal to the difference of &C and &D. If it is (that is, if the test for conditions is "true"), the THEN-action is executed. Otherwise, the procedure executes the ELSE-action. Execution resumes after either path of processing is complete. Note that THEN is on the same line as IF.

Figure 20 illustrates a type of IF-THEN-ELSE sequence that uses an unconditional branch.

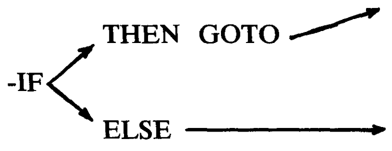


Figure 20. Divergent IF-THEN-ELSE Sequence with an Unconditional Branch

The following example shows this use of IF-THEN-ELSE:

```
.
.
.
if &a = &b then goto label3
else do
  .
  . (DO-group providing an alternative to the
  . unconditional branch)
  .
  end
.
.
label3: ... (a command procedure instruction)
```

In this case, processing takes the unconditional branch only if &A equals &B. Otherwise, the procedure executes the DO-group and the instructions that follow it.

Figure 21 shows an IF statement that provides an action (in the THEN clause) only if the test condition is true. Otherwise, the command procedure executes the instruction following the IF-THEN-ELSE sequence.

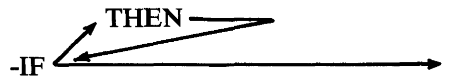


Figure 21. IF Statement without an ELSE Clause

This IF-THEN-ELSE sequence would be coded as follows:

```
.
.
.
if &a = &b then do
  .
  . (DO-group providing an optional action)
  .
  end
.
.
.
```

In this case, the DO-group constituting the THEN clause executes only if & A equals & B. Otherwise, the execution sequence is from the IF statement to the statement following END.

Indenting and Continuing Statement Lines: To help show the structural relationship among the statements in the IF-THEN-ELSE sequence, you should use some indentation scheme similar to the one in the previous examples. This indentation makes your command procedures easier to read.

You can continue an IF statement to successive lines to accommodate lengthy series of test conditions. To continue a line, enter a plus sign or a hyphen at the end of the line you wish to continue. Then continue the statement on the next line, for as many lines as necessary, like this:

```
if &alpha ng &beta and &beta = &gamma and-
  &delta le &epsilon and &epsilon ne &beta and +
  &theta ge &iota and &iota eq &alpha +
then . . .
```

This example illustrates an indentation pattern and the use of both kinds of continuation characters. Note the exact placement of the plus sign. There is at least one blank between it and the variable preceding it. This blank is necessary to keep TSO from parsing the end of the second line and the beginning of the third as AND & THETA, which would make the logical operator "AND" in the third line unrecognizable.

The WHEN Command

The WHEN command tests the return codes from programs that a command procedure has invoked by the CALL or LOADGO command. When the result of the test is true, the command or subcommand specified on the WHEN command is executed. When the result is false, execution proceeds with the next sequential command procedure instruction. In this way, the WHEN command allows you to insert checkpoints into a command procedure. These checkpoints can, for example, cause the procedure to bypass certain processing when program errors make normal execution unnecessary.

To use the WHEN command, specify its system return code (SYSRC) operand, a comparison operator, a numeric value representing a return code, and a command.

```
when sysrc(operator integer) command
```

The following example shows how to code the WHEN command. It illustrates a command procedure that allocates three data sets, invokes a program that uses them, and conditionally invokes an alternate procedure named "checkout" if the return code from the called program equals 8:

```
allocate dataset(input) file(indata) old
allocate dataset(output) block(100) space(300,10)
allocate dataset(list) file(print)
call weekly
when sysrc(= 8) exec checkout
```

Another example could end the preceding procedure if "weekly" returns an error code equal to or greater than 12:

```
when sysrc(ge 12) end
```

The END command is the default command for the WHEN command. If you do not specify any other command after the condition, TSO assumes that you wish to terminate the command procedure.

The WHEN command will work only in the TSO command environment. You cannot use it successfully in any command procedure operating in the subcommand environment. You must also ensure that procedures executed as nested procedures are invoked in the command environment, if they contain WHEN commands. (Do not invoke a nested procedure containing WHEN from a procedure that is using EDIT EXEC, even though the primary command procedure was invoked by the EXEC command.) The WHEN command does not support the use of command procedure statements in place of TSO commands after the statement of return code test conditions. Note that the IF-THEN-ELSE sequence can make any test that the WHEN command can make without being subject to the foregoing restrictions. The IF-THEN-ELSE sequence can operate in the subcommand environment and make use of command procedure statements as well as the TSO command language.

Communicating with the Terminal User

WRITE, WRITENR, TERMIN, READ, and READDVAL statements provide an interactive link between the command procedure and the terminal user.

WRITE and WRITENR statements issue messages to the terminal user, perhaps to tell him why he received control and to prompt him about what he is supposed to do. The TERMIN statement causes a command procedure to pass control to the terminal user. This statement temporarily suspends procedure execution and also defines the character strings the terminal user may enter to return control to the command procedure. TERMIN statements commonly follow WRITE or WRITENR statements. READ and READDVAL statements read the user's entries into designated areas within the command procedure.

Writing Messages to the Terminal User

Two command procedure statements are available for sending messages from a command procedure to the terminal:

- WRITE -- displays a message at the terminal and causes the terminal's display cursor to return to the beginning of a new line after the message finishes displaying.
- WRITENR -- displays a message at the terminal and causes the terminal's display cursor to remain where the message stopped displaying.

To send the messages, use either statement and follow it by one or more blanks and the text of the message; for example:

```
write your previous entry was invalid
write do you wish to retry your previous entry?
writenr enter yes or no.
```

As a result of these statements, the terminal user sees the following messages at his terminal:

```
YOUR PREVIOUS ENTRY WAS INVALID
DO YOU WISH TO RETRY YOUR PREVIOUS ENTRY?
ENTER YES OR NO.
```

The cursor stops after "NO" in the last line to indicate the procedure is waiting for the user's response.

Requesting Terminal Input

To change the current source of TSO's input from your command procedure to the terminal, code a TERMIN statement in your procedure in the following form:

```
termin [string1] [,string2,...]
```

The character strings (STRING1 and so on) are called delimiters and will cause control to return to the command procedure when the terminal user enters any one of them.

As soon as TSO executes the TERMIN statement, the terminal user receives control. The user might or might not receive a mode message after a TERMIN statement is executed. If issued, the mode message might be READY or the name of the command, capable of accepting subcommands, under which the command procedure was invoked. The terminal user can enter TSO commands and subcommands. Normally, control will eventually return to the commmand procedure.

To issue a TERMIN statement that will let the user return control simply by entering a null line, code:

```
termin
```

However, you should be careful about using this form because some TSO command processors use null lines as function delimiters (for example, switching between input and edit modes under EDIT). Make sure your user is aware of the potential problems involved if you code such a statement, perhaps by writing him messages as described in the section "Writing Messages to the Terminal User."

A better way to code the TERMIN statement is to define one or more character strings that will terminate user control; for example:

```
termin stop,error,halt
```

You can also specify a null line as one of the valid entry termination strings, but it must be the first string on the TERMIN statement. To do this, simply precede all the other strings with a comma:

```
termin ,stop,error,halt
```

This statement tells TSO that the user means to return control to the command procedure if he enters either "stop", "error", "halt", or a null line. The same cautions about using a null line as a delimiter in this way apply as they did for using a null line as the only delimiter.

Note: The use of the TERMIN statement during a TEST session that is driven by a command procedure will result in inconsistent operation. It is advisable, in this situation, to use the READ and WRITE statements.

The TERMIN Statement's Effect on Control Variables

Using the TERMIN statement affects the contents of two command procedure control variables:

- &SYSDLM -- contains a number that indicates which character string was entered by the terminal user to return control to the command procedure
- &SYSDVAL -- contains whatever followed the termination delimiter in the line that the user entered to return control to the command procedure

For example, assume a terminal user is in control as a result of this TERMIN statement:

```
termin stop,error,halt
```

Eventually the user enters:

```
halt this entry period now
```

As a result, &SYSDLM has a value of 3 to denote that the user chose "halt", the third termination delimiter defined on the TERMIN statement, to relinquish his control. &SYSDVAL contains "this entry period now." If this user had entered only "halt", &SYSDVAL would have a null value.

Note that there can also be information in &SYSDVAL as a result of a user's response to a READ statement. A description of these contents is in the section "Reading Input from the Terminal."

The contents of &SYSDVAL can consist of meaningful parameter information from the terminal user. (This assumes that the user knows enough from prompting messages or other documentation to enter the kind of information expected.) For example, the user could reply to the previous TERMIN statement with:

```
stop dsname=mydata
```

In this case, the contents of &SYSDVAL become "dsname=mydata", which your command procedure could use as a parameter to influence further processing.

Reading Input from the Terminal

The READ and READDVAL statements provide two ways for command procedures to access user input from the terminal. The READ statement takes information directly from terminal input lines. The READDVAL statement obtains information from the &SYSDVAL control variable. The information in &SYSDVAL is a result of a previous READ statement or of the terminal user entering more than just a TERMIN delimiter in the line with which he returns control to the command procedure. A command procedure may also explicitly place a value into &SYSDVAL, independent of any terminal user response.

Using the READ Statement

The READ statement makes a line of terminal input available to a command procedure in the form of symbolic variables. You should normally precede a read statement with one or more WRITE statements to let the user know that the command procedure is expecting a line of input, and what sort of input it is expecting.

Normally, READ specifies one or more symbolic variables to receive information that the user will enter. If no symbolic variables are specified on READ, the information that the user enters is placed in the control variable &SYSDVAL. Read is entered in the form:

```
read [parm1,parm2...]
```

The READ parameters (&PARM1 and so forth) are the symbolic variables whose contents become the positional user entries in response to the READ statement. Although unnecessary on the READ statement itself, ampersands must precede the parameter names elsewhere in the procedure, to indicate that they are symbolic variables. Coding these variables on the READ statement defines them so that it is unnecessary to define them elsewhere in the procedure.

Assume that a message is sent to the terminal user requesting that he enter four names. The READ statement to read these names could be as follows:

```
read a,b,c,d
```

The user's response to this READ statement could be:

```
smith,jones,kelly,ingalls,greene
```

These names are assigned to the symbolic variables on the READ statement as follows:

```
&a has a value SMITH
&b has a value JONES
&c has a value KELLY
&d has a value INGALLS
```

Because there is not a fifth parameter on the READ statement, the fifth name (GREENE) is superfluous information, which the procedure ignores.

READ can also be used without any variables; for example:

```
read
```

If the user responded with the same five names, they would all be assigned to the control variable &SYSDVAL.

The user's response may be a character string, a character string enclosed in quotes, or a character string enclosed in parentheses.

If the user wants to omit one value from a requested succession, he may use a double comma or a double apostrophe to denote the omitted value. For example, assume that a message is sent to the terminal user requesting that he enter four successive alphabetic characters. The READ statement to read these characters could be:

```
read a,b,c,d
```

The user could respond:

```
m,n,,p
or
'm' 'n' '' 'p'
```

The symbolic variables on the READ statement would then have the following values:

```
&a has the value m
&b has the value n
&c has a null value
&d has the value p
```

You can also use the READ statement to obtain values to assign to PROC statement keywords that do not already have values assigned (see the section "Using the PROC Statement.") Suppose a PROC statement defines &ALPHA as a keyword, but the user does not assign a value to it when he invokes the procedure. The procedure could check for a value assignment and, if it found none, could send a message to the terminal user requesting that he enter a value for &ALPHA and then issue a READ statement with &ALPHA as a parameter.

Using the READDVAL Statement

The READDVAL statement accesses the contents of the &SYSDVAL control variable. It is therefore unnecessary to immediately precede the READDVAL statement with a WRITE statement that requests the terminal user to enter information (he has entered the information already).

The contents of &SYSDVAL result from a terminal user's response to two types of requests:

- A READ statement without operands
- A TERMIN statement (the terminal user entered additional information after the delimiter)

The preceding topics "Using the READ Statement" and "The TERMIN Statement's Effect on Control Variables" explain these two ways that terminal entry affects the contents of &SYSDVAL.

To use READDVAL, write it in the form:

```
readdval parm1 [,parm2,parm3...]
```

The READDVAL parameters (&PARM1 and so forth) are symbolic variables that relate positionally to the character strings in &SYSDVAL. If there is no parameter on the READDVAL statement, no operation takes place. Although unnecessary on the READDVAL statement itself, ampersands must precede the parameter names elsewhere in the procedure, to indicate that they are symbolic variables.

Suppose the following names are in &SYSDVAL:

```
smith,jones,kelly
```

The following statement assigns each name to each specified symbolic variable, in order, from left to right:

```
readdval name1,name2,name3
```

The value of &NAME1 is SMITH, &NAME2 is JONES, and &NAME3 is KELLY.

The following statement, however, assigns values only to the names "smith" and "jones":

```
readdval name1,name2
```

The name "kelly" remains unaccessed. The following statement also reads all three names from &SYSDVAL:

```
readdval name1,name2,name3,name4
```

However, the value of &NAME4 is null because there are not enough character strings in &SYSDVAL to provide a fourth value.

Performing File Input/Output

Four statements are available to command procedures for opening, closing, and accessing the QSAM data sets that command procedures use for file I/O:

- OPENFILE opens a file previously allocated, either by the TSO ALLOCATE command or by step allocation, for input, output, or updating.
- GETFILE reads a record from an open file.
- PUTFILE writes a record to an open file.
- CLOSFILE closes a previously opened file.

To use any of the file accessing statements, enter the statement and the name of the file you want to use. The same name is used several times during file I/O processing. The ddname specified on the ALLOCATE command (for example, INDATA) must also be specified as a symbolic variable when you open and close the file (for example, &INDATA). To read a record from a file or write a record to a file, you must also specify the filename as a symbolic variable (for example, & INDATA). For read requests, the symbolic variable will contain the record read from the file. For write requests, the symbolic variable must contain the record to be written to the file.

Opening a File

To open a file, code:

```
openfile filename operand
```

The filename is a ddname, specified as a symbolic variable, that is already allocated for your terminal session. The operand may be INPUT, OUTPUT, or UPDATE; INPUT is the default. The UPDATE operand indicates that the file can be updated in place; that is, a record written to the data set replaces the previously read record.

Assume that you require three files: an input file, an output file, and a work file. To open them you can code:

```
openfile myinput
openfile myoutput output
openfile workfile update
```

Note that the file MYINPUT defaults to an input file, and that the filenames are symbolic variables.

Reading a Record from an Open File

To read a record from an open file, enter a statement in the form:

```
getfile filename
```

The filename is the ddname, specified as a symbolic variable, by which you allocated and opened the file for your terminal session. The symbolic variable will contain the record retrieved from the file as a result of the statement's execution.

Using the file MYINPUT from the previous example, you can read a record from the input file by coding:

```
getfile myinput
```

After execution of that statement, the value represented by the symbolic variable MYINPUT is the contents of the record obtained from the input file.

Writing a Record to an Open File

To write a record to a file, enter a statement in the form:

```
putfile filename
```

The filename is the ddname, specified as a symbolic variable, by which you allocated and opened the file for your terminal session. You must first set the value of the symbolic variable equal to the contents of the record you wish to place in the output file. Use the SET statement and the name of your output file; for example:

```
set &myoutput = this is my output record
putfile myoutput
```

You must use an assignment statement and the PUTFILE statement in pairs as shown for each record you wish to write, unless you want the same record written more than once.

Closing an Open File

To close an open file, write a statement in the form:

```
closfile filename
```

This process is merely the reverse of opening a file. To close the three files opened in the example in the section "OPENFILE-Opening a File," code:

```
closfile myinput
closfile myoutput
closfile workfile
```

Note that it is not necessary to specify the type of file (input, output, or update) when you close a file.

Executing Nested Command Procedures

A command procedure may invoke another command procedure, which in turn may invoke another, and so forth. Such "nested procedures" are useful for performing hierarchical functions. You can structure a series of nested levels of command procedures in the same way that you can design complex programs with main routines and subroutines arranged in a functionally logically structure.

The command procedure invoked by a user with an EXEC command or subcommand is the top-level or outer-level procedure in any nested hierarchy. Procedures invoked by the outer-level procedure are considered to be nested within it, and they may have lower-level procedures nested within them. Note that any special options established by a nested procedure are in effect only when that nested procedure is executing. In particular, CONTROL statement options and ATTN exits are no longer in effect after a nested procedure returns to the procedure that called it.

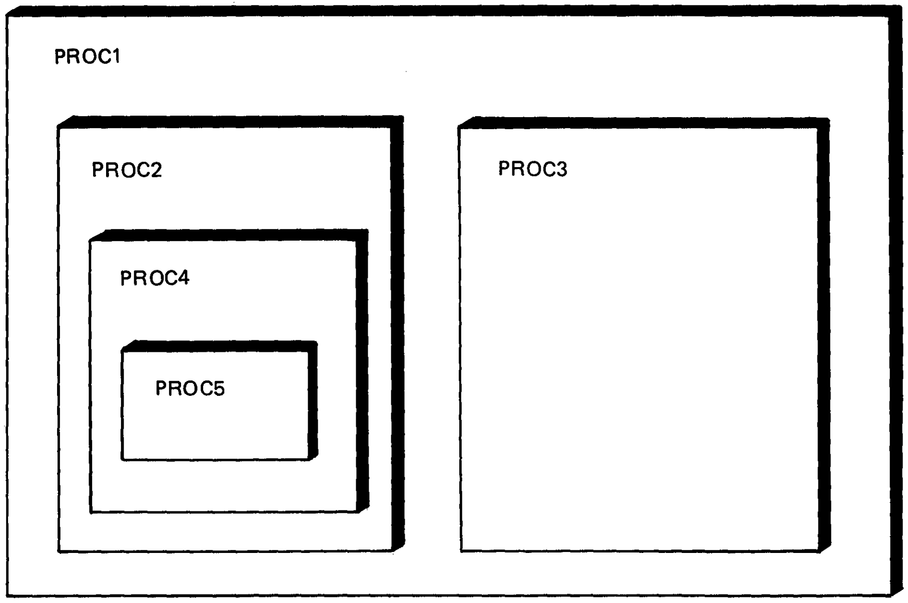


Figure 22. Nested Command Procedures

In Figure 22, PROC1 is the outer-level procedure. It invokes PROC2 and PROC3, which are therefore nested within it. PROC2 invokes PROC4, and PROC4 invokes PROC5. PROC4 is therefore nested within PROC2, and PROC5 within PROC4.

Note that an outer-level procedure with two or more nested procedures within it must execute each procedure sequentially. For example, in Figure 22, if PROC1 invokes PROC2 before PROC3, then PROC4 and PROC5 must execute in nested order before PROC1 may invoke PROC3. Copies of the same procedure can be at two or more levels of a nested hierarchy. This is because each invocation of a nested procedure causes a new copy of that procedure to be brought into storage.

Nested procedures in the subcommand environment may execute only subcommands and command procedure statements. They may not contain TSO commands. A procedure at any given level determines the execution environment--command or subcommand--for the procedures nested at all levels beneath it.

Although the nesting of command procedures provides programming flexibility, you should evaluate whether a compiled program might be more efficient for sophisticated implementations of logic. For example, a PL/I program might be more efficient for a particular application than a complex command procedure.

Establishing Global Symbolic Variables

Global symbolic variables are variables whose contents are set and referenced by both an outer-level command procedure and one or more of its nested procedures.

To establish global symbolic variables, you must first determine the total number of symbolic variables that must be defined as global. Then code a GLOBAL statement in each of the command procedures. The order of the symbolic variables on the outer-level command procedure's GLOBAL statement determines the order in which they must appear on the other GLOBAL statements. That is, they are position-dependent.

For example, assume that command procedure PROC1 has two nested procedures, PROC2 and PROC3. Assume that PROC2 references three variables that are also referenced in PROC1, and PROC3 references those same three variables plus one additional variable that is referenced in PROC1. In this case, PROC1 must have a GLOBAL statement specifying four symbolic variables:

```
global var1 var2 var3 var4
```

If var1, var2, and var3 are the symbolic variables that are also referenced by PROC2, then PROC2 must have a GLOBAL statement specifying those three symbolic variables:

```
global var1 var2 var3
```

Note that since PROC2 does not reference var4, it does not have to specify it on the GLOBAL statement.

PROC3 must specify all four symbolic variables on its GLOBAL statement since it uses all four:

```
global var1 var2 var3 var4
```

Each of these three procedures could now reference the contents of the global symbolic variables and data could be passed from one procedure to another.

In a slightly different situation, assume that PROC3 only uses three of the symbolic variables from PROC1, but they are var2, var3, and var4. The GLOBAL statement in PROC3 must still specify a name for var1 because the variables are position-dependent. However, it can be a "dummy" name because it is just a place holder. For example:

```
global dummy var2 var3 var4
```

In all of the preceding examples, the same symbolic variable names were used on each of the GLOBAL statements. This is not necessary, however, because the symbolic variables are position-dependent. For example, you could specify in PROC1:

```
global var1 var2 var3 var4
```

In PROC2, code:

```
global name1 name2 name3
```

In PROC3, code:

```
global dummy sym1 sym2 sym3
```

TSO associates var1 with name1 and dummy; var2 with name2 and sym1; var3 with name3 and sym2; and var4 with sym3.

Exiting from a Nested Command Procedure

Three ways to exit from a command procedure are to:

- Exit via an EXIT statement
- Let the control return to the calling procedure at the end of the nested procedure
- Issue the END command

When you use the EXIT statement to exit from a command procedure, you can optionally specify a return code to be set in &LASTCC. You can also specify that control is to be passed back up through the nested hierarchy as many levels as necessary to reach a level protected from stack flushing by either a MAIN or NOFLUSH control option. (See the topic "Setting the Input Stack Flushing Options.")

To cause a nested procedure to return to the procedure from which it received control (one level upward), without specifying a return code, code:

```
exit
```

To return a code when you exit, code:

```
exit code(expression)
```

The "expression" may be any expression whose result is numeric.

TSO puts the value specified by the EXIT statement's code expression into the control variables &LASTCC. In this way, lower-level procedures can pass back indications of errors encountered during execution.

The following examples show some valid exit code expressions.

1) exit code(4)

2) set &a=4

   exit code(&a)

3) set &b=8

   exit code(&b+4)

4) set &c=12

   exit code(&c-8)

If you write EXIT without the CODE keyword, the value of &LASTCC is not changed.

To return control to a level protected from stack flushing by either a MAIN or NOFLUSH control option, code:

```
exit quit
or
exit code(expression) quit
```

If no procedure in the nested hierarchy has either a MAIN or NOFLUSH option in effect, coding QUIT causes TSO to flush all current command procedures from the stack and to assume control. When this happens, TSO writes a READY message at the terminal.

Establishing Exit Routines

The ERROR and ATTN statements provide ways for a command procedure to define an action that will occur if the command procedure receives, respectively, a non-zero return code or an attention interruption.

Ordinarily, you would code statements for these exit actions near the beginning of a command procedure and let them apply to the entire procedure. You may optionally cancel the actions at any point, letting the procedure continue without any special exit processing or initializing new exit actions with subsequent ERROR or ATTN statements.

You may initialize new exit actions as many times as you wish. Each respecification overrides all previous specifications.

Error Exits

To create an error exit, code:

```
error action
```

The operand "action" is any executable statement. It is often a DO-group that performs some operation specifically tailored to the expected errors.

For example, a command procedure could specify:

```
error   do
          .
          . (statements constituting
          . an error exit routine)
      end
```

The statements within the DO-group starting on the ERROR statement will execute when the procedure encounters a non-zero return code.

Listing Statements In Error - You can also specify the error statement without an action operand. In this case when the procedure encounters a non-zero return code, TSO lists the statement that received an error and explanatory error messages. If possible, command procedure execution continues with the next statement after the erroneous one.

Protecting the Error Exit - If you use an error exit, you must prohibit TSO from flushing the input stack for any reason. Stack flushing removes the procedure from the system, thereby making error return codes unavailable to a command procedure. This action nullifies the use of an ERROR statement. To prevent stack flushing, use either the MAIN or the NOFLUSH operands on the CONTROL statement, as described in the section "Controlling Command Procedure Options."

Cancelling the Error Exit - To cancel an error exit, code:

```
error off
```

This entry nullifies any previous error action specification, and the procedure continues without any special error processing.

Using Control Variables in an Error Exit

This example illustrates the use of control variables in an error exit routine that checks for various errors.

```
error -
do
    if &lastcc ge 300 && &lastcc le 999 then -
            do
                /*procedure execution error codes */
            end
        if &syspcmd = free then -
            return /* ignore free command errors */
        if &syspcmd = edit then -
            do
                if &sysscmd = submit then -
                    do
                     write error submitting data set
                    end
                .
                .
                .
            end
        if &maxcc gt 63 then -
            exit code( &maxcc )
        return
    end
if &sysnest = yes then -
    do
        write this procedure may not be nested
        exit code( 99 )
    end
```

Attention Exits

To create an attention exit, code:

```
attn action
```

The operand "action" is any executable statement, commonly a DO-group that constitutes an attention exit routine or some other special processing that will take place when an attention interruption occurs during command procedure processing.

For example, a command procedure could issue:

```
attn do
    .
    . (Statements constituting an attention exit
    . routine)
end
```

In this case, the DO-group provides the attention exit processing.

Canceling the Attention Exit - To cancel an attention exit, code:

```
attn off
```

This entry nullifies any previous attention action specification, and the procedure continues without any special attention exit processing.

Note that OFF is a default. If you code ATTN with no other action specification, TSO assumes that you want any attention exit canceled.

Returning Control from an Attention or Error Exit

To return control from an exit routine, code:

```
return
```

The RETURN statement has no operands. It returns control to the command procedure statement, command, or subcommand following the one that either produced the error or received the attention interruption.

The command procedure must issue at least one TSO command or subcommand (a null command is sufficient) prior to the RETURN statement in an attention exit. The following example shows an attention exit that issues a TSO command before issuing the RETURN statement.

```
attn +
do
    set &cmd =    /* default to null */
    write attention exit is in control
    if &oktoterminate = yes then +
        do
           write do you want to terminate? (yes or no)
           read &ans
           if &ans = yes then +
           set &cmd = end
        end
    else +
        write ignoring your attention
        &cmd    /* the tso command */
    return
end
```

Command Procedure Examples

The following command procedures illustrate many command procedure coding techniques, and use many of the facilities available to writers of command procedures.

The first procedure, "PIZZA", determines each person's share of the tab after a group trip to the local pizza emporium. It uses symbolic variables to allow for various individual consumptions of pizza slices and a variety of beverages. Although it is short, it illustrates a relatively complex degree of the nesting of symbolic variables.

The second example consists of a series of command procedures that show how to use command procedures to do programming work. This series consists of:

- A sample terminal session that uses the PROFILE command procedure
- The PROF command procedure
- The SETUP command procedure
- The PROFILE command procedure
- The PRINTA command procedure
- The JCL data set edited by PRINTA

Throughout the series, the userid 'D95MRT2' represents a programmer responsible for maintaining a library of general purpose command procedures. 'D95PGMR' is the userid of a typical user.

The sample PROFILE session shows how the user generated two jobs to list two of his data sets by using the PRINTA procedure.

The PROF and SETUP command procedures are maintained by the typical user in a PDS named 'D95PGMR.CLIST' to make his job of using the general purpose procedures easier. In the sense that these procedures are primarily for the typical user's own use, they are 'private' procedures.

The PROFILE and PRINTA command procedures are the general purpose procedures maintained by the other programmer in the PDS named 'D95MRT2.CLIST'. These procedures are for widespread and general use by anyone who needs them.

The JCL data set edited by PRINTA is a model that provides the basic JCL statements necessary to set up for and execute the IEBPTPCH utility program, which prints the typical user's jobs as shown in the sample PROFILE session.

The SETUP command procedure provides information that is unique to the user and his immediate purpose. The user can create multiple setup members with different parameters, enabling him to specify in advance the setup for different applications.

The PROF command procedure contains an EXEC command that invokes PROFILE and a PROC statement that uses a keyword to define a default setup member. These features enable the user to initiate a PROFILE session with a minimum of terminal entry. That is, the user need not enter the fully qualified name of PROFILE, nor explicitly specify a particular setup member in order to initiate the session. The following entry is all that is necessary to meet minimum requirements by using a default setup member:

```
exec (prof)
```

In the example PROF command procedure that follows, the default setup member name is 'SETUP2.' In the remaining example material, this default is overridden by the use of a setup member named 'SETUP.'

The PROFILE command procedure provides the environment in which other procedures operate by using the globally known information supplied through the user's SETUP member, which PROFILE reads by using file I/O statements, instead of executing it as a procedure. PROFILE also allows the user to alter any of his setup parameters for the life of the PROFILE session, effectively creating a 'SETUP' command. PROFILE also changes the last character of the jobnames of the jobs submitted through PRINTA, to give each job submitted during a PROFILE session a unique identifier.

The PRINTA command procedure receives control from PROFILE to prepare the user's jobs for submission according to the setup criteria established in the user's SETUP member. PRINTA uses these criteria to edit the model JCL appropriately for a given job to be submitted, using the IEBPTPCH utility to print the user's data set.

The third example illustrates the use of &DATATYPE, &LENGTH, &STR, and &SUBSTR built-in functions. It also shows a technique for using a left parenthesis as data.

Example 1

PIZZA.CLIST (Part 1 of 2)

```
PROC 0 SLICE(33) SODA(35) COFFEE(25) MILK(30)
WRITE SLICES        1      2      3      4      5      6      7      8      9     10
WRITE
SET RETNUM=1
SET HEAD1=NO DRINK
SET COST1=0
SET HEAD2=1 SODA
SET COST2=&SODA
SET HEAD3=2 SODAS
SET COST3=&SODA*2
SET HEAD4=3 SODAS
SET COST4=&SODA*3
SET HEAD5=1 COFFEE
SET COST5=&COFFEE
SET HEAD6=2 COFFEES
SET COST6=&COFFEE*2
SET HEAD7=3 COFFEES
SET COST7=&COFFEE*3
SET HEAD8=1 MILK
SET COST8=&MILK
SET HEAD9=2 MILKS
SET COST9=&MILK*2
DO WHILE &RETNUM LE 9
  SET HEAD=&&HEAD         /* HEAD = '&HEAD'              */
  SET COST=&&COST         /* COST = '&COST'              */
  SET HEAD=&HEAD&RETNUM   /* HEAD = '&HEAD1','&HEAD2'... */
  SET COST=&COST&RETNUM   /* COST = '&COST1','&COST2'... */
  GOTO ROUTINE
RETURN: +
END
WRITE
WRITE
WRITE
WRITE THE ABOVE PRICES APPLY TO:
WRITE   SLICE OF PIZZA .&SLICE
WRITE   A SODA         .&SODA
WRITE   A COFFEE       .&COFFEE
WRITE   A MILK         .&MILK
WRITE
WRITE A TIP OF 15% ON THE COST OF THE PIZZA IS ALSO INCLUDED
WRITE
WRITE FOR OTHER DRINK ITEMS, FIGURE RESULTS SEPARATELY
EXIT CODE(0)
ROUTINE: +
SET RETNUM=&RETNUM+1
WRITENR &SUBSTR(1:10,&HEAD           )
SET N=1
```

PIZZA.CLIST. (Part 2 of 2)

```
DO WHILE &N LE 10
  /* CALCULATE A 15% TIP TO THE NEAREST CENT */
  SET AMT=&N*&SLICE+&COST+(&N*&SLICE*15)/100
  IF (&N*&SLICE*15)//100 >= 50 THEN +
    SET AMT=&AMT+1  /* ROUND TO NEAREST CENT */


  /* FOLLOWING ROUTINE ROUNDS EACH PERSON'S BILL TO THE */
  /* NEAREST NICKEL. (IF CENTS=1,2,6 OR 7 ROUND DOWN;   */
  /* IF CENTS=3,4,8 OR 9 ROUND UP)                      */

  SET AMT=&STR(   &AMT)  /* CONVERT NUMBER TO STRING */
  SET AMT1=&SUBSTR(1:&LENGTH(&STR(&AMT))-2,&AMT)
  SET AMT1=&AMT1  /* NUMERIC VALUE OF DOLLARS PORTION */
  SET AMT2=&SUBSTR(&LENGTH(&STR(&AMT))-1,&AMT) /* DIMES DIGIT */
  IF &SUBSTR(&LENGTH(&STR(&AMT)),&AMT) LT 8 THEN +
    DO   /* IF CENTS DIGIT IS LESS THAN 8 */
      SET AMT2=&AMT2  /* CURRENT DIMES DIGIT IS OK */
      IF &SUBSTR(&LENGTH(&STR(&AMT)),&AMT) GT 2 THEN +
        SET AMT3=5  /* IF CENTS DIGIT OVER 2, MAKE IT 5 */
      ELSE +
        SET AMT3=0  /* IF CENTS DIGIT 2 OR LESS, MAKE IT 0 */
    END
  ELSE /* (IF CENTS DIGIT IS 8 OR 9) */ +
    DO  /* ROUND TO A VALUE ENDING IN 0 */
      IF &AMT2=9 THEN +
        DO  /* IF 90 CENTS OR MORE */
          SET AMT2=0  /* RESET DIMES DIGIT TO 0 */
          SET AMT1=&AMT1+1  /* BOOST DOLLAR AMOUNT */
        END
      ELSE +
        SET AMT2=&AMT2+1  /* IF 8, JUST BOOST DIMES DIGIT */
      SET AMT3=0  /* SET CENTS DIGIT TO 0 */
    END
  WRITENR &SUBSTR(&LENGTH(&AMT1)+1:&LENGTH(&AMT1)+4,   &AMT1).&AMT2&AMT3
  SET &N=&N+1
END
WRITE
GOTO RETURN
END OF DATA
```

The "PIZZA" command procedure produces output as shown below. This output is the result of executing the procedure using all the coded default values for slice and beverage prices (note that the EXEC command uses no value list to override the defaults). You could override any or all of the default values with prices more in keeping with the charges at your local pizza emporium. When you go out with a group for pizza, you can take the output from this command to provide a handy reference table for determining each person's share of the tab.

```
exec pizza.clist
SLICES        1      2      3      4      5      6      7      8      9      10

NO DRINK     .40    .75   1.15   1.50   1.90   2.30   2.65   3.05   3.40   3.80
1 SODA       .75   1.10   1.50   1.85   2.25   2.65   3.00   3.40   3.75   4.15
2 SODAS     1.10   1.45   1.85   2.20   2.60   3.00   3.35   3.75   4.10   4.50
3 SODAS     1.45   1.80   2.20   2.55   2.95   3.35   3.70   4.10   4.45   4.85
1 COFFEE     .65   1.00   1.40   1.75   2.15   2.55   2.90   3.30   3.65   4.05
2 COFFEES    .90   1.25   1.65   2.00   2.40   2.80   3.15   3.55   3.90   4.30
3 COFFEES   1.15   1.50   1.90   2.25   2.65   3.05   3.40   3.80   4.15   4.55
1 MILK       .70   1.05   1.45   1.80   2.20   2.60   2.95   3.35   3.70   4.10
2 MILKS     1.00   1.35   1.75   2.10   2.50   2.90   3.25   3.65   4.00   4.40



THE ABOVE PRICES APPLY TO:
  SLICE OF PIZZA .33
  A SODA         .35
  A COFFEE       .25
  A MILK         .30

A TIP OF 15% ON THE COST OF THE PIZZA IS ALSO INCLUDED

FOR OTHER DRINK ITEMS, FIGURE RESULTS SEPARATELY
READY
```

Example 2

Sample PROFILE Session

```
      .
      .
      .
     edit 'd95pgmr.clist(private)' clist
     EDIT
     delete 500
     save
     EDIT
     exec (prof) 'mem(setup)'
     RETURN TO COMMAND MODE BEFORE ISSUING PROFILE.
     end
     READY
     exec (prof) 'mem(setup)'
     MACLIB('D95PGMR.MACLIB' 'D95DEPT.MACLIB' 'SYS1.TSO.MACLIB')
     MNO(123456) ACCT(999999) CLASS(J) RGN(250K)
     JOBCHAR(A) BLDG(705) CUBE(1P2)
     READY
     setup jobchar f
     READY
     setup acct 444555
     READY
     setup maclib ('d22dept.maclib' 'sys1.tso.maclib')
     READY
     setup list
     MACLIB('D22DEPT.MACLIB' 'SYS1.TSO.MACLIB')
     MNO(123456) ACCT(444555) CLASS(J) RGN(250K)
     JOBCHAR(F) BLDG(705) CUBE(1P2)
     READY
     printa 'd95pgmr.clist'
     *******************************************************
     * PREPARING JOB D95PGMRF TO BE SUBMITTED AT 09:27:16  *
     * JOB D95PGMRF SUBMITTED AT 09:27:46 ON 11/05/74      *
     *******************************************************
     READY
     printa jobs.cntl lrecl(80) recfm(fb) dsorg(po)
     *******************************************************
     * PREPARING JOB D95PGMRG TO BE SUBMITTED AT 09:28:52  *
     * JOB D95PGMRG SUBMITTED AT 09:29:21 ON 11/05/74      *
     *******************************************************
     setup quit
     END OF PROFILE SESSION - RE-PROFILE TO USE AGAIN
     READY
       .
       .
       .
```

PROF Command Procedure

'D95PGMR.CLIST(PROF)'

This is the highest level procedure that each user will execute to issue the PROFILE command procedure.

```
PROC 0 MEM(SETUP2)
GLOBAL D1 D2 D3 D4 D5 D6 D7 D8 D9 D10 D11 D12
EXEC 'D95MRT2.CLIST(PROFILE)' 'MEM(&MEM)'
```

SETUP Member

'D95PGMR.CLIST(SETUP)'

This member provides the information unique to each user of the PROFILE command procedure. It is processed by PROFILE using the file I/O statements.

```
ȸMACLIB ('D95PGMR.MACLIB' 'D95DEPT.MACLIB')  /* USER'S MACRO-LIBRARY */
ȸMACLIB (&MACLIB 'SYS1.TSO.MACLIB')  /*        CONCATENATION        */
ȸBLDG 705                            /* BUILDING                    */
ȸCUBE 1P2                            /* CUBICLE                     */
ȸCLASS J                             /* CLASS FOR SUBMITTED JOBS    */
ȸACCT 999999                         /* ACCOUNT NUMBER              */
ȸJOBCHAR A      /* LAST CHARACTER OF NEXT SUBMITTED JOB'S NAME      */
ȸRGN 250K       /* REGION SIZE FOR SUBMITTED JOBS                   */
ȸMNO 111111                          /* USER'S MAN NUMBER           */
ȸLISTDEF        /* DISPLAY USER'S SETUP WHEN PROFILE IS ISSUED      */
```

Note: The leading blank is necessary in each line of this data set because of the READDVAL statement used to process the line read by the GETFILE statement. The blank separates the line number that appears at the beginning of each of the data set's variable length records from the text or data that follows, thereby preventing TSO from interpreting the line number as part of the meaningful content of the line.

'D95MRT2.CLIST(PROFILE)' Command Procedure (Part 1 of 4)

```
PROC 0 QUICK MEM(SETUP)
CONTROL MAIN NOMSG
IF &SYSSCMD=EXEC THEN /* IF IN SUBCOMMAND ENVIRONMENT */+
  DO
    WRITE RETURN TO COMMAND MODE BEFORE ISSUING PROFILE.
    EXIT CODE(0)
  END
IF &SYSPROC=PROC01 && &QUICK¬=QUICK THEN /* NOT USING STEPLIB */+
  DO
AL: FREE DA(CLIST 'SYS1.TSO.CLIST' 'D95MRT2.PROCS') FI(SYSPROC)
    ERROR +
      DO
        WRITE RRRR....RING....GGGG
        WRITE CANNOT ALLOCATE FILE SYSPROC FOR YOU
        EXIT CODE(12)
      END
    ALLOC FI(SYSPROC) DA(CLIST 'D95MRT2.PROCS' 'SYS1.TSO.CLIST')
    ERROR OFF
  END
ELSE +
  IF &QUICK¬=QUICK THEN /* NOT USING DEFAULT SYSPROC CONCATENATION */+
    DO
      WRITENR DO YOU WANT TO RE-ALLOC FI(SYSPROC)? (Y OR N)
      READ &ANS
      IF &ANS=Y THEN +
        GOTO AL
    END

/* DEFINE GLOBAL VARIABLES */
GLOBAL ACCT JOBCHAR BLDG CUBE MACLIB RGN
GLOBAL GMNO CLASS GMACS GNEST

/* INITIALIZE VARIABLES    */
SET GMACS=NO
SET &CMD=&STR(&&CMD) /* SET VALUE OF &CMD TO STRING '&CMD' */
SET &CMD1=SETUP /* CMD TO LIST/CHANGE PROFILE ENVIRONMENT */
SET &CMD2=COMPILE /* CMD TO SUBMIT COMPILE JOB */
SET &CMD3=PRINTA /* CMD TO PRINT A PDS (SEE EX. 3) */
SET &CMD4=       /* SLOT FOR FUTURE COMMAND        */
SET &CMD5=       /* SLOT FOR FUTURE COMMAND        */
SET &TAB=&STR(&&TAB)
/* USE READDVAL TO SET &TAB1 TO 'ACCT' ,&TAB2 TO 'JOBCHAR', AND SO ON */
SET &SYSDVAL = ACCT  JOBCHAR BLDG  CUBE   MACLIB
READDVAL       &TAB1 &TAB2   &TAB3 &TAB4  &TAB5
SET &SYSDVAL = RGN   MNO     CLASS LISTDEF QUIT
READDVAL       &TAB6 &TAB7   &TAB8 &TAB9  &TAB10
SET &TABN=10    /* SET LENGTH OF SETUP OPTION TABLE */
SET &ERRSAV=0
```

'D95MRT2.CLIST(PROFILE)' Command Procedure (Part 2 of 4)

```
ERROR +
  DO
    WRITE YOUR SETUP MEMBER CANNOT BE FOUND OR READ.
    EXIT CODE(12)
  END
/* THE USER'S SETUP MEMBER MUST BE IN A DATASET NAMED */
/* 'USERID.CLIST'                                     */
ALLOC FI(SYSDVAL) DA(CLIST(&MEM)) SHR
OPENFILE SYSDVAL /* USE &SYSDVAL AS A FILE VARIABLE */
SET EOF=NO /* END OF FILE HAS NOT OCCURRED */
ERROR OFF /* CANCEL PREVIOUS ERROR EXITS */
SET ERRSAV=0

ERROR +
  IF &LASTCC=400 THEN /* IF END-OF-FILE */ +
    DO
      CLOSFILE SYSDVAL /* CLOSE THE QSAM FILE */
      FREE FI(SYSDVAL)
      SET EOF=YES /* INDICATE END-OF-FILE HAS OCCURRED */
      IF &GNEST=YES THEN /* IF PROFILE HAS BEEN RE-ISSUED */ +
        DO
          WRITE VARIABLES HAVE BEEN RESTORED TO ORIGINAL VALUES.
          EXIT CODE(0)
        END
      SET GNEST=YES /* INDICATE PROFILE HAS BEEN RE-ISSUED */ +
      GOTO TERMIN
    END
  ELSE +
  DO /**************** AN ERROR HAS OCCURRED ***************/
    IF &LASTCC=&ERRSAV THEN /* RECURRENT ERROR? */ +
      DO
        WRITE THE SAME ERROR (&LASTCC) RE-OCCURRED - WANT DEBUG MODE?
        SET &LASTCC=0 /* CLEAR ERROR INDICATION */
        READ &ANS
        IF &ANS=YES THEN /* IF 'YES' TRACE EXECUTION */ +
          CONTROL CONLIST MSG SYMLIST LIST /* SET DEBUG OPTIONS */
        ELSE +
          IF &ANS=Y THEN /* GET ERROR MESSAGES AND CONTINUE */ +
            ERROR
          ELSE /* TURN OFF ALL DEBUGGING OPTIONS */ +
          CONTROL NOCONLIST NOMSG NOSYMLIST NOLIST
      END
    IF &LASTCC>=300 && &LASTCC<=999 THEN +
      SET &ERRSAV=&LASTCC /* SAVE CLIST ERROR CODE */
    IF &LASTCC=12 THEN +
      RETURN /* 12 IS SEVERE ERROR CODE FROM MOST COMMANDS */
  END
GOTO TERMIN
```

'D95MRT2.CLIST(PROFILE)' Command Procedure (Part 3 of 4)

```
TOP: CONTROL NOCONLIST NOSYMLIST NOLIST
SET &ERRSA\ 0
TERMIN: +
 IF &EOF=NO THEN /* IF NOT FINISHED READING USER'S SETUP FILE */+
  DO
    GETFILE SYSDVAL /* READ A LOGICAL RECORD FROM USER'S SETUP CLIST */
    READDVAL A1 A2 A3 /* PARSE RECORD SKIPPING THE LINE NUMBER IN A1 */
    SET SYSDVAL=&STR(&A2 (&A3)) /* SET SECOND OPERAND (A3) IN PARENS */
  END
 ELSE +
 TERMIN &CMD1 &CMD2 &CMD3 &CMD4 &CMD5
 SET &SYSSCAN=16
IF &SYSDLM>1 THEN /* IF THE DELIMITER IS A JOB */ +
  DO
    &CMD&SYSDLM &SYSDVAL     /* EXECUTE THE SELECTED CLIST TO    */
                             /* SUBMIT A PARTICULAR TYPE OF JOB. */
  /*****************************************************************/
  /* INCREMENT THE JOB NAME CHARACTER                              */
  /*****************************************************************/
    SET &A=ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890A
    SET &N=1
    DO WHILE &JOBCHAR¬=&SUBSTR(&N,&A) /* FIND CURRENT JOBCHAR IN STRING */
      SET &N=&N+1
    END
    SET &JOBCHAR=&SUBSTR(&N+1,&A) /* SET JOBCHAR TO NEXT CHAR IN STRING */

    GOTO TOP
  END
ELSE                         /* IF SETUP WAS ENTERED */ +
  DO
    READDVAL &A1 &A2 /* PARSE SETUP OPTION AND ITS VALUE */
    SET &NDX=1
    DO WHILE &NDX<=&TABN     /* NUMBER OF KEYWORDS */
      IF &LENGTH(&A1)<=&LENGTH(&TAB&NDX) THEN/* IF LENGTHS OK */ +
        IF &SUBSTR(1:&LENGTH(&A1),&TAB&NDX)=&A1 THEN /* AND KEY MATCH */ +
          GOTO LBL&NDX         /* GO TO FUNCTION REQUESTED */
        SET &NDX=&NDX
      SET &NDX=&NDX+1        /* INCREMENT INDEX    */
    END
    WRITE UNIDENTIFIABLE KEYWORD &A1 ON SETUP
    GOTO TOP
```

'D95MRT2.CLIST(PROFILE)' Command Procedure (Part 4 of 4)

```
  LBL1: /* SETUP ACCT ROUTINE */+
    SET &ACCT=&A2
    GOTO TOP
  LBL2: /* SETUP JOBCHAR ROUTINE */+
    SET &JOBCHAR=&A2
    GOTO TOP
  LBL3: /* SETUP BLDG (BUILDING) ROUTINE */+
    SET &BLDG=&A2
    GOTO TOP
  LBL4: /* SETUP CUBE (CUBICLE) ROUTINE */+
    SET &CUBE=&A2
    GOTO TOP
  LBL5: /* SETUP MACLIB ROUTINE...A PARENTHESIZED STRING CONTAINING */+
    IF &GMACS=YES THEN     /* FULLY QUALIFIED MACRO LIBRARY NAMES    */+
      DO                   /* SHOULD BE ENTERED.                     */
        FREE DA(&MACLIB) FI(SYSLIB)
       IF &A2= THEN        /* IF NO MACRO LIBRARIES */+
          SET &GMACS=NO
        ELSE +
          ALLOC DA(&A2) FI(SYSLIB)
      END
    SET &MACLIB=&A2
    GOTO TOP
  LBL6: /* SETUP RGN (REGION FOR SUBMITTED JOBS) ROUTINE */+
    SET &RGN=&A2
    GOTO TOP
  LBL7: /* SETUP MNO (MAN/EMPLOYEE NUMBER) ROUTINE */+
    SET &GMNO=&A2
    GOTO TOP
  LBL8: /* SETUP CLASS (JOBCLASS FOR SUBMITTED JOBS) ROUTINE */+
    SET &CLASS=&A2
    GOTO TOP
  LBL9: /* SETUP LIST (LISTDEF) ROUTINE (LISTS PROFILE OPTIONS) */+
    WRITE MACLIB(&MACLIB)
    WRITE MNO(&GMNO) ACCT(&ACCT) CLASS(&CLASS) RGN(&RGN)
    WRITE JOBCHAR(&JOBCHAR) BLDG(&BLDG) CUBE(&CUBE)
    GOTO TOP
  LBL10: /* SETUP QUIT ROUTINE (LEAVES PROFILE ENVIRONMENT) */+
    WRITE END OF PROFILE SESSION - RE-PROFILE TO USE AGAIN
    IF &GMACS=YES THEN +
      FREE DA(&MACLIB) FI(SYSLIB)
    EXIT CODE(0)
END
```

'D95MRT2.CLIST(PRINTA)' Command Procedure (Part 1 of 4)

```
PROC 1 DSNAME                  /* DATA SET NAME TO BE PRINTED          */+
       MEMBR()                 /* A LIST OF MEMBERS TO BE PRINTED      */+
       RECFM()                 /* RECORD FORMAT TO BE USED             */+
       DSORG()                 /* DATA SET ORGANIZATION TO USE         */+
       LRECL()                 /* LOGICAL RECORD LENGTH OF RECORDS     */+
       JOBCHAR()               /* JOB CHARACTER TO BE USED             */+
       BLDG()                  /* BUILDING LOCATION OF USER            */+
       CUBE()                  /* CUBE OF USER                         */+
       COPIES()                /* NUMBER OF COPIES DESIRED             */+
       ACCT()                  /* USERS ACCOUNT NUMBER (REQ FOR BATCH) */+
       CLASS()                 /* JOB CLASS TO BE USED                 */+
       NONUM                   /* DATASET IS NOT LINE NUMBERED         */

ERROR                            /* SET UP AN ERROR EXIT               */+
  DO                             /* DO                                 */
    DATA                         /* FOR TSO END STMT                   */
    END
    ENDATA                       /* END OF LIST OF TSO STMT            */
    WRITE PRINT ERROR OCCURRED - TRY AGAIN (RC=&LASTCC)
    EXIT CODE(99) QUIT
  END
CONTROL NOFLUSH                /* DO NOT ALLOW FLUSHING                */


/*********************************************************************/
/*                                                                   */
/* SETUP REFERENCE TO GLOBAL PARMS THEN DEFAULT PARAMETERS           */
/*                                                                   */
/*********************************************************************/

GLOBAL &GACCT &GJOBCHAR &GBLDG &GCUBE &G5 &G6 &G7 &GCLASS


IF &RECFM=     THEN             /* IF RECORD FORMAT NOT SPECIFIED      */+
  SET &RECFM=V                  /* SET DEFAULT TO VARIABLE             */

DO WHILE &RECFM¬=V && &RECFM¬=VB /* CHECK FOR VALID RECORD FORMATS     */+
      && &RECFM¬=F && &RECFM¬=FB /* OF V,VB,F, AND FB                  */
                                /* IF NOT ANY OF THESE, PROMPT USER    */
  WRITE INVALID RECFM &RECFM - REENTER &STR(-)
  READ &RECFM                   /* READ A NEW RECORD FORMAT AND TRY    */
END                             /* AGAIN                               */

SET &RECFM=&SUBSTR(1,&RECFM)    /* MAKE &RECFM V OR F FOR USE LATER    */

IF &LRECL=     THEN             /* IF LRECL IS NULL, DEFAULT TO PROPER */+
  IF &RECFM=V THEN              /* LRECL FOR RECORD FORMAT             */+
    SET &LRECL=255              /* VARIABLE GETS 255                   */
  ELSE                          /* OTHERWISE                           */+
    SET &LRECL=80               /* FIXED GETS 80                       */
ELSE                            /* IF LRECL HAS A VALUE                */+
  IF &DATATYPE(&LRECL)=CHAR |   /* CHECK IF NOT NUMERIC                */+
    &LRECL<50 THEN              /* OR THEN LRECL TOO SMALL             */+
    SET &LRECL=50               /* DEFAULT LRECL TO AT LEAST 50        */
```

'D95MRT2.CLIST(PRINTA)' Command Procedure (Part 2 of 4)

```
IF &DSORG=    THEN            /* IF DSORG NOT SPECIFIED              */+
  SET &DSORG=PU               /* DEFAULT TO PARTITIONED              */

DO WHILE &DSORG¬=PU && &DSORG¬=PS /* CHECK FOR VALID DSORG           */
  WRITE INVALID DSORG &DSORG - REENTER &STR(-)
  READ &DSORG                 /* GET ANOTHER DSORG AND CHECK IT      */
END                           /*                                     */

/*********************************************************************/
/*                                                                   */
/* BUILD THE REQUIRED VARIABLES                                      */
/*                                                                   */
/*********************************************************************/



SET &TAB1=6ACCT               /* KEYWORD ACCT      SYNTAX LEN=6      */
SET &TAB2=1JOBCHAR            /* KEYWORD JOBCHAR   SYNTAX LEN=1      */
SET &TAB3=0BLDG               /* KEYWORD BLDG      NOSYNTAX          */
SET &TAB4=0CUBE               /* KEYWORD CUBE      NOSYNTAX          */
SET &TAB5=1CLASS              /* KEYWORD CLASS     SYNTAX LEN=1      */
SET &TABN=5                   /* NUMBER OF KEYWORDS TO BE CHECKED    */

SET &A=&STR(&&A)              /* FOR VARIABLE RESCAN                 */
SET &G=&STR(&&G)              /* FOR VARIABLE RESCAN                 */
SET &TAB=&STR(&&TAB)          /* FOR VARIABLE RESCAN                 */
SET &LEN=&STR(&&LEN)          /* FOR VARIABLE RESCAN                 */


SET &COUNT=1                  /* CURRENT COUNT FOR KEYWORD CHECK     */
DO WHILE &COUNT <= &TABN      /* CHECK THIS MANY KEYWORDS            */
  SET &LEN&COUNT=&SUBSTR(1,&TAB&COUNT) /* GET SYNTAX LENGTH          */
  SET &TAB&COUNT=&SUBSTR(2:&LENGTH(&TAB&COUNT),&TAB&COUNT) /* GET NAM*/
  SET A&TAB&COUNT=&&&TAB&COUNT /* SET CHECKER                        */
  IF &A&TAB&COUNT =       THEN/* IF KEYWORD VALUE IS NULL            */+
    IF &G&TAB&COUNT =     THEN/* AND GLOBAL IS NULL ALSO             */+
      DO                     /* PROMPT USER FOR VALUE                */
        WRITE ENTER VALUE FOR &TAB&COUNT &STR(-)
        READ &ANS            /* GET ANSWER FROM PROMPT               */
        SET &&TAB&COUNT = &ANS /* PLACE ANSWER IN PROPER VARIABLE    */
      END                    /* CONTINUE                             */
    ELSE                     /* IF GLOBAL PART NOT NULL              */+
      SET &&TAB&COUNT = &G&TAB&COUNT /* COPY INTO LOCAL VARIABLE     */



  DO WHILE (&LEN&COUNT =0 && &LENGTH(&A&TAB&COUNT)=0)|  +
           (&LEN&COUNT¬=0 && &LENGTH(&A&TAB&COUNT)¬=&LEN&COUNT)
                             /* CHECK VALUE FOR                     */
                             /*   1) NOSYNTAX - NOT NULL            */
                             /*   2) SYNTAX   - EQUAL TO LEN SPECIF */
    WRITE INVALID &TAB&COUNT &A&TAB&COUNT - REENTER &STR(-)
    READ &ANS
    SET &&TAB&COUNT=&ANS     /* PUT ANSWER INTO PROPER VARIABLE     */
  END                        /* AND CHECK ITS SYNTAX                */
  SET &COUNT=&COUNT+1        /* GO TO NEXT KEYWORD                  */
END
```

'D95MRT2.CLIST(PRINTA)' Command Procedure (Part 3 of 4)

```
/**********************************************************************/
/*                                                                    */
/* PREPARE TO EDIT THE MASTER JCL DATA SET                            */
/*                                                                    */
/**********************************************************************/

IF &MEMBR¬=  && &DSORG=PS THEN/* IF MEMBER LIST IS NOT NULL FOR PS   */+
  DO                          /* ISSUE MESSAGE AND SET TO NULL       */
    WRITE MEMBER LIST IGNORED FOR SEQUENTIAL PROCESSING
    SET &MEMBR=               /* CLEAR MEMBER LIST                   */
  END
IF &SUBSTR(1,&DSNAME)=' THEN  /* IF DSNAME FULLY QUALIFIED           */+
  SET &DSNAME=&SUBSTR(2:&LENGTH(&DSNAME)-1,&DSNAME)/* STRIP QUOTES   */
ELSE                          /* OTHERWISE                           */+
  SET &DSNAME=&SYSUID..&DSNAME/* APPEND USERID TO DSNAME             */
IF &COPIES= THEN             /* IF COPIES NOT SPECIFIED              */+
  SET &COPIES=1              /* DEFAULT TO ONE                       */
ELSE                           /* VALIDATE NUMBER OF COPIES SPEC     */+
  DO WHILE &DATATYPE(&COPIES)=CHAR  | /* DO WHILE NOT NUMERIC        */+
       &COPIES<1|&COPIES>16 /* OR OUT OF RANGE                       */
    WRITE INVALID NUMBER OF COPIES &COPIES - REENTER &STR(-)
    READ &COPIES             /* GET A NEW VALUE OF COPIES            */
  END                         /*                                     */
IF &NONUM=NONUM THEN            /* NONUM DATASET                     */+
  DO                            /* PROCESS NONUM                     */
    SET &FLDS=1                 /* ONLY ONE FLD IN THIS RECORD       */
    IF &LRECL>100 THEN          /* IF LRECL TOO BIG                  */+
      SET &SIZE=130             /* DEFAULT TO 130                    */
    ELSE                        /* OTHERWISE                         */+
      SET &SIZE=&LRECL          /* SET TO LRECL                      */
    SET &RECORD=&STR(RECORD FIELD=(&SIZE))
  END
ELSE                            /* OTHERWISE                         */+
  DO                            /* PROCESS NUMBERED DATASET          */
    SET &FLDS=2                 /* TWO FIELDS PER RECORD STMT        */
    IF &LRECL 128 THEN          /* IF LRECL TOO BIG                  */+
      SET &SIZE=120             /* DEFAULT SIZE                      */
    ELSE                        /* OTHERWISE                         */+
      SET &SIZE=&LRECL-8        /* SET SIZE                          */
    IF &RECFM=F THEN           /* FOR RECORD FORMAT FIXED            */+
      DO                       /*                                    */
        SET &LOCA=&LRECL-7     /* START COLUMN FOR SEQ NUMBER        */
        SET &LOCB=1            /* START DATA BYTE                    */
      END                      /*                                    */
    ELSE                       /* OTHERWISE                          */+
      DO                       /*                                    */
        SET &LOCA=1            /* START COLUMN FOR SEQ NUMBER        */
        SET &LOCB=9            /* START DATA BYTE                    */
      END                      /*                                    */
    SET &RECORD=&STR(RECORD FIELD=(8,&LOCA,,1),FIELD=(&SIZE,&LOCB,,10))
  END                          /*                                    */
WRITE ******************************************************
WRITE * PREPARING JOB &SYSUID&JOBCHAR TO BE SUBMITTED AT &SYSTIME   *
```

'D95MRT2.CLIST(PRINTA)' Command Procedure (Part 4 of 4)

```
EDIT 'D95MRT2.CNTL(SUBMITPA)' CNTL OLD
010 //&SYSUID&JOBCHAR JOB '&ACCT,B&BLDG&CUBE,S=1','&SYSUID',
012 //   CLASS=&CLASS,MSGLEVEL=1,NOTIFY=&SYSUID
014 ZZ*************************************************
016 ZZ*  JOB SUBMITTED AT &SYSTIME ON &SYSDATE         ***
020 ZZ*************************************************
024 SEND '%%JOB &SYSUID.&JOBCHAR  STARTING ' U(&SYSUID) WAIT
CHANGE 014 020 ?ZZ?//? ALL
060 //SYSUT1   DD DSN=&DSNAME,DISP=SHR
SET &SYSDVAL=&MEMBR                /* PREPARE TO EDIT MEMBER CARDS      */
READDVAL A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15 A16 A17 A18
SET &N=1                           /* SET BEGINNING INDEX TO 1          */
SET &COUNT=100                     /* SET BEGINNING LINE NUMBER         */
DO WHILE &A&N¬= && &N<19           /* DO UNTIL NO MORE                  */
  &COUNT     MEMBER NAME=&A&N
  SET &N=&N+1                      /* INCREMENT INDEX                   */
  SET &COUNT=&COUNT+50             /* INCREMENT COUNT                   */
  &COUNT     &RECORD
  SET &COUNT=&COUNT+50             /* INCREMENT LINE NUMBER AGAIN       */
END                                /*                                   */
SET &N=&N-1                        /* BACK DOWN INDEX                   */
SET &MSG=                          /* CLEAR MSG VARIABLE                */
IF &N=0 THEN                       /* IF NO MEMBER RECORDS              */+
  &COUNT     &RECORD
ELSE                               /* OTHERWISE                         */+
  DO                               /* SET &FLDS USED                    */
    SET &FLDS=&FLDS*&N             /* ADJUST NUMBER OF FIELDS           */
    SET &MSG=&STR(,MAXNAME=&N)     /* SET NUMBER OF NAMES               */
  END                              /*                                   */
080     PRINT TYPORG=&DSORG,MAXFLDS=&FLDS&MSG
090     TITLE  ITEM=('DATE=&SYSDATE        TIME=&SYSTIME',20)
091     TITLE  ITEM=('DATA   SET   &DSNAME',1)
35 ZZOUTPUT MRT   COPIES=&COPIES
CHANGE ?ZZ?/*?
CONTROL NOMSG
SUBMIT
CONTROL MSG
WRITE * JOB &SYSUID&JOBCHAR SUBMITTED AT &SYSTIME ON &SYSDATE        *
WRITE ******************************************************
END
```

'D95MRT2.CNTL(SUBMITPA)'

This data set consists of the model JCL edited by the PRINTA command procedure to produce a job subsequently submitted to the system.

```
-- JOB CARD INSERT
-- JOB CARD CONT.
//STEP1   EXEC  PGM=IKJEFT01
//SYSPRINT DD SYSOUT=A
//SYSIN    DD *

-- SEND CARD
/*
//STEP2   EXEC  PGM=IEBPTPCH
//SYSPRINT DD SYSOUT=A
//SYSUT2   DD SYSOUT=(A,,MRT),DCB=LRECL=133
--SYSUT1   DD INSERT
//SYSIN DD *
/*
```

Example 3

This segment of a command procedure illustrates the use of &DATATYPE, &LENGTH, &STR, and &SUBSTR built-in functions.

```
write please enter a number from 1 through 999
read &ans
do while &datatype( &ans )=char or &length( &ans ) gt 3
  writenr invalid answer &ans - reenter &str( - )
  read &ans
end
set &cnt=&substr( &length( &ans ),&ans )
write the last digit is a &cnt
set &alphabet = abcdefghijklmnopqrstuvwxyz
write the &cnt letter of the alphabet is -
  &substr( &cnt,&alphabet )
write the next five letters are -
  &substr( &cnt+1:&cnt+5,&alphabet )
write enter a simple expression (for example, 1+1)
read
write the answer is &eval( &sysdval )
/* If a variable or a string contains any of the */
/* operators, use &str to use them as characters */
  if &str( &variable ) = &str( * ) then +
   write variable is an asterisk
/* Use &substr to check a character string */
/* for left parenthesis */
set &=1
do while &i le &length( &dsn )
  if &substr( &i,&dsn )=&substr( 1,( ) then goto member
end
```

Appendix A: IBM 2741 Communication Terminal

The IBM 2741 Communication Terminal resembles a conventional IBM Selectric ® typewriter mounted on a terminal stand. The 2741, however, has two controls not found on the typewriter; the Terminal-Mode switch (labeled COM/LCL) is located on the left side of the terminal stand, and the Attention key (labeled ATTN) is located on the upper right side of the keyboard,replacing the INDEX key.

If the COM/LCL switch is set to LCL, the terminal can be used as a conventional typewriter. To use the 2741 with TSO, set the COM/LCL switch to COM.

The ATTN key is described in "How to Interrupt Operations from the Terminal".

TSO supports three special features available on the 2741:

- The Transmit Interrupt special feature, discussed in "How to Enter Data"
- The Print Inhibit special feature, discussed in "Contacting TSO"
- The Receive Interrupt special feature, discussed in "How to Interrupt Operations from the Terminal"

All of these special features are recommended for use with TSO.

This appendix discusses how to:

- Start a TSO terminal session
- Enter data
- Interrupt operations from the terminal
- End a TSO terminal session

How To Start a TSO Terminal Session

Three operations are involved in starting a TSO terminal session:

1. Contacting the computer — that is, establishing a connection between the terminal and the main computer system
2. Contacting TCAM — that is, identifying your terminal to the proper TCAM Message Control Program if your terminal is attached to TCAM through VTAM
3. Contacting TSO — that is, identifying yourself to TSO

Contacting the Computer

A 2741 can be permanently connected to a computer system through a non-switched (or leased) line, or temporarily connected (like a telephone connection) through a switched (or dial) line.

If your terminal has a non-switched line:

1. Set the COM/LCL switch to COM.
2. Turn the ON/OFF switch located on the right of the keyboard to ON. (If the switch is on, turn it off, then on again.) If the keyboard

unlocks, the system is ready to receive input data. If the keyboard does not unlock, the system is not available, and you must try later.

Because of the many types of dial-up devices, procedures for dialing the CPU cannot be described in detail. Figure 23 shows typical procedures for terminals connected to the switched line through a telephone data set. Figure 24 shows typical procedures for terminals using an acoustic coupler.

For more detailed instructions, you should refer to the operating instructions for the specific type of dialing device being used. If your terminal connects to the CPU through an IBM 3704/3705 Communications Controller using the Multiple Terminal Access feature (MTA), you must sign on by doing the additional steps shown in Figure 25. This sign-on allows the 3704/3705 to identify the type of terminal you are using.

1. Set the COM/LCL switch to COM.
2. Turn the ON/OFF switch located on the right of the keyboard to ON.
3. Press the TALK button on the telephone modem.
4. Remove the handset from the cradle and dial the system's telephone number. Your installation should supply the number.
5. Wait for a high-pitched tone. If the number is busy or if there is no answer, hang up and try again.
6. When you hear the high-pitched tone, push the DATA button. The DATA light should go on. The keyboard will unlock and the system is ready to receive input data. If the DATA light goes off at any time during the terminal session, you must retry from step 3.
7. Place the handset in the cradle.

Figure 23. Telephone Modem Technique for the IBM 2741 Communication Terminal

1. Set the COM/LCL switch to COM.
2. Turn the ON/OFF switch located on the right of the keyboard to ON.
3. Make sure the acoustic coupler is:
   (a) connected to a power supply
   (b) turned off
   (c) connected to the terminal.
4. Remove the handset from the cradle and dial the system's telephone number. Your installation should supply the number.
5. Wait for a high-pitched tone. If the number is busy or if there is no answer, hang up and try again.
6. When you hear the high-pitched tone, place the handset face down in the coupler box. Make sure the cord is in the slot. Close the lid of the acoustic coupler and latch it.
7. Turn on the acoustic coupler within 20 seconds of when you hear the high-pitched tone. The keyboard will unlock and the system is ready to receive input data.

Figure 24. Acoustic Coupler Technique for the IBM 2741 Communication Terminal

1. When the keyboard unlocks, enter the two characters /" (slash, double quotes).
2. If an MTA index number is required in your installation, follow the /" with the two-digit index number (00, 11, etc.).
3. Press the RETURN key.

If the type element does not move within a few seconds after the carrier returns, you have signed on successfully and can continue with the procedures for contacting TSO. If the type element "wiggles" shortly after carrier return, sign-on was unsuccessful and you should begin again with step 1 of this procedure.

If you delay too long in completing the sign-on message, or if it is entered in error more times than allowed in your installation, the line connection to the CPU is automatically broken and you must begin again with step 3 of Figure 23 or 24.

Figure 25. Sign-On Technique for Terminals Attached to an IBM 3704/3705 Communications Controller MTA Line

Contacting TCAM

If your terminal is attached to TCAM through VTAM, you must contact TCAM as an application of VTAM before contacting TSO. If your terminal is automatically put in contact with TCAM or is put in contact with TCAM by the network operator, you need only contact TSO. If you are not put in contact with TCAM by either of the above mentioned methods, you must enter an installation-defined character string from your terminal to contact TCAM. This character string will either be the character string defined in the VTAM Interpret Table or the OS/VS2 (standard) logon message consisting of the characters LOGON followed by the name of the TCAM MCP to be contacted. Contacting TCAM must be a separate operation from contacting TSO. After you are in contact with TCAM, then contact TSO.

Contacting TSO

Issue the LOGON command to contact TSO. You have to supply your user identification number (userid). If you do not supply a userid in your LOGON command, the system will prompt you for it.

When there are no defaults, TSO prompts you for any missing operands on the LOGON command. Check with your manager for the defaults at your installation.

When entering the LOGON command, if more than 28 seconds elapse between characters, portions of the command may be lost. TSO will prompt you to re-enter the command.

In some cases, TSO indicates that it is processing your LOGON command by responding:

```
LOGON PROCEEDING
```

but when you are logged on it always types:

```
READY
```

The READY message means that you can enter a command. (Note that TSO responds in uppercase except when it is displaying data defined as text.)

Some installations will also require:

- A valid password
- An account number (ACCT operand)
- A procedure name (PROC operand)

The procedure name and the account number, if required by your installation, must be enclosed in parentheses.

If your 2741 has the Print Inhibit special feature, you can enter your password without having it printed at the terminal. Enter the LOGON command and your userid. If your installation requires a password and you have not supplied it with your LOGON command, TSO types:

```
ENTER PASSWORD
```

When you type your password, it will be sent to the system, but it will not be printed as you type. Effectively, the typing element will be disconnected while you are typing in the password. After you press the RETURN key to enter the password, the terminal operates in the usual way, printing each character that is typed.

For a description of the full syntax of the LOGON command, see **OS/VS2 TSO Command Language Reference.**

Example

The userid is MYNUM. The procedure is TRYOUT1. Type

```
logon mynum proc(tryout1)
```

and press the RETURN key. (Note that you do not have to type in uppercase.)

Example

The userid is MYID. The password is APASS. The procedure name is TSOPROC. Type

```
logon myid/apass proc(tsoproc)
```

and press the RETURN key. The userid must be the first operand after the LOGON command word. The slash must be entered as a delimiter between the userid and the password. Separate the other operands with a comma or a space.

How To Enter Data

To enter a line of input into the system, type the line of information and press the RETURN key. The system does not consider the line of information complete until the RETURN key is pressed. Consequently, you can correct typing errors in the line of input anytime before you press the RETURN key.

A terminal session is a series of interactions between the terminal and the system. These interactions follow a pattern:

1. The system notifies you that it is ready to accept input by printing one of the following:
   - A message (for example, READY)
   - A line number (for example, 00180)
2. Type a line of input and correct any typing errors in the line.
3. Press the RETURN key.
4. When the system is again ready to accept information, the sequence described in step 1 is repeated.

If your 2741 has the Transmit Interrupt special feature, you can either wait for the system to supply a message to indicate that it is ready to accept input, or you can type ahead without waiting for a message. This special feature also allows the system to interrupt you while you are entering information. If the system has a high priority message to send to you, it will interrupt the input operation and print the message.

If your 2741 does not have the Transmit Interrupt special feature, you must wait for the system to indicate, by unlocking the keyboard, that it is ready to accept input.

A 2741 used with TSO can have one of three character sets: EBCDIC (Extended Binary Coded Decimal Interchange Code, Part number 1167963), BCDIC (Binary Coded Decimal Interchange Code, Part number 1167938), or Correspondence (Part number 1167043). Figure 26 shows the three keyboards associated with the three character sets. The print element on your terminal will have the last three digits of the part number printed on the top. A few special characters are interpreted differently from their keyboard representation. These are:

For Correspondence

± (plus-minus)	becomes | (or)
] (right bracket)	becomes > (greater than)
[(left bracket)	becomes < (less than)

For BCDIC

± (plus-minus)	becomes | (or)
¤ (lozenge)	becomes ¬ (not)

Correcting Typing Errors

Two ways to correct typing errors are:

- Backspace to the error and then retype the line from that point. When the line contains the correct information, press the RETURN key to enter the information into the system.
- Press the ATTN key to delete the entire line. The system acknowledges that it has deleted the line by printing the characters !D. The system then advances the paper to accept a new line.

The techniques above are defaults. Other ways to correct typing errors can be defined with the PROFILE command, which is described in **OS/VS2 TSO Command Language Reference.**

How To Interrupt Operations from the Terminal

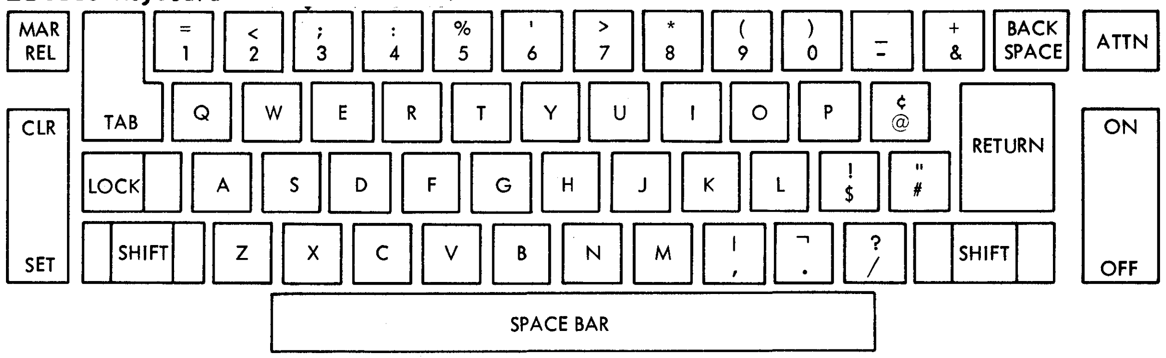
An attention interruption is a signal from your terminal to TSO that you want to interrupt the operation that is taking place. You can use an attention interruption to:

- Delete a line of input that you have typed but have not entered into the system
- Stop the listing of output being sent to your terminal by the system
- Interrupt the command or program that is executing

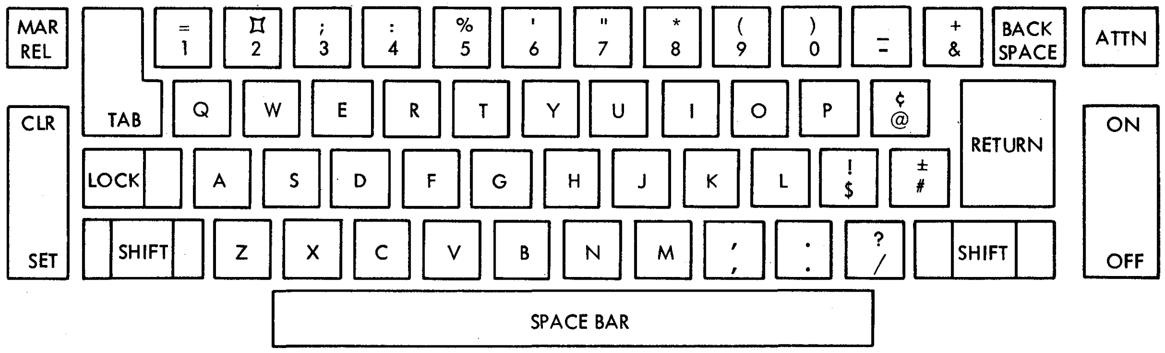
An attention interruption may be entered by:

- Pressing the ATTN key
- Simulating an attention interruption

EBCDIC Keyboard (Part Number 1167963)



BCDIC Keyboard (Part Number 1167938)



Correspondence Keyboard (Part Number 1167043)

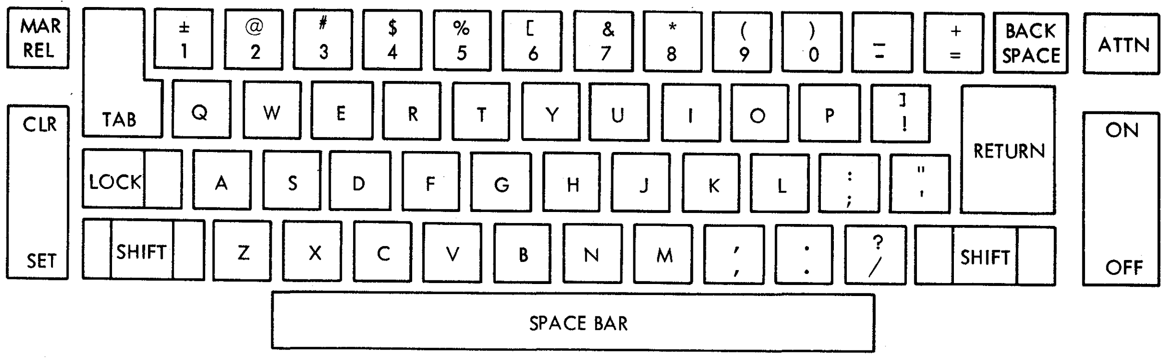


Figure 26. IBM 2741 Communication Terminal Keyboards

The ATTN Key

The simplest way to enter an attention interruption is to press the ATTN key. The system replies by printing one of the following responses:

```
! or !D or !|
```

If your 2741 has the Receive Interrupt special feature, the system will respond to the ATTN key at any time. Without the special feature, the system will respond to the ATTN key only when input can be entered.

TSO Responses to an Attention Interruption

There are three possible responses to an attention interruption:

```
!D or ! or !|
```

If an attention interruption was entered to delete a line of input that you have typed but have not entered into the system, TSO responds by printing !D and advancing the paper to accept a new line of input.

If you enter an attention interruption to stop printing at your terminal or to stop the execution of a program or command, TSO prints !, advances the paper to a new line, and prints a message.

If the message is READY, you have interrupted a command. You can either:

- Enter another command
- Enter a null line by pressing the RETURN key to continue execution of the interrupted command
- Enter another attention interruption to receive the READY message.

If the message is a command name, you have interrupted a subcommand. You can either:

- Enter another subcommand
- Enter a null line by pressing the RETURN key to continue execution of the subcommand
- Enter another attention interruption to receive the READY message.

Simulated Attention Interruptions

Another way to enter an attention interruption is to simulate one. Early in the terminal session, enter the TERMINAL command. The TERMINAL command specifies the conditions under which you want a simulated attention interruption to occur. (See **OS/VS2 TSO Command Language Reference**, for a description of the TERMINAL command.)

If your 2741 has both the Transmit Interrupt special feature and the Receive Interrupt special feature, you can enter an attention interruption at any time by pressing the ATTN key. You also can use the TERMINAL command to specify when you want to enter a simulated attention interruption.

If your 2741 has neither the Transmit Interrupt nor the Receive Interrupt special features, you can only use the ATTN key when you can enter a line of input. Therefore, early in the terminal session, issue a TERMINAL command to specify when TSO is to allow you the opportunity to request an attention interruption. This will allow you to enter an attention interruption by:

- Pressing the ATTN key when you can enter input
- Having previously specified (with the TERMINAL command) that an opportunity to request an attention interruption is to take place after a given time interval. For example, assume a 30-second time interval. The computer would process your command for 30 seconds; if neither input nor output takes place, TSO will signal that it is ready to accept an attention interruption by jiggling the print element. You may then press the ATTN key, enter a character string that you have defined as the attention interruption indicator, or enter a null line. (To enter a null line, press the RETURN key without typing anything.) After 30 more seconds of program execution, the sequence will repeat.
- Having previously specified (with the TERMINAL command) that an attention interruption is to take place after a given number of lines of output have been printed at the terminal. For example, assume you have specified a 50 line interval. After 50 consecutive lines of output have been printed at the terminal, TSO will signal that it is ready to accept an attention interruption. You may then press the ATTN key, enter a character string that you have defined as the attention interruption indicator, or continue by entering a null line. (To enter a null line, press the RETURN key without typing anything.) You will have the opportunity to request an attention interruption after every 50th consecutive line of output.

If your 2741 has the Receive Interrupt special feature, but not the Transmit Interrupt special feature, you can signal an attention interruption by:

- Pressing the ATTN key at any time.
- Having previously specified (with the TERMINAL command) that an attention interruption is to take place after a given number of lines of output have been printed at the terminal. For example, if you have specified a 50-line interval, after 50 consecutive lines of output have been printed at the terminal, TSO will signal that it is ready to accept an attention interruption by jiggling the print element. You may then enter an attention interruption by pressing the ATTN key, or continue by entering a null line (pressing the RETURN key). You will have the opportunity to request an attention interruption after every 50th consecutive line of output.

Attention Interruption Levels

When TSO indicates that it is ready to accept a simulated attention interruption, you can press the ATTN key, enter a character string that you defined in the TERMINAL command, or enter a digit from 1 to 9. Entering a 1 is the same as entering the simulated attention interruption character string. If you enter a digit other than 1, you will cause a higher level of attention interruption.

The following sample of a portion of a terminal session illustrates how to obtain a higher level of attention interruption.

Assume you are listing part of a data set using the LIST subcommand of the EDIT command, and have requested, through the TERMINAL command, that you be given a chance to request a simulated attention interruption after every third line of continuous output.

The listing at the terminal would look like this:

```
list
000010This is the first line of the data set.
000020This is the second line of the data set.
000030This is the third line of the data set.
```

At this point TSO would indicate that you have the opportunity to cause a simulated attention interruption.

By Entering a Character String: If you entered the character string defined in the TERMINAL command (for example SIMA), TSO would print !, advance the paper one line, and print a message telling you that a subcommand of EDIT has been interrupted:

```
sima!
EDIT
```

At this point, you could enter any subcommand of EDIT, enter a null line to continue with the listing of the data set, or cause another attention interruption.

If you then entered the character string for a simulated attention interruption, or caused another attention interruption by pressing the ATTN key, TSO would again print the character !, advance the paper, and print a READY message.

```
sima!
READY
```

The READY message means that you can enter any command.

The sequence would look like this:

```
list
000010This is the first line of the data set.
000020This is the second line of the data set.
000030This is the third line of the data set.
sima!
EDIT
sima!
READY
```

By Entering a Digit: If, instead of entering the character string for simulated attention interruption after the third line of the data set, you enter a 2, TSO would print the character !, advance the paper, and print the READY message. The sequence would look like this:

```
list
000010This is the first line of the data set.
000020This is the second line of the data set.
000030This is the third line of the data set.
2!
READY
```

You could use this facility to stop using a subcommand of EDIT and start using another command.

Entering a digit in response to an opportunity for a simulated attention interruption is like entering that many attention interruptions, and letting the system respond each time.

How to End a TSO Terminal Session

To end a terminal session, enter either:

```
logoff
   or
logon
```

Logging on ends the present session and automatically starts a new session. In either case, TSO types:

```
userid LOGGED OFF TSO AT time ON date
```

In place of the lowercase letters, TSO prints out information applicable to the terminal session.

When you are finished, turn the terminal off.

Appendix B: IBM 1052 Printer-Keyboard

TSO supports the IBM 1052 Printer-Keyboard.

Several special features are available with the 1052 Printer-Keyboard. Those supported by TSO are:

- The Receive Interrupt special feature, discussed in "How to Interrupt Operations from the Terminal"
- The Transmit Interrupt special feature, discussed in "How to Enter Data"
- The Auto EOB special feature, discussed in "How to Enter Data"
- The Text Time-Out Suppression special feature, discussed in "How to Enter Data"

This appendix discusses how to:

- Start a TSO terminal session
- Enter data
- Interrupt operations from the terminal
- End a TSO terminal session

The control panel switches must be set as shown in Figure 28. The diagram of the 1052 control panel in Figure 27 also shows the correct switch settings.

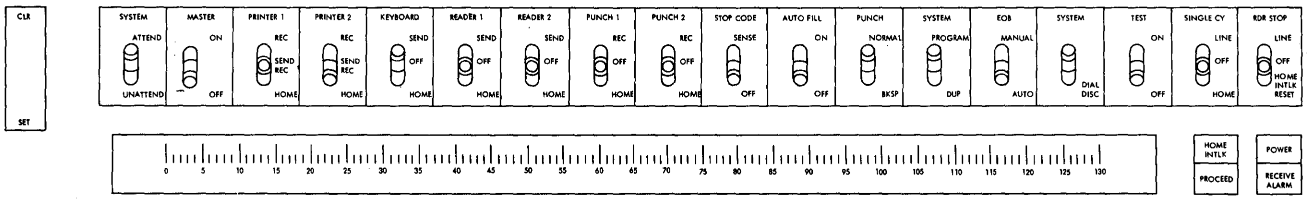


Figure 27. IBM 1052 Printer-Keyboard Control Panel

Switch	*Setting*
SYSTEM	ATTEND
MASTER	OFF
PRINTER1	SEND/REC
PRINTER2	HOME
KEYBOARD	SEND
READER1	OFF
READER2	OFF
PUNCH1	OFF
PUNCH2	OFF
STOP CODE	OFF
AUTO FILL	OFF
PUNCH	NORMAL
SYSTEM	PROGRAM
EOB	AUTO
SYSTEM	UP
TEST	OFF
SINGLE CY	OFF
RDR STOP	OFF

Figure 28. Proper Switch Settings on the IBM 1052 Printer-Keyboard Control Panel

How to Start a TSO Terminal Session

Three operations are involved in starting a TSO terminal session:

1. Contacting the computer — that is, establishing a connection between the terminal and the main computer system.
2. Contacting TCAM — that is, identifying your terminal to the proper TCAM Message Control Program if your terminal is attached to TCAM through VTAM.
3. Contacting TSO — that is, identifying yourself to TSO.

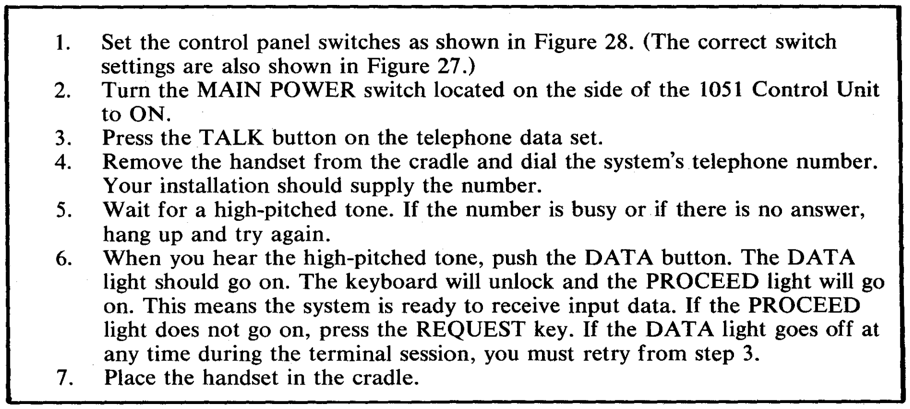
Contacting the Computer

A 1052 Printer-Keyboard can be permanently connected to a computer system through a nonswitched (or leased) line or temporarily connected (like a telephone connection) through a switched (or dial) line.

If your terminal has a nonswitched line:

1. Set the control panel switches as shown in Figure 28. (The correct switch settings are also shown in Figure 27).
2. Turn the MAIN POWER switch located on the side of the 1051 Control Unit to ON. The keyboard will unlock and the PROCEED light will go on. This means the system is ready to receive input data. If the PROCEED light does not go on, press the REQUEST key. If the PROCEED light still does not go on, the system is unavailable and you must try again later.

Because of the many types of dial-up devices, procedures for dialing the CPU cannot be described in detail. Figure 29 shows typical procedures for terminals connected to the switched line through a telephone modem. Figure 30 shows typical procedures for terminals using an acoustic coupler. For more detailed instructions, you should refer to the operating instructions for the specific type of dialing device being used. If your terminal connects to the CPU through an IBM 3705 Communications Controller using the Multiple Terminal Access feature (MTA), you must sign on by doing the additional steps shown in Figure 31. This sign-on allows the 3705 to identify the type of terminal you are using.



1. Set the control panel switches as shown in Figure 28. (The correct switch settings are also shown in Figure 27.)
2. Turn the MAIN POWER switch located on the side of the 1051 Control Unit to ON.
3. Press the TALK button on the telephone data set.
4. Remove the handset from the cradle and dial the system's telephone number. Your installation should supply the number.
5. Wait for a high-pitched tone. If the number is busy or if there is no answer, hang up and try again.
6. When you hear the high-pitched tone, push the DATA button. The DATA light should go on. The keyboard will unlock and the PROCEED light will go on. This means the system is ready to receive input data. If the PROCEED light does not go on, press the REQUEST key. If the DATA light goes off at any time during the terminal session, you must retry from step 3.
7. Place the handset in the cradle.

Figure 29. Telephone Modem Technique for the IBM 1052 Printer-Keyboard

1. Set the control panel switches as shown in Figure 28. (The correct switch settings are also shown in Figure 27.)
2. Turn the MAIN POWER switch located on the side of the 1051 Control Unit to ON.
3. Make sure the acoustic coupler is:
   (a) connected to a power supply
   (b) turned off
   (c) connected to the terminal
4. Remove the handset from the cradle and dial the system's telephone number. Your installation should supply the number.
5. Wait for a high-pitched tone. If the number is busy or if there is no answer, hang up and try again.
6. When you hear the high-pitched tone, place the handset face down in the coupler box. Make sure the cord is in the slot. Close the lid of the acoustic coupler and latch it.
7. Turn on the acoustic coupler within 20 seconds after you hear the high-pitched tone. The keyboard will unlock and the PROCEED light will go on. This means the system is ready to receive input data. If the PROCEED light does not go on, press the REQUEST key.

Figure 30. Acoustic Coupler Technique for the IBM 1052 Printer-Keyboard

1. When the PROCEED light turns on, enter the two characters /" (slash, double quotes).
2. If an MTA index number is required in your installation, follow the /" with the two-digit index number (00, 11, etc.)
3. Press the RETURN key.
4. Enter an EOB.

If the type element does not move within a few seconds after the carrier returns, you have signed on successfully and can continue with the procedures for contacting TSO. If the type element "wiggles" shortly after carrier return, sign-on was unsuccessful and you should begin again with step 1 of this procedure.

If you delay too long in completing the sign-on message, or if it is entered in error more times than allowed in your installation, the line connection to the CPU is automatically broken and you must begin again with step 3 of Figure 29 or 30.

Figure 31. Sign-On Technique for Terminals Attached to an IBM 3705 Communications Controller MTA Line

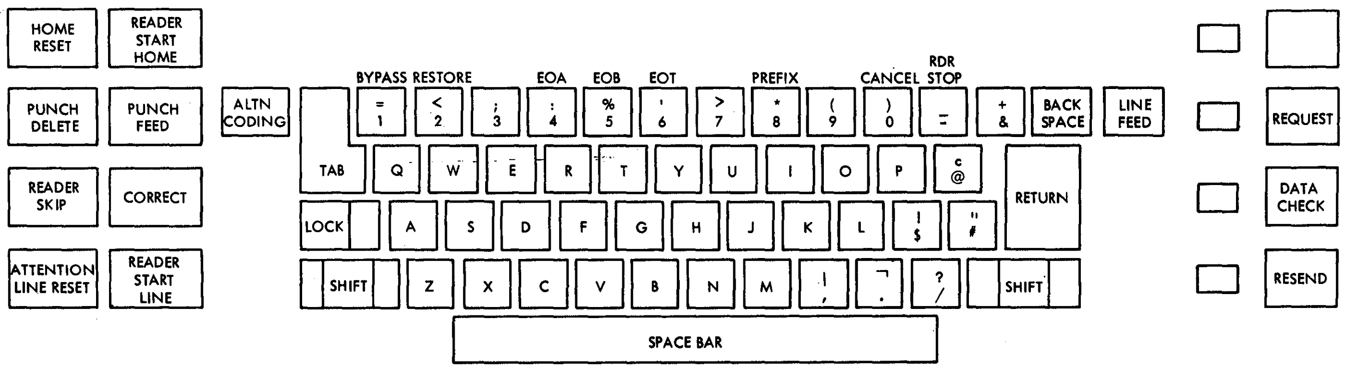


Figure 32. Keyboard of the IBM 1052 Printer-Keyboard

Contacting TCAM

If your terminal is attached to TCAM through VTAM, you must contact TCAM as an application of VTAM before contacting TSO. If your terminal is automatically put in contact with TCAM or is put in contact with TCAM by the network operator, you need only contact TSO. If you are not put in contact with TCAM by either of the above mentioned methods, you must enter an installation-defined character string from your terminal to contact TCAM. This character string will either be the character string defined in the VTAM Interpret Table or the OS/VS2 (standard) logon message consisting of the characters LOGON followed by the name of the TCAM MCP to be contacted. Contacting TCAM must be a separate operation from contacting TSO. After you are in contact with TCAM, then contact TSO.

Contacting TSO

Issue the LOGON command to contact TSO. You have to supply your user identification number(userid). If you do not supply a userid on your LOGON command, the system will prompt you for it.

When there are no defaults, TSO prompts you for any missing operands on the LOGON command. Check with your manager for the defaults at your installation.

In some cases, TSO indicates that it is processing the LOGON command by responding:

```
LOGON PROCEEDING
```

but when you are logged on it always types:

```
READY
```

The READY message means that you can enter a command. (Note that TSO responds in uppercase except when it is displaying data defined as text.)

Some installations will also require:

- A valid password
- An account number (ACCT operand)
- A procedure name (PROC operand)

The procedure name and the account number, if required by your installation, must be enclosed in parentheses.

You can enter your password without having it printed at the terminal. First, enter the LOGON command and your userid. If the installation requires a password and you have not supplied it with your LOGON command, TSO will prompt you for it:

```
ENTER PASSWORD-
```

When you type your password, it will be sent to the system, but will not be printed as you type. Effectively, the typing element will be disconnected while you are typing in the password. After you press the RETURN key to enter the password, the terminal operates in the usual way, printing each character that is typed.

For a description of the full syntax of the LOGON command, see **OS/VS2 TSO Command Language Reference.**

Example

The userid is MYNUM. The procedure name is TRYOUT1. (If the PROCEED light is not on, press the REQUEST key.) Type:

```
    logon mynum proc(tryout1)
```

and press the RETURN key. If the PROCEED light does not go off, hold down the ALTN CODING key and press the EOB(5) key. (Note that you do not have to type in uppercase.)

Example

The userid is MYID; the password is APASS; the procedure name is TSOPROC. (If the PROCEED light is not on, press the REQUEST key.) Type:

```
    logon myid/apass proc(tsoproc)
```

and press the RETURN key. If the PROCEED light does not go off, hold down the ALTN CODING key and press the EOB (5) key. The slash must be entered as a delimiter between the userid and the password. The userid must be the first operand after the LOGON command word. Separate the other operands with a comma or a space.

How to Enter Data

To enter a line of input into the system:

1. Make sure that the PROCEED light is on; if it is off, press the REQUEST key.
2. Type the line of information.
3. Press the RETURN key. If the PROCEED light does not go off, hold down the ALTN CODING key and press the EOB (5) key.

The system does not consider the line of information complete until the PROCEED light goes off. Consequently, you can correct typing errors in the line of input anytime before you press the RETURN key. If you have the Auto EOB special feature, pressing the RETURN key ends a line; that is, causes the PROCEED light to go off. If your 1052 Printer-Keyboard does not have the Auto EOB special feature, that is, if pressing the RETURN key does not turn off the PROCEED light, hold down the ALTN CODING key and press the EOB (5) key to end a line.

A terminal session is a series of interactions between the terminal and the system. These interactions follow a pattern:

1. The system notifies you that it is ready to accept input by printing one of the following:
   - A message (for example, READY)
   - A line number (for example, 00180)
   - A prompting character (an underscore, "—", followed by a backspace)
2. The system then turns on the PROCEED light.
3. Type a line of input and correct any typing errors in the line.
4. Press the RETURN key (or EOB).
5. The system turns off the PROCEED light.
6. When the system is again ready to accept information, the sequence described in step 1 is repeated.

If your 1052 Printer-Keyboard has the Transmit Interrupt special feature, you can either "type ahead" without waiting for a message, or you can wait for the message. You can enter data whenever the PROCEED light is on.

If your 1052 Printer-Keyboard does not have the Text Time-Out Suppression special feature and no data has been entered for approximately 9 to 18 seconds, the keyboard will lock. You must then wait for TCAM to poll the terminal before you can enter data. If your 1052 has the Text Time-Out Suppression special feature, coding the NOTIMEOUT operand on the TERMINAL command prevents the keyboard from locking, regardless of the number of seconds that has elapsed without data being entered.

Correcting Typing Errors

Two ways to correct typing errors are:

- Backspace to the error and then retype the line from that point. When the line contains the correct information, press the RETURN key (or EOB) to enter the information into the system.
- Hold down the ALTN CODING key and press the EOT (6) key to delete the entire line. The system acknowledges that it has deleted the line by printing the characters !D. The system then advances the paper to accept a new line.

The above techniques are defaults. Other ways to correct typing errors can be defined with the PROFILE command, which is described in **OS/VS2 TSO Command Language Reference.**

How to Interrupt Operations from the Terminal

An attention interruption is a signal from your terminal to TSO that you want to interrupt the operation that is taking place. You can use an attention interruption to:

- Delete a line of input that you have typed but have not entered into the system

- Stop the listing if output is being sent to your terminal by the system
- Interrupt the command or program that is executing

An attention interruption may be entered by:

- Holding down the ALTN CODING key and pressing the EOT (6) key
- Pressing the LINE RESET/ATTN key
- Simulating an attention interruption

The LINE RESET/ATTN Key and the EOT Key

The simplest way to cause an attention interruption is to hold down the ALTN CODING key and press the EOT (6) key when the keyboard is unlocked. The system replies by printing one of the following responses:

```
! or !D or !|
```

If your 1052 Printer-Keyboard has the Receive Interrupt special feature, and the keyboard is locked (the PROCEED light is off), you can cause an attention interruption by pressing the LINE RESET/ATTN key. Make sure that the PROCEED light is not on (and will not come on if you hit the REQUEST key).

TSO Responses to an Attention Interruption

There are three possible responses to an attention interruption:

```
! or !D or !|
```

If an attention interruption was entered to delete a line of input which you have typed but have not entered into the system, TSO responds by printing !D, advancing the paper to accept a new line of input, and unlocking the keyboard.

If you cause an attention interruption to stop printing at your terminal or to stop the execution of a program or command, TSO prints either ! or !|, advances the paper to a new line, and prints a message.

If the message is READY, you have interrupted a command. You can:

- Enter another command
- Enter a null line by pressing the RETURN key to continue execution of the interrupted command
- Cause another attention interruption to receive the READY message

If the message is a command name, you have interrupted a subcommand. You can:

- Enter another subcommand of that command
- Enter a null line by pressing the RETURN key to continue execution of the subcommand
- Cause another attention interruption to receive the READY message

Simulated Attention Interruptions

Another way to cause an attention interruption is to simulate one. Early in the terminal session, enter the TERMINAL command. The TERMINAL command specifies the conditions under which you want a simulated attention interruption to occur. (See **OS/VS2 TSO Command Language Reference** for a description of the TERMINAL command.)

If your 1052 Printer-Keyboard has the Receive Interrupt special feature, but not the Transmit Interrupt special feature, you can signal an attention interruption by:

- Holding down the ALTN CODING key and pressing the EOT (6) key any time the PROCEED light is on.
- Pressing the LINE RESET/ATTN key when the PROCEED light is off (and cannot be turned on with the REQUEST key).
- Having previously specified (with the TERMINAL command) that an attention interruption is to take place after a given number of lines of output have been printed at the terminal. For example, assume you have specified a 50-line interval. After 50 consecutive lines of output have been printed at the terminal, TSO will signal that it is ready to accept an attention interruption by jiggling the type element. You may then cause an attention interruption by holding down the ALTN CODING key and pressing the EOT (6) key or continue processing by entering a null line (pressing the RETURN key). You will have the opportunity to request an attention interruption after every 50th consecutive line of output.

Attention Interruption Levels

When TSO indicates that it is ready to accept a simulated attention interruption, you can hold down the ALTN CODING key and press the EOT (6) key, enter a character string that you defined in the TERMINAL command, or enter a digit from 1 to 9. Entering a 1 is the same as entering the simulated attention interruption character string. If you enter a digit other than 1, you will cause a higher level of attention interruption.

The following example of a portion of a terminal session illustrates how to obtain a higher level of attention interruption.

Assume you are listing part of a data set using the LIST subcommand of the EDIT command, and have requested, through the TERMINAL command, that you be given a chance to request a simulated attention interruption every third line of continuous output.

The listing at the terminal would look like this:

```
list
000010This is the first line of the data set.
000020This is the second line of the data set.
000030This is the third line of the data set.
```

At this point TSO would indicate that you have the opportunity to cause a simulated attention interruption.

By Entering a Character String: If you entered the character string defined in the TERMINAL command, (for example, SIMA), TSO would print !, advance the paper one line, and print a message telling you that a subcommand of EDIT has been interrupted:

```
sima!
EDIT
```

At this point, you could enter any subcommand of EDIT, enter a null line to continue with the listing of the data set, or cause another attention interruption.

If you then entered the character string for a simulated attention interruption, or caused an attention interruption by holding down the ALTN CODING key and pressing the EOT (6) key, TSO would again print the character !, advance the paper. and print a READY message.

```
sima!
READY
```

The READY message means that you can enter any command.

The sequence would like this.

```
list
000010This is the first line of the data set.
000020This is the second line of the data set.
000030This is the third line of the data set.
sima!
EDIT
sima!
READY
```

By Entering a Digit: If, instead of entering the character string for a simulated attention interruption after the third line of the data set, you enter a 2, TSO would print the character !, advance the paper, and print the READY message. The sequence would look like this:

```
list
000010This is the first line of the data set.
000020This is the second line of the data set.
000030This is the third line of the data set.
2!
READY
```

You could use this facility to stop using a subcommand of EDIT and start using another command.

Entering a digit in response to an opportunity for a simulated attention interruption is like entering that many attention interruptions, and letting the system respond to each one.

How to End a TSO Terminal Session

To end a terminal session, enter either:

```
logoff
 or
logon
```

Logging on ends the present session and automatically starts a new session. In either case, TSO types:

```
    userid LOGGED OFF TSO AT time ON date
```

In place of the lowercase letters, TSO prints out information applicable to the terminal session.

When you are finished, turn the MAIN POWER switch to OFF.

Appendix C: Teletype* Model 33 and 35

TSO supports the Teletype* Model 33 and 35 terminals (see Figure 33 and Figure 34).

Note: Terminals that are equivalent to those explicitly supported may also function satisfactorily with TSO. The customer is responsible for establishing equivalency. IBM assumes no responsibility for the impact that any changes to IBM-supplied products or programs may have on such terminals.

This appendix discusses how to:

- Start a TSO terminal session
- Enter data
- Interrupt operations from the terminal
- End a TSO terminal session

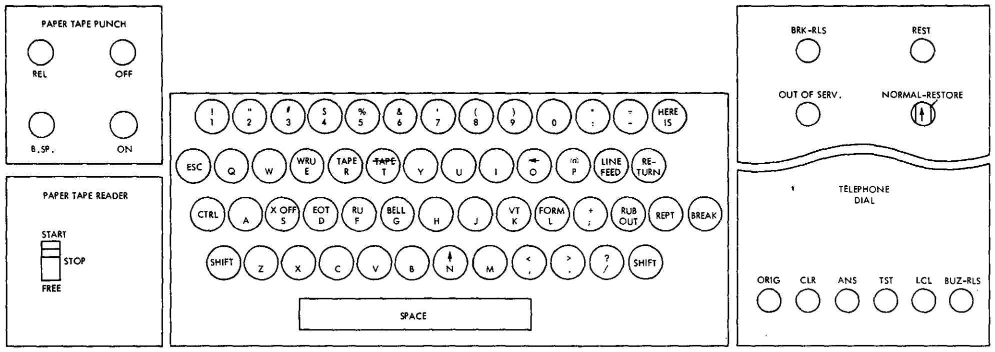


Figure 33. Teletype* Model 33 Keyboard

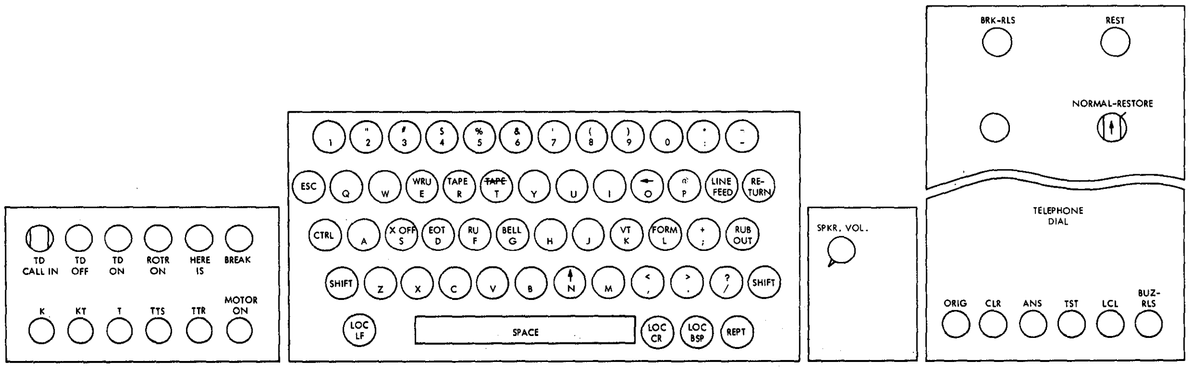


Figure 34. Teletype* Model 35 Keyboard

*Trademark of Teletype Corporation

How to Start a TSO Terminal Session

Three operations are involved in starting a TSO terminal session:

1. Contacting the computer — that is, establishing a connection between the terminal and the main computer system.
2. Contacting TCAM — that is, identifying your terminal to the proper TCAM Message Control Program if your terminal is attached to TCAM through VTAM.
3. Contacting TSO — that is, identifying yourself to TSO.

Contacting the Computer

To make a telephone connection between your Teletype* and the computer:

1. Press the ORIG button to obtain a dial tone. The speaker volume control is under the keyboard shelf to the right. Make sure that the volume is high enough for the dial tone to be audible. After you make a contact, you may turn the volume down.
2. Dial the system's telephone number. Your installation should supply you with the number.
3. A high-pitched tone indicates the terminal is in contact with the system.

If you have a Model 35, press the K button on the left side of the control panel to turn the keyboard on.

If there is no answer or if the number is busy, push the CLR button to break the connection and try again later.

Contacting TCAM

If your terminal is attached to TCAM through VTAM, you must contact TCAM as an application of VTAM before contacting TSO. If your terminal is automatically put in contact with TCAM or is put in contact with TCAM by the network operator, you need only contact TSO. If you are not put in contact with TCAM by either of the above mentioned methods you must enter an installation-defined character string from your terminal to contact TCAM. This character string will either be the character string defined in the VTAM Interpret Table or the OS/VS2 (standard) logon message consisting of the characters LOGON followed by the name of the TCAM MCP to be contacted. Contacting TCAM must be a separate operation from contacting TSO. After you are in contact with TCAM, then contact TSO.

Contacting TSO

Issue the LOGON command to contact TSO. You have to supply your user identification number (userid). If you do not supply a userid on your LOGON command, the system will prompt you for it.

*Trademark of Teletype Corporation

The userid must be the first operand after the LOGON command. the slash is used as a delimiter between the userid and the password. Separate the other operands with a comma or a space.

TSO prompts you for any missing operands on the LOGON command when there are no defaults. Check with your manager for the defaults used by your system.

TSO may type a preliminary message:

```
LOGON PROCEEDING
```

but when you are logged on, TSO types the message:

```
READY
```

The message READY means that you can enter a command.

For a description of the full syntax of the LOGON command, see **OS/VS2 TSO Command Language Reference.**

Some installations also require:

- A valid password
- An account number (ACCT operand)
- A procedure name (PROC operand)

The procedure name and the account number, if required by your installation, must be enclosed in parentheses.

Example

The userid is MYNUM. The procedure name is TRYOUT1. Type:

```
LOGON MYNUM PROC( TRYOUT1 )
```

and press the RETURN key. (Some TSO installations require you to end a line by holding down the CTRL key and pressing the X-OFF key. Check with your installation manager to determine your requirements.)

Note: Teletype does not have lowercase letters.

Example

The password is APASS. The userid is MYID. The procedure name is TSOPROC. Type:

```
LOGON MYID/APASS PROC( TSOPROC )
```

and press the RETURN key (or if your installation requires it, hold down the CTRL key and press the X-OFF key.)

How to Enter Data

A terminal session is a series of interactions between the terminal and the system. These interactions follow a pattern:

1. The system notifies you that it is ready to accept input by printing one of the following:
   - A message (for example, READY)
   - A line number (for example, 00180)
   - A prompting character (a period followed by a carriage return)

   The keyboard on a Teletype* cannot be locked. If the system is not ready to receive input, it will send a character that does not print. This will cause a noise at the Teletype*, an indication not to enter data. When the system is ready to accept input, the noise will cease.

2. Type a line of input and correct any typing errors.
3. Press the RETURN key (or if your installation requires it, hold down the CTRL key and press the X-OFF key).
4. When the system is again ready to accept information, the sequence described in step 1 is repeated.

To enter a line of input into the system, type the line of information and press the RETURN key (or hold down the CTRL key and press the X-OFF key).

The system does not consider the line of information complete until the RETURN key is pressed (or the CTRL key is held down and the X-OFF key is pressed).

Correcting Typing Errors

To correct a typing error of one or more characters, hold down the SHIFT key and press the Backarrow (←) key. Either a backarrow or an underscore prints, and the character immediately preceding it in the input line is deleted. For example, if you mistype TRIAL as

```
TRISL
```

hold down the CTRL key, press the backarrow (←) key twice, and type

```
←← AL
```

The line then appears as:

```
TRISL←←AL
```

but TSO interprets the line as:

```
TRIAL
```

*Trademark of Teletype Corporation

The techniques above are defaults. Other ways to correct typing errors can be defined with the PROFILE command, described in **OS/VS2 TSO Command Language Reference.**

To delete an entire line, hold down the CTRL key and press the X key. (If your installation requires you to end a line by holding down the CTRL key and pressing the X-OFF key, then to delete a line you must hold down the CTRL key and press the X key, and then press the X-OFF key.)

How to Interrupt Operations from the Terminal

An attention interruption is a signal from your terminal to TSO that you want to interrupt the operation that is taking place. Use an attention interruption to:

- Stop the listing of output being sent to your terminal by the system
- Interrupt the command or program that is executing

There are two ways to signal an attention interruption:

- You can cause an attention interruption by pressing the BREAK key, then pressing the BRK-RLS key. If you have a Model 35, you then must press the K key located on the left side of the control panel to reset your terminal so that the system can receive your input.
- You can simulate an attention interruption.

When you cause an attention interruption, the system replies by printing

```
! or !|
```

TSO Responses to an Attention Interruption

If you cause an attention interruption to stop printing at your terminal or to stop the execution of a program or command, TSO prints !, advances the paper to a new line, and prints a message.

If the message is READY, you have interrupted the execution of a command. You can either:

- Enter another command
- Continue execution of the interrupted command by entering a null line (pressing the RETURN key)
- Cause another attention interruption to receive the READY message

If the message is a command name, you have interrupted the execution of a subcommand. You can either:

- Enter another subcommand of that command
- Continue execution of the subcommand by entering a null line (pressing the RETURN key)
- Cause another attention interruption to receive the READY message

Simulated Attention Interruptions

Another way to cause an attention interruption is to simulate one. Early in your terminal session, enter the TERMINAL command. The TERMINAL command specifies the conditions under which you want a simulated attention interruption to occur. (For a description of the TERMINAL command, see **OS/VS2 TSO Command Language Reference**).

You can signal an attention interruption two ways:

1. If your terminal is typing output, you can cause an attention interruption by pressing the BREAK key then pressing the BRK-RLS key. If you have a Model 35, you must then press the K key, located on the left side of the control panel, to reset your terminal so that the system can receive your input.
2. By having previously specified (with the TERMINAL command) that an attention interruption is to take place after a given number of lines of output have been printed at the terminal. For example, assume you have specified a 50-line interval. When 50 consecutive lines of output have been printed at the terminal, TSO will signal that it is ready to accept an attention interruption by jiggling the print element. You can enter a digit to specify level of attention interruption, a character string you have defined as the attention interruption indicator, or continue execution of the program by entering a null line. You will have the opportunity to request an attention interruption after every 50th consecutive line of output.

Attention Interruption Levels

When TSO indicates that it is ready to accept a simulated attention interruption, you can enter either a character string that you defined in the TERMINAL command, or a digit from 1 to 9. Entering a 1 is the same as entering the simulated attention interruption character string. If you enter a digit other than 1, you will cause a higher level of attention interruption.

The following example of a portion of a terminal session illustrates how to obtain a higher level of attention interruption.

Assume you are listing part of a data set using the LIST subcommand of the EDIT command, and have requested, through the TERMINAL command, that you be given a chance to request a simulated attention interruption every third line of continuous output. The listing at the terminal would look like this:

```
LIST
000010THE FIRST LINE OF THE DATA SET.
000020THE SECOND LINE OF THE DATA SET.
000030THE THIRD LINE OF THE DATA SET.
```

At this point TSO would indicate that you have the opportunity to cause a simulated attention interruption.

By Entering a Character String: If you enter the character string defined in the TERMINAL command (for example, SIMA) TSO prints !, advances the paper one line, and prints a message indicating that a subcommand of EDIT has been interrupted:

```
SIMA!
EDIT
```

At this point, you could enter any subcommand of EDIT, continue with the listing of the data set by entering a null line, or cause another attention interruption.

If you then enter the character string for a simulated attention interruption, or cause an attention interruption by pressing the BREAK key and then the BRK-RLS key, TSO again prints the character !, advances the paper, and prints a READY message

```
SIMA!
READY
```

The READY message means that you can enter any command.

The sequence would look like this.

```
LIST
000010THE FIRST LINE OF THE DATA SET.
000020THE SECOND LINE OF THE DATA SET.
000030THE THIRD LINE OF THE DATA SET.
SIMA!
EDIT
SIMA!
READY
```

By Entering a Digit: If, instead of entering the character string for a simulated attention interruption after the third line of the data set, you enter a 2, TSO prints the character !, advances the paper one line, and prints the READY message:

```
LIST
000010THE FIRST LINE OF THE DATA SET.
000020THE SECOND LINE OF THE DATA SET.
000030THE THIRD LINE OF THE DATA SET.
2!
READY
```

You can use this facility to stop using a subcommand of EDIT and start using another command.

Entering a digit in response to an opportunity for a simulated attention interruption is like entering that many attention interruptions, and letting the system respond to each one.

How to End a TSO Terminal Session

To end a terminal session enter either:

```
LOGOFF
 or
LOGON
```

Logging on ends the present session and automatically starts a new session without disconnecting the terminal from the system. In either case, TSO types:

```
userid LOGGED OFF TSO AT time ON date
```

In place of the lowercase letters, TSO prints information applicable to your terminal session.

When you are finished, press the CLR button to break the telephone connection.

Appendix D: IBM 2260 and 2265 Display Stations

The IBM 2260 and 2265 Display Stations are visual display devices with display screens and alphameric keyboards for data entry.

The keyboards resemble IBM Selectric ® typewriter keyboards with additional control keys that modify and control the format and contents of the display screen. Letters, digits, and special characters can be entered on a 2260 or 2265 (see Figure 36).

The format of a standard 2260 or 2265 display screen is 12 rows and 80 columns.

This appendix discusses how to:

- Control the cursor symbol
- Start a TSO terminal session
- Enter data
- Use the TERMINAL command
- Interrupt operations from a terminal
- Control the display
- End a TSO terminal session

Figure 35 lists the various screen control symbols on the 2260 and 2265 Display Stations.

Start a Message	►
2265 Nondestructive Cursor	‾
2260 Nondestructive Cursor	▮
Destructive Cursor	▬
End of Message	■
New Line Symbol	◢

Figure 35. IBM 2260 and 2265 Display Screen Control Symbols

How to Control the Cursor Symbol

The cursor is the system control symbol that indicates on the display screen the location of the next input character in a row.

Two kinds of cursors are available with the IBM 2260 and 2265 Display Stations, the destructive cursor and the nondestructive cursor. (This appendix assumes that your terminal is a 2265 Display Station with a nondestructive cursor.) The nondestructive cursor is a standard feature on the IBM 2265 Display Station and is a recommended special feature on the IBM 2260 Display Station.

The nondestructive cursor is recommended with TSO and appears underneath and slightly to the left of a character position. It does not interfere with a displayed character, but indicates where the next character entered will go.

The nondestructive cursor does not occupy a character position. When backspacing to correct an error, you only have to correct the characters in error.

The destructive cursor occupies a character position and erases any character in that position; that is, you have to enter any characters you passed while backspacing.

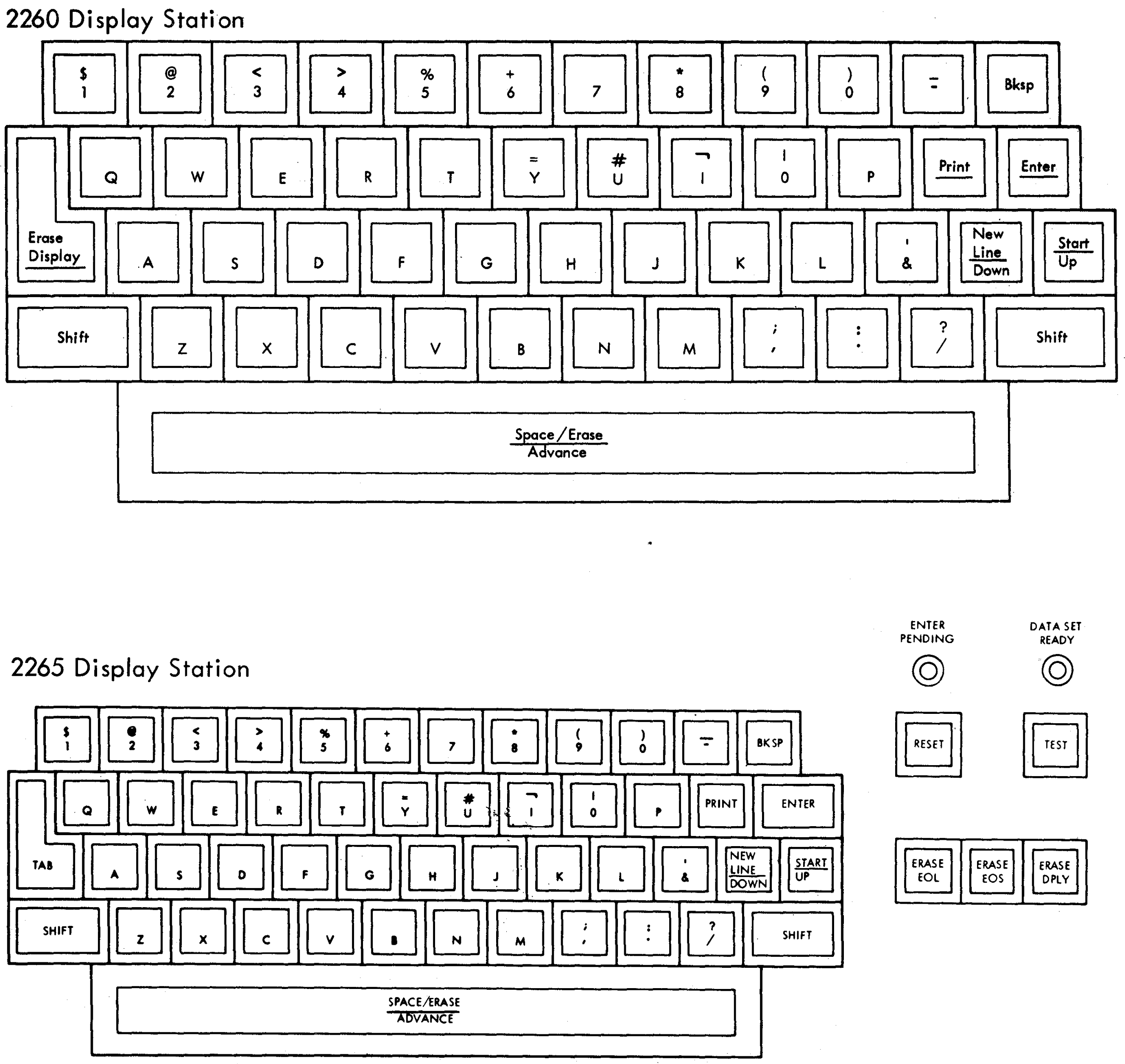


Figure 36. Keyboards for IBM 2260 and 2265 Display Station

The user can move the cursor in four directions. To move the cursor:

- Forward - press the SPACE/ERASE/ADVANCE key and do not hold the SHIFT key down. If the cursor is at the last position in the row, it will go to the first position on the next row.
- Backward - press the BKSP key. If the cursor is at the first position in the row, it will go to the last position in the row above.
- Up - press the START/UP key without holding down the SHIFT key. If the cursor is on the top row, it will move to the corresponding position on the bottom row.
- Down - press the NEWLINE/DOWN key without holding down the SHIFT key. If the cursor is on the bottom row, it will move to the corresponding position on the top row.

How to Start a TSO Terminal Session

An IBM 2260 or 2265 Display Station is permanently connected to a computer system through a direct (or leased) line. When you turn the power on at your display station, your terminal is in contact with the computer system.

The POWER control knob on the IBM 2260 is located on the right side panel of the display screen. Pull the knob out to turn the power on and push it in to turn the power off. The same knob controls the intensity of the display image. Turn the knob clockwise to brighten the image and counterclockwise to darken the image.

The POWER OFF key on the IBM 2265 Display Station is located on the front of the display screen unit just to the right of the screen. The POWER ON key is just below it. The brightness control knob is just above the two power keys. Turn the brightness control knob clockwise to brighten the image and counterclockwise to darken the image.

To start a TSO terminal session:

1. Turn the power on at your Display Station.
2. Using the cursor control keys, move the cursor to the upper left corner of the screen.

After you have positioned the cursor, the top row of your screen should look like this:

```
▬
```

3. Hold down the SHIFT key and press the START/UP key. This will put a Start of Message (▶) symbol in the position the cursor occupies and move the cursor to the right of it.

This is the only time you should press the START key. After you are logged on, TSO will prompt you for input by displaying a Start of Message (▶) symbol.

The first row of the display should look like this:

```
▶ ▬
```

4. Use the LOGON command to identify yourself to TSO. You have to supply your user identification (userid). Some installations may also require:

   - A valid password
   - An account number (ACCT number)
   - A procedure name (PROC operand)

Type the word LOGON, a space, and your userid. If a password is required, type it after the userid, separating the two with a slash. If required, an account number and a procedure name follow, separated with spaces or commas.

Example

The userid is MYNUM. The procedure name is TRYOUT1. Type

```
LOGON MYNUM PROC(TRYOUT1)
```

Your screen will look like this:

```
▶ LOGON MYNUM PROC(TRYOUT1)▬
```

Press the ENTER key. The display will look like this:

```
LOGON MYNUM PROC(TRYOUT1)◢
▬
```

Example

The userid is MYID. The password is APASS. The procedure name is TSOPROC. Make sure the Start of Message (▶) symbol is on the display screen, then type

```
LOGON MYID/APASS PROC(TSOPROC)
```

and press the ENTER key. The display would look like this:

```
LOGON MYID/APASS PROC(TSOPROC)◢
▬
```

TSO may display a preliminary message:

```
LOGON PROCEEDING◢
▬
```

but when you are logged on, TSO displays the message:

```
READY ◢
▶▬
```

The READY message with the Start of Message (▶) symbol means that you can enter any command.

How to Enter Data

A terminal session is a series of interactions between the terminal and the system. These interactions follow a pattern:

1. The system notifies you that it is ready to accept input by displaying either:
   - A message (for example, READY)

     or

   - A line number (for example, 00180) followed by a Start of Message (▶) symbol. The Start of Message (▶) symbol means that TSO is ready to accept your input.
2. Type a line of input and correct any typing errors.
3. Press the ENTER key.
4. The system locks the keyboard.
5. When the system is again ready to accept information, the sequence described in step 1 is repeated.

Note that once the system displays the Start of Message (▶) symbol, it will not display any messages until the ENTER key is pressed.

For example, if after receiving the Start of Message (▶) symbol you type

```
THIS IS A LINE OF INPUT
```

The display would look like this:

```
  ▶THIS IS A LINE OF INPUT ▬
```

If you press the ENTER key, the display would look like this:

```
  ▶THIS IS A LINE OF INPUT ■
```

The cursor (▬) is replaced by the End of Message (■) symbol. When TSO reads the line, it replaces the End of Message (■) symbol with a New Line (◢) symbol unless you have positioned the cursor on the next line by entering a New Line symbol. To enter a New Line (◢) symbol, hold down the SHIFT key and press the NEW LINE/DOWN key, before pressing the ENTER key.

When you press the ENTER key, TSO receives:

```
THIS IS A LINE OF INPUT
```

If you hold down the SHIFT key and press the NEW LINE key instead of the ENTER key, and type:

```
AND THIS ALSO IS INPUT ▬
```

The display would look like this:

```
 ▶ THIS IS A LINE OF INPUT ◢
AND THIS ALSO IS INPUT ▬
```

If you now press the ENTER key, TSO receives two lines:

```
THIS IS A LINE OF INPUT
AND THIS ALSO IS INPUT
```

If the input you are typing is longer than one row on the display screen, the data will automatically go onto the next row, but TSO will interpret it as one line.

If you press the NEW LINE key, any data in a row to the right of a New Line symbol is not sent to TSO.

A line of input to TSO is the data between a Start of Message (▶) symbol or a New Line (◢) symbol, and an End of Message (▬) symbol or a New Line (◢) symbol.

Correcting Typing Errors

You can correct a typing error before you press the ENTER key by backspacing the cursor and retyping. If you want to replace a character by a space, hold down the SHIFT key and press the SPACE/ERASE/ADVANCE key. You should not use the cursor to correct errors in portions of the display that have already been sent to TSO. Doing this will correct the display but will have no effect on the information TSO has already received.

How to Use the Terminal Command

Use the TERMINAL command to specify to TSO:

- The character string to cause an attention interruption (see "How to Interrupt Operations from the Terminal").
- A time interval in seconds. After this time interval has elapsed without opportunity for you to enter input, TSO will allow you to cause an attention interruption (see "How to Interrupt Operations from the Terminal").
- The character string to clear the display screen (see "How to Control the Display").

The options you specify in a TERMINAL command remain in effect until you issue another TERMINAL command or until the end of a session. This means that you should enter a TERMINAL command at the beginning of every TSO session, unless your installation has provided defaults. Your installation may provide a default TERMINAL command through the LOGON procedure you name in the LOGON command. Check with your manager to see what defaults exist for your system.

A TERMINAL command consists of the word TERMINAL which may be abbreviated TERM, and a series of options called keywords, each separated by spaces or commas. Among the keyword options are:

Keywords	*Functions*
SECONDS(nnnn)	Specifies a time interval in seconds. After this number of seconds(nnnn) has elapsed with neither input allowed nor output displayed, TSO allows you to cause an attention interruption to enter the character string that clears the display screen. Specify a number that is a multiple of 10 and is from 10 to 2250.
CLEAR(string	Specifies the character string that clears the display screen. The character string can be up to four characters.
INPUT(string)	Specifies a character string that causes an attention interruption. The character string can be up to four characters.

Example

KLR is the character string that clears the display screen. SIMA is the character string that causes an attention interruption. The time interval is 30 seconds. Type:

```
TERM CLEAR(KLR) INPUT(SIMA) SECONDS(30)
```

TSO responds by displaying:

```
READY◢
▶ ▬
```

When you receive the Start of Message (▶) symbol after the READY message, you can enter any command.

How to Interrupt Operations from the Terminal

An attention interruption is a signal from your terminal to TSO that you want to interrupt the operation that is taking place. Use an attention interruption to:

- Stop the display of output by the system at your terminal
- Interrupt the command or program that is executing
  You can cause an attention interruption by:
- Entering as the only input in any line, the character string you defined in the TERMINAL command
- Entering as the only input in any line, the character string you defined in the TERMINAL command, followed by a digit from one to nine
- Entering a digit to request one or more attention interruptions, when TSO has given you the opportunity to request an attention interruption

TSO indicates that you can request an attention interruption by displaying

```
***▶
```

when

- The display screen is full
- The number of seconds specified in the SECONDS operand of the TERMINAL command has elapsed without your having the opportunity to enter input, or without any output being displayed

An attention interruption may be entered by the user at the request of TSO by:

- Entering the character string specified in the INPUT operand of the TERMINAL command
- Entering a digit from one to nine

If you do not want to cause an attention interruption, enter a null line (press the ENTER key).

TSO Responses to an Attention Interruption

There are two possible responses to an attention interruption:

```
| or |I
```

If you cause an attention interruption to stop the display of output at your terminal or to stop the execution of a program or command, TSO displays |, and a message.

If the message is READY, you have interrupted the execution of a command.

After the Start of Message (▶) symbol has been displayed, you can:

- Enter another command
- Continue execution of the interrupted command by entering a null line (pressing the ENTER key)
- Cause another attention interruption to receive the READY message

If you cause another attention interruption at this point, TSO displays

```
I
```

to indicate that your attention interruption was ignored.

If the message is a command name, you have interrupted a subcommand.

After the Start of Message (▶) symbol has been displayed, you can:

- Enter another subcommand of that command.
- Continue execution of the subcommand by entering a null line (pressing the ENTER key). (If you are displaying the contents of a data set, and enter a null line to continue, a few lines of output may be lost.)
- Cause another attention interruption to receive the READY message.

The following portion of a sample terminal session shows how to cause an attention interruption.

Assume you are executing a program called SUMER and you specified that you wanted the opportunity to cause an attention interruption every 60 seconds. After 60 seconds without output, or without your having the chance to enter input, (signalled by TSO displaying a Start of Message (▶) symbol), TSO displays

```
***▶
```

to allow you to cause an attention interruption.

At this point, you could enter a null line to continue executing SUMER or cause an attention interruption by entering the character string you defined with the INPUT keyword of the TERMINAL command (for example, SIMA).

If you enter the character string, TSO displays the character |, and on successive rows, the message READY and a Start of Message (▶) symbol:

```
***SIMA|◢
READY◢
▶ ▬
```

Attention Interruption Levels

When TSO indicates that it is ready to accept an attention interruption, you can enter the character string that you defined in the TERMINAL command, or a digit from 1 to 9. Entering a 1 is the same as entering the simulated attention interruption character string. If you enter a digit other than 1, you will cause a higher level of attention interruption.

The following sample portion of a terminal session illustrates how to obtain a higher level of attention interruption.

Assume you are creating a data set using the INPUT subcommand of the EDIT command. The display at the terminal would look like this:

```
INPUT ◢
00010THE FIRST LINE OF THE DATA SET. ◢
00020THE SECOND LINE OF THE DATA SET.◢
00030THE THIRD LINE OF THE DATA SET. ◢
00040 ▶ ▬
```

At this point you want to cause a simulated attention interruption.

By Entering a Character String: If you enter the character string defined in the TERMINAL command (for example, SIMA), TSO displays | and on the next row displays a message telling you that you have interrupted a subcommand of EDIT.

```
00040 SIMA|◢
EDIT◢
▶ ▬
```

At this point, you can enter any subcommand of EDIT or cause another attention interruption.

If you then enter the character string for a simulated attention interruption, TSO displays the character | and, on successive rows, the message READY and the Start of Message (▶) symbol:

```
SIMA|◢
READY◢
▶ ▬
```

The READY message means that you have caused the highest level of attention interruption and can enter any command.

The sequence looks like this:

```
INPUT◢
00010THE FIRST LINE OF THE DATA SET. ◢
00020THE SECOND LINE OF THE DATA SET.◢
00030THE THIRD LINE OF THE DATA SET. ◢
00040SIMA|◢
EDIT◢
SIMA|◢
READY◢
▶▬
```

By Entering a Digit: If, instead of entering only the character string for simulated attention interruption after the third line of the data set, you enter the character string followed by a 2, TSO displays the character |, and on successive rows, the message READY and the Start of Message (▶) symbol:

```
INPUT◢
00010THE FIRST LINE OF THE DATA SET.◢
00020THE SECOND LINE OF THE DATA SET.◢
00030THE THIRD LINE OF THE DATA SET.◢
00040SIMA2|◢
READY◢
▶▬
```

You can use this facility if you want to stop using a subcommand of EDIT and start using another command.

If you enter a digit greater than the number of attention interruption levels, you will cause an attention interruption at the highest level available.

How to Control the Display

You can erase the screen any time you can enter data by entering as the only input on a line the CLEAR character string you defined in the TERMINAL command.

After entering the data into the system, the screen may be cleared by entering the CLEAR character string previously defined in the TERMINAL command. The cursor controls may be used to erase the material; however, they are not as efficient as using the CLEAR command.

If you erased the Start of Message symbol:

1. Hold down the SHIFT key and press the START/DOWN key
2. Enter the character string you defined in the TERMINAL command to clear the display screen.

Handling a Full Display Screen

When output data is displayed on the next to the last row, TSO displays:

```
***▶▬
```

on the last row.

When TSO displays this character string, it is about to erase the screen and is allowing you time to review the contents. You can:

- Enter a null line to allow TSO to erase the screen and continue operation
- Enter the character string you defined in the TERMINAL command to cause an attention interruption
- Enter a digit to cause more than one attention interruption.

Note: No matter what you type, after you press the ENTER key, TSO erases the display screen.

Any command or input for a command will be ignored, except for the attention interruption character string, or a digit.

If the amount of time you specified in the SECONDS operand of the TERMINAL command has elapsed and TSO displays ***▶ on the next to the last row, regardless of your response, TSO will clear the display screen.

If you enter data on the next to last line of the screen (or beyond), review the contents before you press the ENTER key. After you press the enter key, TSO will erase the screen and display the contents of this input at the top of the screen, giving you the chance to review your last input.

How to End a TSO Terminal Session

To end a terminal session, enter either:

```
LOGOFF
 or
LOGON
```

Logging on ends the present session and automatically starts a new session. In either case, TSO types:

```
userid LOGGED OFF TSO AT time ON date
```

In place of the lowercase letters, TSO displays information applicable to the terminal session.

Appendix E: IBM 3270 Information Display System (Using TSO/TCAM)

The IBM 3270 Information Display System is a system with display screens and alphameric keyboards for data entry. The 3270 is permanently connected to a computer system through a leased line (Remote) or directly wired to the channel (Local). As a remote, the 3270 is supported in three ways:

- BSC (binary synchronous communication)
- SDLC (synchronous data link control) -- attached directly to 370x
- SDLC (synchronous data link control) -- attached to 3790 control unit with 3270 running as a 3270 emulator

The format of a standard 3270 display screen is either 12 rows and 40 columns or 24 rows and 80 columns.

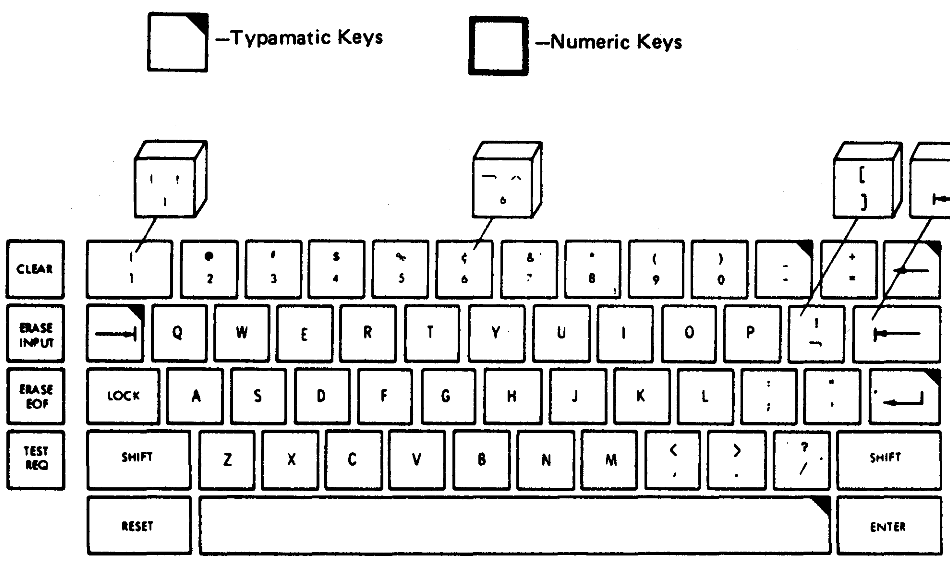
The keyboards resemble IBM Selectric ® typewriter keyboards with additional control keys that modify and control the format and contents of the display screen. Letters, digits, and special characters can be entered on a 3270 (see Figure 37).

TSO support of the 3270 includes a formatted screen and output data of higher intensity than input data. (The first line of output data is indented one position.)

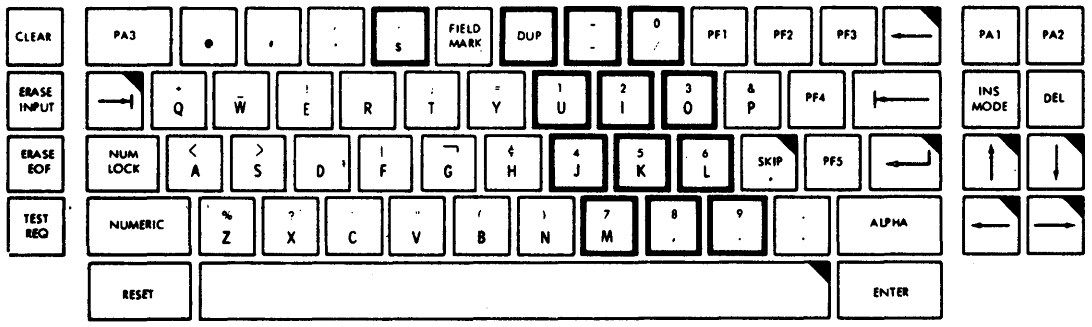
Before you start a TSO terminal session you must know which access method, TCAM or VTAM, your version of TSO uses to direct the transmission of data between your terminal and the host computer. Your system programmer should have this information. This appendix describes how to use a 3270 under TSO/TCAM. If your 3270 uses TSO/VTAM, go to Appendix F.

This appendix discusses how to:

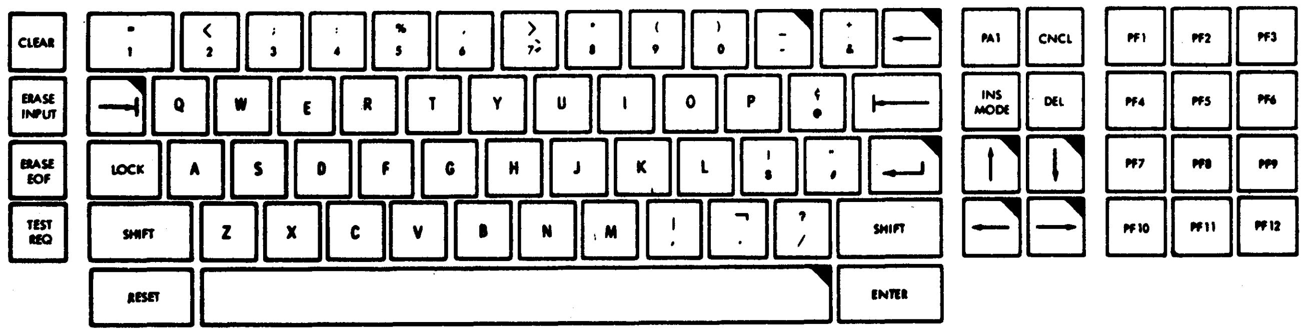
- Control the cursor symbol
- Start a TSO terminal session
- Enter data
- Use the TERMINAL command
- Interrupt operations from a terminal
- Control the display
- End a TSO terminal session



Typewriter Keyboard (EBCDIC) - **The ASCII typewriter keyboard which accommodates both ASCII-A and ASCII-B character set options has four different keys, shown above keyboard.**



Data Entry Keyboard



Operator Console Keyboard

Figure 37. Basic Keyboards for the IBM 3270 Information Display System

How to Control the Cursor Symbol

The cursor is the system control symbol (▬) that automatically indicates on the display screen the location of the next input character in a row.

The user can move the cursor in four directions. To move the cursor:

- Forward - press the space bar or the Forward Cursor key (→). If the cursor is at the last position in a row, it will go to the first position in the next row.
- Backward - press the Backspace or Backward Cursor key (←). If the cursor is at the first position in a row it will go to the last position in the row above.
- Up - press the Up Cursor key (↑). If the cursor is on the top row, it will move to the corresponding position on the bottom row.
- Down - press the Down Cursor key (↓). If the cursor is on the bottom row, it will move to the corresponding position on the top row.

How To Start a TSO Terminal Session

To start a TSO terminal session:

1. If power is off:

   - Pull out the POWER control knob on the left side panel of the display screen. The terminal is now in contact with the system.

   - Turn the POWER control knob clockwise to brighten the image or counterclockwise to darken the image.

   If power is on:

   - Press the CLEAR key and then the RESET key. The cursor moves to the upper left corner of the screen and the INPUT INHIBITED light goes off.

   - If you are at a 3270 emulator (3790) and you are not automatically in contact with TCAM, press the CLEAR key. Logon as a 3790 user. Then invoke function PGM (3270 emulator).

2. If your terminal is an SDLC 3270, before you can log on to TSO a SNA session must first be established between the terminal and TCAM. There are several ways for automatically establishing this session; your installation should define the method to be used. If your installation does not automatically establish the SNA session with TCAM, you must enter the installation-defined character string required and press the TEST REQ key. For BSC and local 3270s, perform step 4.

3. If your installation does not start the TSO session in step 2, then step 4 must be performed. Your installation should provide this information.

4. Enter the LOGON command to identify yourself to TSO. Type the word LOGON, a space, and your user identification (userid).

5. Transmit the LOGON command to TSO by pressing the ENTER key; the INPUT INHIBITED light comes on.

6. Wait for TSO to reply to your LOGON command. TSO may display a preliminary message:

```
LOGON PROCEEDING
```

but when you are logged on, TSO displays the message:

```
READY
```

and turns the INPUT INHIBITED light off. You can now enter any command.

As part of the LOGON command (see the LOGON command in **OS/VS2 TSO Command Language Reference**), installations may also require a password, an account number, and a cataloged procedure name.

First type the word LOGON, a space, and your userid. If a password is required, type it after the userid, separating the two with a slash(/). If required, an account number and a cataloged procedure name follow, separated with spaces or commas.

Example

The userid is MYNUM. The cataloged procedure name is TRYOUT1. Type:

```
LOGON MYNUM PROC(TRYOUT1)
```

Press the ENTER key; the INPUT INHIBITED light comes on. Wait for TSO to display the message READY and to turn the INPUT INHIBITED light off. You may now enter any command.

Example

The userid is MYSEVEN. The password is APASS. The account number is AN38. Type:

```
LOGON MYSEVEN/APASS ACCT(AN38)
```

Press the ENTER key; the INPUT INHIBITED light comes on. Wait for TSO to display the message READY and to turn the INPUT INHIBITED light off. You may now enter any command.

How to Enter Data

TSO accepts one line of input at a time. A line of input is defined as the data typed after the system turns the INPUT INHIBITED light off and before the ENTER key is pressed.

If the keyboard locks because of a hardware malfunction, press the RESET key.

Entering Data

The system notifies you that it is ready to accept a line of input by displaying either a message:

```
  READY
  ▬
or a line number:
  00180
```

and turning the INPUT INHIBITED light off.

Data may now be entered:

1. Type a line of input. Correct any typing errors (see below).
2. Press the ENTER key. The INPUT INHIBITED light comes on.
3. Return to step 1 if you desire to enter more input.

Correcting Typing Errors

If you wish to correct typing errors, you must correct them before you press the ENTER key. Move the cursor under the error and type the correct character. To replace a character by a space, move the cursor under the character and press the Space bar.

NEW LINE Key

If you want to move the cursor to the first position in the next row, press the NEW LINE (↵) key.

Example

You stop typing, press the NEW LINE key, and begin typing on the next row.

```
THIS IS INPUT↵
AND THIS IS ALSO INPUT▬
```

Press the ENTER key; the INPUT INHIBITED light comes on. TSO receives:

```
THIS IS INPUTAND THIS IS ALSO INPUT
```

Note: The 3270 considers the second row a continuation of the first and does not insert a space when moving the cursor to the second row.

On 3270s with the feature RPQ AB3953, pressing the NEW LINE key positions the cursor as described above and, in addition, provides the function of the FIELD MARK key (displays a semicolon at the end of the line and inserts a FIELD MARK character in the transmitted data).

FIELD MARK Key

The FIELD MARK key may be used to enter multiple commands on a single line. Command entry should be concluded with depression of the FIELD MARK key. All commands so entered will be read by TCAM at one time, and processed by TSO in a manner similar to multiple commands entered on a break terminal.

How to Use the Terminal Command

Use the TERMINAL command to specify to TSO:

- The character string to clear the display screen (see "How to Control the Display)"
- The character string to cause an attention interruption (see "How to Interrupt Operations from the Terminal")
- A time interval, after which TSO will allow you to cause an attention interruption (see "How to Interrupt Operations from the Terminal")

Enter a TERMINAL command at the beginning of every TSO session, unless your installation has provided a default TERMINAL command through the logon procedure (PROC) in the LOGON command. The options specified in a TERMINAL command remain in effect until the end of the session or until another TERMINAL command is issued. (See the TERMINAL command in **OS/VS2 TSO Command Language Reference.)**

A TERMINAL command consists of the word TERMINAL (or TERM) and a series of options called keywords, separated by spaces. Among the keyword options are:

Keywords	*Functions*
SECONDS(nnnn)	Specifies a time interval in seconds. After this number of seconds (nnnn) has elapsed with neither input allowed nor output displayed, TSO allows you to cause an attention interruption or enter the character string that clears the display screen. Specify a number that is a multiple of 10 and is from 10 to 2250.
CLEAR(string)	Specifies the character string that clears the display screen. The character string can be up to 4 characters. (The CLEAR key provides the same function.)
INPUT(string)	Specifies a character string that causes an attention interruption. The string can be up to 4 characters. (The PA1 key provides the same function.)

Example

KLR is the character string that clears the display screen. SIMA is the character string that causes an attention interruption. The time interval is 30 seconds. Type:

```
TERM CLEAR(KLR) INPUT(SIMA) SECONDS(30)
```

Press the ENTER key; the INPUT INHIBITED light comes on. Wait for TSO to display the message READY and to turn the INPUT INHIBITED light off. You can now enter any command.

How to Interrupt Operations from the Terminal

An attention interruption is a signal from your terminal to TSO that you want to either stop the display of output at your terminal, or interrupt the command or program that is executing.

An attention interruption may be entered at your own request or in response to a request by TSO.

An attention interruption may be entered at the user's own request by:

- Pressing the PA1 key when the INPUT INHIBITED light is off

- Entering, as the only input in any line, the character string you defined in the INPUT operand of the TERMINAL command
- Entering, as the only input in any line, the character string you defined in the INPUT operand of the TERMINAL command, followed by a digit from 1 to 9.

TSO indicates that you can request an attention interruption by displaying

```
*** —
```

when:

- The display screen is full
- The number of seconds specified in the SECONDS operand of the TERMINAL command has elapsed without your having the opportunity to enter input, or without any output being displayed

An attention interruption may be entered by the user at the request of TSO by:

- Pressing the PA1 key when the INPUT INHIBITED light is off
- Entering the character string you specified in the INPUT operand of the TERMINAL command
- Entering a digit from 1 to 9

TSO Responses to an Attention Interruption

There are two possible responses to an attention interruption:

```
| or |I
```

If you cause an attention interruption to stop the display of output at your terminal or to interrupt the execution of a program or command, TSO displays |, and a message on the next line:

```
  |
  READY
  —
or
  |
  (command)
  —
```

If the message is READY, you have interrupted the execution of a command.

After you have interrupted the execution of a command and the INPUT INHIBITED light goes off, you can:

- Resume execution of the command by entering a null line (pressing the ENTER key)
- Enter another command

If the message is a command name, you have interrupted a subcommand.

After you have interrupted the execution of a subcommand and the INPUT INHIBITED light goes off, you can:

- Resume execution of the subcommand by entering a null line (pressing the ENTER key). (If you were displaying the contents of a data set and enter a null line to continue, a few lines of output may be lost.)
- Enter another subcommand of that command.

If, after interrupting a command, you enter another attention interruption, TSO ignores it and displays

```
|I
_
```

The following example example shows the responses by TSO to an attention interruption that was requested by TSO.

Example

You specified 60 seconds in the SECONDS operand of the TERMINAL command and the time has elapsed without output being displayed or without your having the opportunity to enter input (indicated by TSO's turning the INPUT INHIBITED light on). TSO displays

```
*** _
```

You cause an attention interruption by entering the character string (SIMA) you defined in the INPUT operand of the TERMINAL command:

```
*** SIMA_
```

TSO responds to the attention interruption by displaying the character |, the message READY, and the cursor on successive lines:

```
*** SIMA
|
READY
_
```

TSO turns the INPUT INHIBITED light off.

Attention Interruption Levels

When TSO indicates that it is ready to accept an attention interruption, you can press the PA1 key, enter a character string that you defined in the INPUT operand of the TERMINAL command, or enter a digit from 1 to 9.

Entering a 1 is the same as pressing the PA1 key or entering the character string that causes an attention interruption. If you enter a digit other then 1, you will cause a higher level of attention interruption.

The following sample portion of a terminal session illustrates how to obtain a higher level of attention interruption.

Example

Assume you are creating a data set using the INPUT subcommand of the EDIT command. The screen displays

```
INPUT

00010THE FIRST LINE OF THE DATA SET
00020THE SECOND LINE OF THE DATA SET
00030THE THIRD LINE OF THE DATA SET
00040 —
```

At this point you want to cause an attention interruption be entering a character string or a digit.

By entering a character string: If you enter the character string (for example, SIMA) defined in the INPUT operand of the TERMINAL command, TSO displays | , and on the next line the message EDIT. The message EDIT means that you have interrupted a subcommand of that command.

```
00040SIMA
|
EDIT
—
```

At this point, you can enter any subcommand of EDIT or cause another interruption.

If you enter the character string (SIMA) for an attention interruption, TSO displays the character | and on the next line the message READY, followed by the INPUT INHIBITED light going off.

```
00040SIMA
|
READY
—
```

The message READY means that you have caused the highest level of attention interruption and can enter any command. The entire session looks like this:

```
INPUT
00010THE FIRST LINE OF THE DATA SET
00020THE SECOND LINE OF THE DATA SET
00030THE THIRD LINE OF THE DATA SET
00040SIMA
|
EDIT

—
SIMA
READY
—
```

By entering a digit: If you enter the character string (SIMA) followed by a 2, TSO displays the character | and on the next line the message READY, followed by the INPUT INHIBITED light going off.

```
INPUT
00010THE FIRST LINE OF THE DATA SET
00020THE SECOND LINE OF THE DATA SET
00030THE THIRD LINE OF THE DATA SET
00040SIMA2
|
READY
_
```

If you enter a digit greater than the number of attention interruption levels, you will cause an attention interruption at the highest level available.

How to Control the Display

You can clear the display screen any time you can enter data by entering as the only input on a line the character string you defined in the CLEAR operand of the TERMINAL command. (The CLEAR key provides the same function.)

Handling a Full Display Screen

Note: Handling a full screen of data should not be confused with Fullscreen support for a 3270. Fullscreen support is an optional function available through TSO. See the **OS/VS2 TCAM Programmer's Guide** for information concerning 3270 Fullscreen support.

When output data is displayed on the next to last row, TSO displays *** — on the last row. In this case, *** means that TSO is about to clear the display screen and is allowing you time to review the contents. This "simulated attention" is not provided when operating in full screen mode. When the simulated attention is displayed, you can:

- Clear the display screen by entering a null line (pressing the ENTER key)
- Clear the display screen by entering the character string you defined in the CLEAR operand of the TERMINAL command (the CLEAR key provides the same function)
- Cause an attention interruption by entering the character string you defined in the INPUT operand of the TERMINAL command (the PA1 key provides the same function)
- Cause more than one attention interruption by entering a digit from 1 to 9
  Note: No matter what you type, after you press the ENTER key, TSO clears the display screen.

If the amount of time you specified in the SECONDS operand of the TERMINAL command has elapsed and TSO displays *** on the next to the last row, regardless of your response, TSO will clear the display screen.

If you enter data on either of the last two rows and press the ENTER key, TSO will clear the display screen and display that input at the top of the display screen.

WARNING: Data entered beyond the last row will "wrap" and be displayed at the top of the screen. If you then press the ENTER key to clear the screen and display the last two rows of data entered, the data sent to TSO and displayed on the screen will be invalid. Therefore, do not wrap the 3270 display screen when entering input.

How to End a TSO Terminal Session

To end a terminal session with TSO, you may either log off or log on. Logging off ends the terminal session with TSO; logging on ends the terminal session with TSO and automatically starts a new session with TSO. If your terminal is supported by TSO/TCAM, your installation can choose to end the SNA session with TCAM automatically when the TSO session ends. If this is the case, follow the procedure below. If your installation selects another option, the installation should provide the proper sequence of commands for logging off.

To end a terminal session, enter either:

```
  LOGOFF —
or
  LOGON  —
```

Logging on ends the present TSO session and automatically starts a new session. In either case, TSO displays:

```
  userid LOGGED OFF TSO AT time ON date
```

In place of the lowercase letters, TSO displays information applicable to the terminal session.

Appendix F: IBM 3270 Information Display System (Using TSO/VTAM)

The IBM 3270 Information Display System is a system with display screens and alphameric keyboards for data entry. The 3270 is permanently connected to a computer system through a leased line (Remote) or directly wired to the channel (Local).

The format of a standard 3270 display screen is either 12 rows and 40 columns or 24 rows and 80 columns.

The keyboards resemble IBM Selectric ® typewriter keyboards with additional control keys that modify and control the format and contents of the display screen. Letters, digits, and special characters can be entered on a 3270 (see Figure 37).

TSO support of the 3270 includes a formatted screen and output data of higher intensity than input data. (The first line of output data is indented one position.)

Before you start a TSO terminal session you must know the following information, which should be supplied by your system programmer:

- Which access method, TCAM or VTAM, your version of TSO uses to direct the transmission of data between your terminal and the host computer. This appendix describes how to use a 3270 under TSO/VTAM. If your 3270 uses TSO/TCAM, go to Appendix E.
- Whether your terminal is a SNA (systems network architecture) or non-SNA 3270.
- The format of your logon command.
- Whether you should use the TSO LOGOFF command or a VTAM LOGOFF command. If using a VTAM LOGOFF command you must know its format.
- Which key to press if your installation allows you to log on or off using one of the program access (PA) or program function (PF) keys.

This appendix discusses how to:

- Control the cursor symbol
- Start a TSO terminal session
- Enter data
- Interrupt operations from a terminal
- Handle a full display screen
- End a TSO terminal session

How to Control the Cursor Symbol

The cursor is the system control symbol (—) that automatically indicates on the display screen the location of the next input character in a row.

The user can move the cursor in four directions. To move the cursor:

- Forward-press the space bar or the Forward Cursor key (→). If the cursor is at the last position in a row, it will go to the first position in the next row.

- Backward-press the Backspace or Backward Cursor key (←). If the cursor is at the first position in a row it will go to the last position in the row above.
- Up-press the Up Cursor key (↑). If the cursor is on the top row, it will move to the corresponding position on the bottom row.
- Down-press the Down Cursor key (↓). If the cursor is on the bottom row, it will move to the corresponding position on the top row.

How to Start a TSO Terminal Session

To start a TSO terminal session:

1. If power is off:

   - Pull out the POWER control knob on the left side panel of the display screen. The terminal is now in contact with the system.

   - Turn the POWER control knob clockwise to brighten the image or counterclockwise to darken the image.

   If power is on:

   - In rapid succession, press the CLEAR key and then the RESET key. The cursor moves to the upper left corner of the screen and the INPUT INHIBITED light goes off.

2. Enter a logon command to identify yourself to TSO, unless you are going to use one of the PA or PF keys in place of typing in a logon command, in which case you should go to step 3. The logon command consists of the word LOGON or its installation-defined substitute, and may be followed by operands containing such information as a password, account number, and cataloged procedure name. Your system programmer should tell you the format of the logon command for your terminal.

3. Transmit the logon command to TSO. If you are typing in the command:

   - Press the ENTER key if your terminal is a non-SNA 3270.

   - Press the TEST REQ key if your terminal is a SNA 3270.

   If you are using one of the PA or PF keys to logon, press the appropriate key. In either case, the INPUT INHIBITED light comes on.

4. Wait for TSO to reply to your logon. TSO displays a preliminary message:

   ```
   userid LOGON IN PROGRESS AT time ON date
   _
   ```

   It might also display:

   ```
   LOGON PROCEEDING
   _
   ```

When you are logged on, TSO displays the message:

```
READY
_
```

and turns the INPUT INHIBITED light off. You can now enter any command.

How to Enter Data

You can enter input to TSO one line at a time or multiple lines at a time. A line of input is defined as the data that is typed at a terminal, followed by pressing the ENTER key or the FIELD MARK key.

The single-line input technique transmits a line of input data to TSO (using the ENTER key) as soon as it is typed. The multiple-line technique accumulates lines of input data (using the FIELD MARK key) in your 3270 terminal buffer and then transmit them together to TSO when the ENTER key is pressed. "Stacking" lines improves performance by decreasing the number of transmissions across the communication line to TSO, thus requiring less processing by TSO.

Entering Data

The system notifies you that it is ready to accept a line of input by displaying either a message:

```
    READY
    _
or a line number:
    00180 _
```

and turning the INPUT INHIBITED light off.

Data may now be entered:

1. Type a line of input.
2. Correct any typing errors (see "Correcting Typing Errors" below).
3. If this is a single-line operation, press the ENTER key. The INPUT INHIBITED light comes on and the data is transmitted to TSO.
4. If this is a multiple-line operation, press the FIELD MARK key at the end of each line except the last. This stacks each line in your 3270 buffer. Press the ENTER key at the end of the last line. The INPUT INHIBITED light comes on and the entire group of stacked lines is transmitted to TSO.
5. Return to step 1 if you want to enter more input.

Correcting Typing Errors

If you wish to correct typing errors, you must correct them before you press the ENTER key. Move the cursor under the error and type the correct character. To replace a character with a space, move the cursor under the character and press the Space bar.

CLEAR Key

You can press the CLEAR key to clear the display screen any time you can enter data. This blanks the screen and positions the cursor at the first position on line 1.

NEW LINE Key

If you want to move the cursor to the first position in the next row, press the NEW LINE (↵) key.

Example

You stop typing, press the NEW LINE key, and begin typing on the next row.

```
    THIS IS INPUT↵
    AND THIS IS ALSO INPUT_
```

Press the ENTER key; the INPUT INHIBITED light comes on. TSO receives:

```
    THIS IS INPUTAND THIS IS ALSO INPUT
```

Note: The 3270 considers the second row a continuation of the first and does not insert a space when moving the cursor to the second row.

On 3270s with the feature RPQ AB3953, pressing the NEW LINE key positions the cursor as described above and, in addition, provides the function of the FIELD MARK key (displays a semicolon at the end of the line and inserts a Field Mark character in the transmitted data).

How to Interrupt Operations from the Terminal

An attention interruption is a signal from your terminal to TSO that you want to either stop the display of output at your terminal, or interrupt the command or program that is being executed.

You can enter an attention interruption at any time by pressing the PA1 key. If the keyboard is locked, press the RESET key to unlock it, then press the PA1 key.

TSO Responses to an Attention Interruption

There are two possible responses to an attention interruption:

```
    | or |I
```

If you cause an attention interruption to stop the display of output at your terminal or to interrupt the execution of a program or command, TSO displays |, and a message on the next line:

```
    |
    READY
    _
  or
    |
    (command)
```

If the message is READY, you have interrupted the execution of a command.

After you have interrupted the execution of a command and the INPUT INHIBITED light goes off, you can:

- Resume execution of the command by entering a null line (pressing the ENTER key)
- Enter another command

If the message is a command name, you have interrupted a subcommand.

After you have interrupted the execution of a subcommand and the INPUT INHIBITED light goes off, you can:

- Resume execution of the subcommand by entering a null line (pressing the ENTER key). (If you were displaying the contents of a data set and enter a null line to continue, an indefinite number of output lines are lost.)
- Enter another subcommand of that command.

If, after interruption a command, you enter another attention interruption, TSO ignores it and displays

```
 | I
 _
```

How to Handle a Full Display Screen

When TSO sends you output data and the display screen becomes full, you must clear the screen. A full screen has output data in the next to last row, and ***— displayed in the last row. The ***— indication from TSO means that you can review the screen, after which you can do one of the following:

- Press the ENTER key to clear the screen.
- Press the PA1 key to cause an attention interruption that stops processing of the current command, clears the screen, and allows you to enter a new command.

If you are sending data to TSO, when you enter data in either of the last two rows and press the ENTER key, TSO clears the display screen and displays that input at the top of the screen.

Note: Data entered beyond the last row "wraps" around and is displayed at the top of the screen. If you then press the ENTER key to clear the screen and display the last two rows of data entered, the data sent to TSO and displayed on the screen will be invalid. Therefore, do not wrap the 3270 display screen when entering input.

How to End a TSO Terminal Session

To end a terminal session, either log off or log on. Logging off ends the terminal session; logging on ends the current terminal session and automatically starts a new session. (*Note:* Use the TSO LOGON command, not one of the VTAM logon commands, to end a terminal session by logging on.)

To log off:

1. Enter a logoff command, unless you are going to use one of the PA or PF keys in place of typing in a logoff command, in which case you should go to step 2. Your system programmer should tell you the format of the logoff command for your terminal.
2. Transmit the logoff command to TSO. If you are typing in the command:
   - Press the ENTER key if your terminal is a non-SNA 3270.
   - Press the ENTER key (after entering a TSO LOGOFF command) or the TEST REQ key (after entering a VTAM logoff command) if your terminal is a SNA 3270.

If you are using one of the PA or PF keys to log off, press the appropriate key.

Whether you are logging off or logging on, TSO displays:

```
userid LOGGED OFF TSO AT time ON date
```

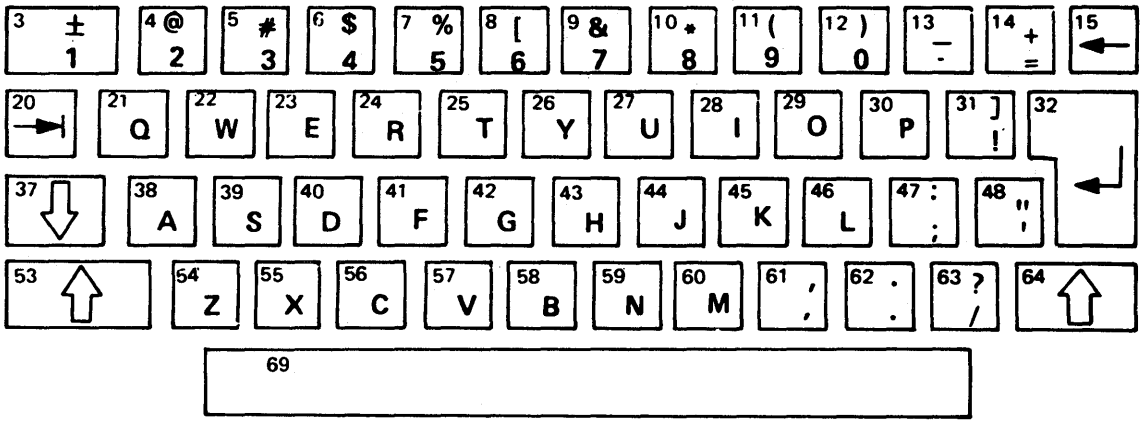
Appendix G: IBM 3767 Communication Terminal

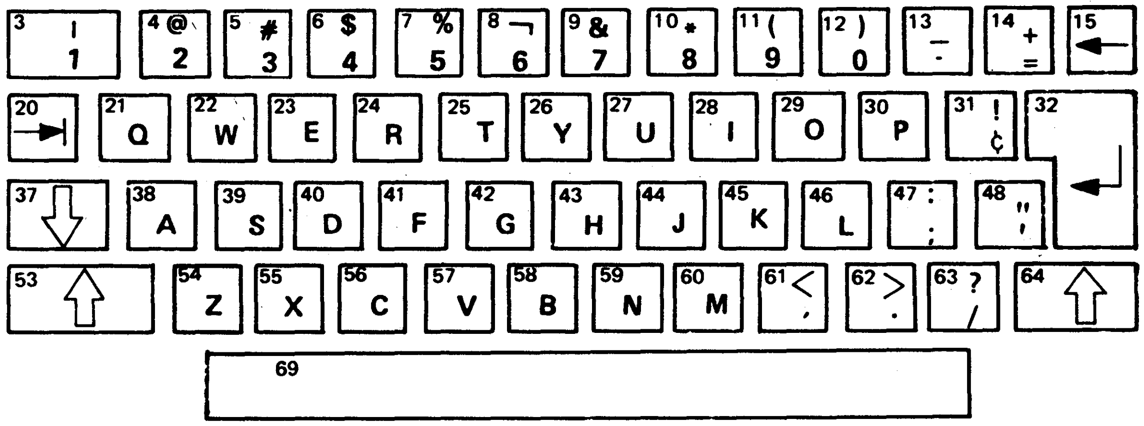
The IBM 3767 Communication Terminal is a compact, movable, desk-top terminal that looks like a conventional typewriter. Its flexibility allows it to be used as a keyboard-printer for normal secretarial typing (in local mode), as a 16-digit calculator when the Calculate-Scientific feature is included, and as a buffered data entry terminal for transmission of data to and from a remote computer.

TSO supports the 3767 as a data entry terminal in communicate mode. The keyboard arrangements supported by TSO are EBCDIC, ASCII, Correspondence (US only), and Katakana (Japan only). Figure 38 shows the four keyboards. In addition to the data keys there are function keys, indicator lights, operating mode switches, and a print position indicator to aid the operator. The printer is a bidirectional matrix printer.

TSO supports the 3767 terminal in a systems network architecture (SNA) configuration. TSO uses a telecommunications access method, either TCAM or VTAM, to direct the transmission of data over the communication line between the terminal and the host computer. A 3767 terminal can be permanently connected to a computer network through a nonswitched (leased) line, or temporarily connected through a switched (dial-up) line. TCAM supports nonswitched lines only; VTAM supports both switched and nonswitched lines.

Before you start your TSO terminal session you must know the type and format of the logon and logoff commands that you should use. Your system programmer should give you this information.


Correspondence


EBCDIC (USA)

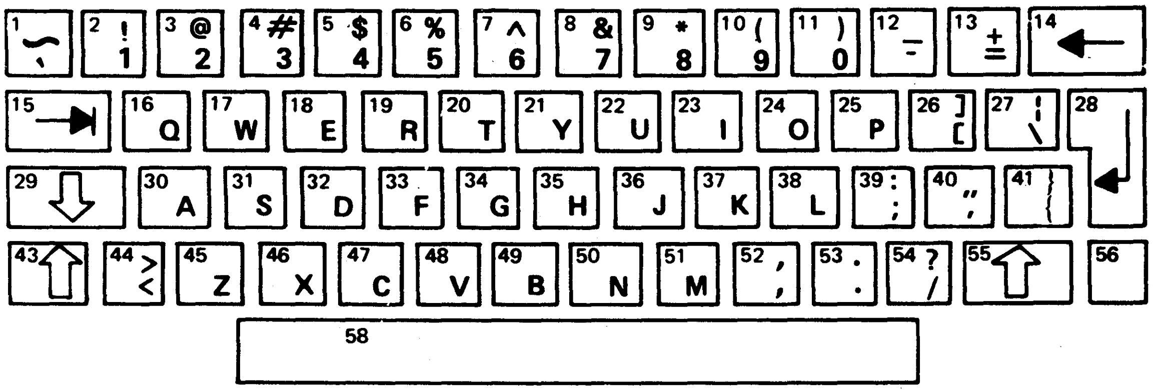
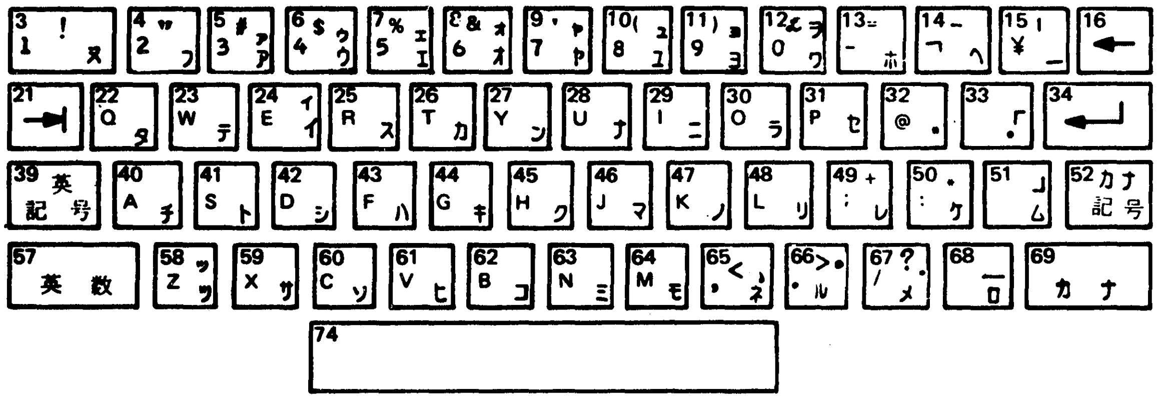
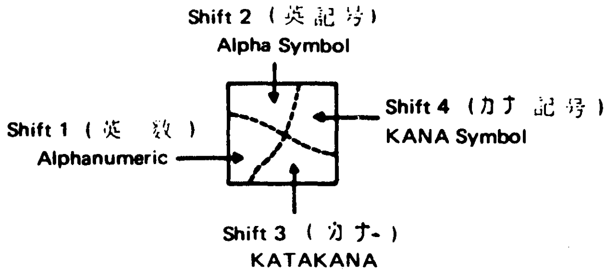

ASCII

Figure 38. IBM 3767 Communication Terminal Keyboards. (Part 1 of 2)



16	Back Space
34	New Line
39	Alpha Symbol Shift
52	Katakana Symbol Shift
57	Alphanumeric Shift
69	Katakana Shift
74	Space
9	Apostrophe
13	Minus
14	Over Line
15	Cho-on
68	Under Line
65	Comma
66	Katakana Period



Katakana (Japan)

Figure 38. IBM 3767 Communication Terminal Keyboards. (Part 2 of 2)

How to Start a TSO Terminal Session

Three operations are involved in starting a TSO terminal session:

1. Setting the switches
2. Contacting the computer
3. Contacting TSO

Setting the Switches

If your terminal has the Security Keylock special feature (located on the right side of the terminal), insert the key and turn it to the rear.

The switches are located above the keyboard, on each side of the alphanumeric readout indicator (ANR). TSO requires these initial settings:

- If the Start-Stop Migration Aid feature is installed, set the SDLC-S/S switch to SDLC. You must set the SDLC-S/S switch before you turn the power on; after the power is on, changing this switch has no effect.
- If the Alternate Character Set (EBCDIC/APL) feature is installed, set the EBCDIC/APL switch to EBCDIC.
- If your terminal is connected to the host computer through a nonswitched line, or through a switched line using a telephone modem, set the Comm/Local switch to Comm. If your terminal is connected through a switched line using an acoustic coupler, set the Comm/Local switch to Local.
- Set the power On/Off switch to On.

The settings for the Auto/Off, Edit/Off, Auto View/Off, and Data/Talk switches depend on the operation you are performing. These settings will be given as each operation is discussed below. The rest of the switches have no required settings for TSO, that is, their use is not significant to TSO. For a description of each switch, see **IBM 3767 Communication Terminal Operator's Guide.**

Contacting the Computer

For Nonswitched Lines: Twenty to thirty seconds after you turn the power On/Off switch on, the On Line light and the Data Set Ready Light will go on. This indicates that the terminal is in contact with the host computer. If the light does not go on, turn the power On/Off switch off, then on again. If the On Line light does not go on this time, the system is not available, and you must try later.

For Switched Lines: Your procedure for dialing the computer depends on the type of dial-up device that you use. Figure 39 shows typical procedures for terminals using an acoustic coupler. Figure 40 shows typical procedures for terminals using a telephone modem. For more detailed instructions, refer to the operating instructions for your specific type of dialing device.

1. Make sure the acoustic coupler:
   (a) is connected to a power supply
   (b) is turned on
   (c) is connected to the terminal
2. Set the Comm/Local switch on the terminal to Local.
3. Remove the handset from the cradle and dial the computer's telephone number. Your installation should supply the number.
4. Wait for a high-pitched tone. If the number is busy or if there is no answer, hang up and try again.
5. When you hear the high-pitched tone, place the handset face down in the coupler box. Make sure the cord is in the slot. Close the lid of the acoustic coupler and latch it.
6. Set the Comm/Local switch to Comm. Now you are ready to contact TSO.

Figure 39. Acoustic Coupler Technique for the IBM 3767 Communication Terminal

1. Press the TALK button on the telephone modem; for terminals in World Trade countries, set the Data/Talk switch on the terminal to Talk.
2. Remove the handset from the cradle and dial the computer's telephone number. Your installation should supply the number.
3. Wait for a high-pitched tone. If the number is busy or if there is no answer, hang up and try again.
4. When you hear the high-pitched tone, push the DATA button on the telephone modem; the Data light should go on. For terminals in World Trade countries, set the Data/Talk switch to Data. The Data Set Ready light on the terminal goes on.
5. Place the handset in the cradle. Now you are ready to contact TSO.

Note: If the DATA light on the telephone modem goes off at any time during the terminal session, you must retry from step 1.

Figure 40. Telephone Modem Technique for the IBM 3767 Communication Terminal.

Contacting TSO

If your terminal is supported by TCAM, your installation can provide a LOGON procedure that automatically establishes a SNA session with TCAM. If this option is selected, enter the TSO LOGON command to identify yourself to TSO: type the word LOGON, a space, and your user identification (userid). Then perform steps 3 and 4 below. If this option is not selected, or if your terminal is not supported by TCAM, use the following procedure to contact TSO:

1. Press the SYS REQ key. The Proceed light goes on.

2. Enter a logon command to identify yourself to TSO. The logon command consists of the word LOGON or its installation-defined substitute, and may be followed by a character string containing such information as a password, account number, and cataloged procedure name. Your system programmer should tell you the format of the logon command for your terminal.

3. Transmit the logon command to TSO by pressing the Return (↵) key and, if the Auto/Off switch is set to Off, the EOM key. The Proceed light goes off.

4. Wait for TSO to reply to your logon command. TSO types a preliminary message:

```
userid LOGON IN PROGRESS AT time ON date
```

It might also type:

```
LOGON PROCEEDING
```

When you are logged on, TSO types the message:

```
READY
```

and turns the Proceed light on. You can now enter any command.

How to Enter Data

When you key in data at the terminal, the data is stored in a buffer until it is sent to the host computer. The capacity of the buffer in your terminal depends on the model and the features attached; buffer capacities range from 256 to 1024 characters. Your system programmer should give you this information.

You may choose to fill the buffer before transmitting to the host, or your may transmit a partially filled buffer. The topics that follow tell (1) how to transmit a single line of data to the host (2) how to send multiple lines of data in one transmission.

Transmitting a Single Line of Data

1. Make sure that the Auto/Off switch is set to Auto; if the Edit/Off switch is on your terminal, set it to Off.
2. Type a line of input. You may type as much as will fit on one line.
3. Correct any typing errors in the line. This is the only time that corrections can be made before the line is transmitted to TSO in the host computer.
4. Press the Return (↵) key. This transmits the data to TSO, which starts to process it.
5. If you want to enter more input, return to step 2 after the Proceed light goes on and the keyboard unlocks.

Transmitting Multiple Lines of Data

1. Set the Auto/Off switch to the Off position.
2. Type a line of input. You may type as much as will fit on the line.
3. You may correct any typing errors in the line, or you may choose to make corrections as described later under "Buffered SDLC Corrections." All corrections must be made before the data is transmitted to the host.
4. Press the Return (↵) key. This stores the line in your terminal's buffer. When you are storing data in the buffer, any character that is entered in the last ten positions causes the audible alarm to sound. Return to step 2 if you want to store more lines in the buffer before transmitting them to TSO.

5. Transmit the buffer contents to TSO by either:
   - Pressing the EOB key: TSO will hold the data for processing. The Proceed light stays on and you can enter more data immediately (step 2). If the buffer becomes full, the Proceed light goes off and no more data is stored.
   - Pressing the EOM key: TSO will begin processing the data. The Proceed light will go off and then comes on again. If you have more lines to enter, go back to step 2.

Making Corrections

If your 3767 does not have the Edit/Off switch, only single-line (line currently being keyed) correction can be done, (see "Basic SDLC Data Correction," the next topic). If the Edit/Off switch is present, correction of stored data is referred to as editing. The following paragraphs describe editing.

If an edited line is shorter than the old line, following data is shifted ahead. If the edited line becomes longer than the original line, all following data is shifted back to make room. If editing causes a character to go into the last ten buffer positions, the alarm does not sound; if editing causes overflow from the buffer by shifting, the overflow data is lost. (A possible overflow will be apparent when you advance to the end of the stored data by pressing PRINT BUFFER. Any data originally keyed in that does not appear in the printout has been lost.)

Once a Return code (caused by pressing the Return key) has been stored in a buffer, it cannot be removed or replaced with other data. If you want to delete or insert one or more complete lines, you must clear the buffer and key in the entire data block again.

After editing, data transmission begins at the first buffer position and continues to the position where editing stopped. In order to include all of the stored data that is to be transmitted, use a print key (PRINT BUFFER, PRINT LINE, or PRINT CHAR) to advance the editing operation. (See the topic "Buffered SDLC Corrections.")

Caution: The Auto/Off switch must be set to Off when printing from the buffer.

Note: The audible alarm will sound if the stored line is longer than 128 characters; it may sound at the end of the last line printed. Neither of these conditions requires any operator action.

Basic SDLC Corrections

If your 3767 does not have an Edit/Off switch (Buffer with Edit feature), or if the Auto/Off switch is set to Auto, correction is limited to the current line.

To correct the current line:

1. If you are using the Auto View/Off switch, set it to Auto View.
2. Repeatedly press the BUFFR BKSP key until either:
   a. The vertical line on the cut form guide aligns with the first error character., or
   b. The alphanumeric readout (ANR) indicator displays the first error position.
3. Advance the form by rotating the platen knob (to avoid overprinting).
4. Key in the correct data to the end of the line.

To reenter the entire buffer:

1. Press the BUFFR RTN key. The print head returns to the left margin, one line feed takes place and the alphanumeric readout (ANR) displays the value of the left margin.
2. Key in the data from the beginning of the data block.

Buffered SDLC Data Correction

Make sure the Edit/Off switch is set to Edit and the Auto/Off switch is set of Off.

To correct the current line:

1. If you are using the Auto View/Off switch, set it to Auto View.
2. Repeatedly press the BUFFR BKSP key until the vertical line on the cut forms guide aligns with the error character, or until the alphanumeric readout displays the error position. If you backspace too far, repeatedly press (or press and hold) the PRINT CHAR key until the vertical line aligns with the proper position, or until the alphanumeric readout shows the desired value.
3. Advance the form by rotating the platen knob if you want to avoid overprinting.
4. Key in only the corrections.
5. Press the PRINT LINE key if you are doing a character-for-character correction. If you added or deleted data, you must key in the remainder of the line.
6. Continue keying in.

To correct a previous line:

1. Press the BUFFR RTN key to return to the beginning of the buffer.
2. Press the PRINT LINE key once for each line before the error line.
3. Repeatedly press (or press and hold) the PRINT CHAR key until you come to the first error position of the error line.
4. Key in only the corrections.

5. Press the PRINT LINE key if you made a character-for-character correction. If you add or delete data, you must key in the remainder of the line.
6. Press the Return key to indicate the end of editing for the line.
7. Press the PRINT BUFFR (or PRINT LINE) key to advance the edit action to the end of the data originally stored.
8. The buffer now contains correct data; continue the operation.

To reenter the entire buffer:

1. Press the BUFFR RTN key. The print head returns to the left margin, one line feed takes place, and the alphanumeric readout displays the position of the left margin.
2. Key in the data from the beginning of the data block.

To print the current line:

1. Press the BUFFR LINE RTN key. The carrier returns to the left margin and a line feed takes place.
2. Press the PRINT LINE key. The contents of the current buffer line are printed. (Instead of pressing the PRINT LINE key, you can press and hold the PRINT CHAR key until you get to the end of the line.)

To print any line:

1. Count backward from the current line until you reach the line where you want to start printing.
2. Press the BUFFR LINE RTN key as many times as you counted in step 1. The carrier returns to the left margin and a line feed takes place.
3. Press the PRINT BUFFER key. The contents of the buffer are printed, from the beginning of the selected to the last position keyed in.

Instead of pressing the PRINT BUFFR key in step 3, you may press the PRINT LINE key or press and hold the PRINT CHAR key.

To print the entire buffer:

1. Press the BUFFR RTN key. The carrier returns to the left margin and a line feed takes place.
2. Press the PRINT BUFFR key. The contents of the buffer are printed from the beginning.

How to Interrupt Operations from the Terminal

Press the ATTN key (causing an attention interruption) or the CNCL key to signal TSO that you want to interrupt the operation that is taking place. The following operations can be interrupted:

- To stop the transmission of input being sent from your terminal to TSO, press the ATTN key or the CNCL key. (If using buffered SDLC data transmission, pressing CNCL also causes data entered since the last time EOM was pressed to be lost.)

- To stop the listing of output being sent to your terminal from TSO, press the ATTN key or the CNCL key.
- To interrupt a command or program that is executing, press the ATTN key. Pressing ATTN does not affect data at your terminal. It affects the transmission of data and the execution of a command of program.

The ATTN key and the CNCL key never lock, so you can press them at any time. After you press the ATTN key, TSO replies by printing the following response:

```
!
```

You can then proceed with any operation you want to. TSO does not reply when you press the CNCL key, and you can proceed immediately with any operation.

If you want to continue with the operation that you interrupted, press the Return (↵) key before typing anything else; however, input data that was being typed or output data that was being printed at the time of the interruption may be lost.

How to End a TSO Terminal Session

To end a terminal session with TSO, you may either log off or log on. Logging of ends the terminal session with TSO; logging on ends the terminal session with TSO and automatically starts a new session with TSO. (***Note:*** Use the TSO LOGON command, not one of the SNA logon commands, to end a terminal session by logging on.)

If your terminal is supported by TSO/TCAM, your installation can choose to end the SNA session with TCAM automatically when the TSO session ends. If this is the case, follow the procedure below. If your installation selects another option, the installation should provide the proper sequence of commands for logging off.

To log off:

1. Press the SYS REQ key if you are using an SNA logoff command.
2. Enter the logoff command. This command will consist of the word LOGOFF or its installation-defined substitute. Your system programmer should tell you the format of the logoff command for your terminal.
3. Transmit the logoff command to TSO by pressing the Return (↵) key.

Whether logging off or logging on, TSO will type:

```
userid LOGGED OFF TSO AT time ON date
```

Appendix H: IBM 3770 Data Communication System

The IBM 3770 Data Communication System is a family of keyboard-printer terminals and attachable I/O devices (diskette, card reader, card punch, printer) that permits a variety of terminal configurations and is designed for a range of data entry, inquiry, remote printing, and card punching applications. It allows interactive and batch operations.

TSO supports the 3770 as an interactive terminal with EBCDIC or Katakana (Japan only) keyboards, as shown in Figure 41. In addition to the data keys there are function keys, indicator lights, and operating mode switches. The 3771, 3773, and 3774 have bidirectional matrix printers. The 3775 has a line printer with a continuously rotating metal belt.

TSO supports the 3770 terminal in a systems network architecture (SNA) configuration. TSO uses a telecommunications access method, either VTAM or TCAM, to direct the transmission of data over the communication line between the terminal and the host computer.

Before you start your TSO terminal session you must know the type and format of the logon and logoff commands that you should use. Your system programmer should give you this information.

How to Start a TSO Terminal Session

Three operations are involved in starting a TSO terminal session:

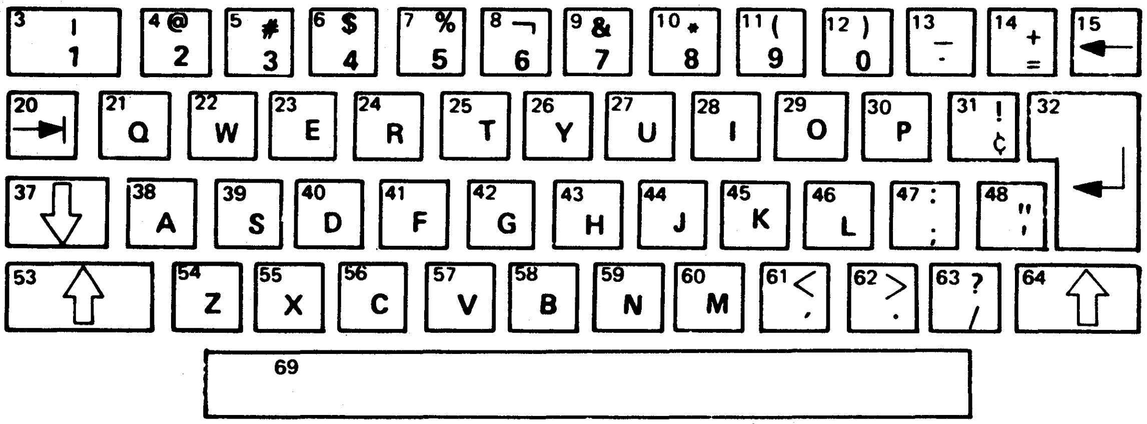
1. Setting the switches
2. Contacting the computer
3. Contacting TSO

Setting the Switches

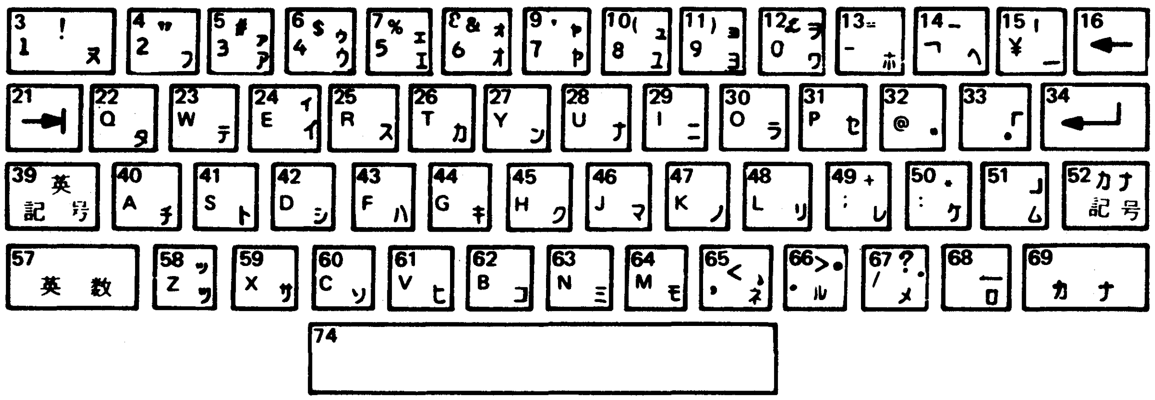
If your terminal has the Security Keylock special feature, insert the key and turn it to the Unlock position.

The switches are located on the operator panel and on the auxiliary operator panel. Set the BSC/SDLC switch to SDLC, then the On/Off switch to On. (You must set the BSC/SDLC switch before you turn the Power switch on because the BSC/SDLC switch takes effect during the power-on sequence; after power is on, changing this switch has no effect.) For a description of each switch, see **IBM 3770 Data Communication: System Components**.

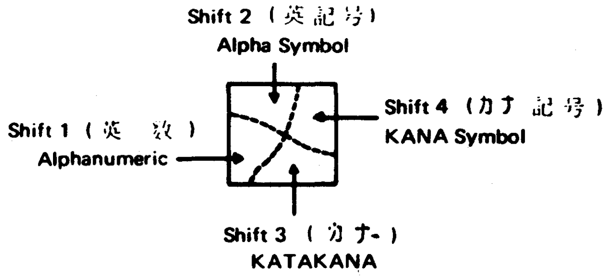
After a 20-30 second delay following power-on, while terminal self-testing occurs, the Proceed light will go on indicating that the terminal is operable.



EBCDIC



16	Back Space
34	New Line
39	Alpha Symbol Shift
52	Katakana Symbol Shift
57	Alphanumeric Shift
69	Katakana Shift
74	Space
9	Apostrophe
13	Minus
14	Over Line
15	Cho-on
68	Under Line
65	Comma
66	Katakana Period



Katakana

Figure 41. IBM 3770 Data Communication System Keyboards

Contacting the Computer

A 3770 terminal can be permanently connected to a computer network through a nonswitched (leased) line, or temporarily connected through a switched (dial-up) line. TCAM supports leased lines only.

If your terminal uses a nonswitched line, you should already be in contact with the computer. Twenty to thirty seconds after you turned the On/Off switch on, the Proceed light should have gone on and the keyboard should have unlocked. If this did not happen, turn the On/Off switch off, then on again. If the Proceed light does not go on and the keyboard does not unlock, the system is not available, and you must try later.

If your terminal uses a switched line, your procedure for dialing the computer depends on the type of dial-up devices you use. Figure 42 shows typical procedures for terminals using an acoustic coupler. Figure 43 shows typical procedures for terminals using a telephone data set (modem). For more detailed instructions, refer to the operating instructions for your specific type of dialing device.

1. Make sure the acoustic coupler:
   (a) is connected to a power supply
   (b) is turned on
   (c) is connected to the terminal
2. Remove the handset from the cradle and dial the computer's telephone number. Your installation should supply the number.
3. Wait for a high-pitched tone. If the number is busy or if there is no answer, hang up and try again.
4. When you hear the high-pitched tone, place the handset face down in the coupler box. Make sure the cord is in the slot. Close the lid of the acoustic coupler and latch it.
5. The keyboard unlocks, and the system is ready to receive input data.

Figure 42. Acoustic Coupler Technique for the IBM 3770 Data Communication System

1. Press the TALK button on the telephone modem, if present; otherwise, set the Talk/Data switch on the terminal to Talk.
2. Remove the handset from the cradle and dial the computer's telephone number. Your installation should supply the number.
3. Wait for a high-pitched tone. If the number is busy or if there is no answer, hang up and try again.
4. When you hear the high-pitched tone, push the DATA button on the telephone modem, if present; otherwise, set the Talk/Data switch on the terminal to Data. The DATA light should go on. The keyboard unlocks, and the system is ready to receive input data. If the DATA light goes off at any time during the terminal session, you must retry from step 1.
5. Place the handset in the cradle.

Figure 43. Telephone Modem Technique for the IBM 3770 Data Communication System

Contacting TSO

If your terminal is supported by TCAM, your installation can provide a LOGON procedure that automatically establishes a SNA session with TCAM. If this option is selected, enter the TSO LOGON command to identify yourself to TSO: type the word LOGON, a space, and your user identification (userid). Then perform steps 3 and 4 below. If this option is not selected, or if your terminal is not supported by TCAM, use the following procedure to contact TSO:

1. Press the SYS REQ key. The KBD and LINE lights go on.
2. Enter a logon command to identify yourself to TSO. The logon command consists of the word LOGON or its installation-defined substitute, and may be followed by a character string containing such information as a user ID, password, account number, and cataloged procedure name. Your system programmer should tell you the format of the logon command for your terminal.
3. Transmit the logon command to TSO by pressing the Return (...) key and, if the Auto/Off switch is set to Off, the EOM key. The Proceed light goes off.
4. Wait for TSO to reply to your logon command. TSO types a preliminary message:

```
userid LOGON IN PROGRESS AT time ON date
```

   It might also type:

```
LOGON PROCEEDING
```

When you are logged on, TSO types the message:

```
READY
```

and turns the Proceed light on. You can now enter any command.

How to Enter Data

There are two modes of sending data from your terminal to TSO: basic SDLC transmission (one line at a time) and buffered SDLC transmission (multiple lines at a time). Basic mode transmits a line of input data to TSO as soon as it is typed. Buffered mode accumulates (or "stacks") lines of input data in your 3770 terminal buffer and then transmits them to TSO. Stacking lines improves performance by decreasing the number of transmissions across the communication line to TSO, thus requiring less processing by TSO.

TSO notifies you that it is ready to accept a line of input by printing a message (for example, READY), a line number (for example, 00120), or a character-prompt (for example,-). Data can now be entered.

Basic SDLC Transmission

To transmit a single line of data do the following:

1. Make sure that your terminal is in auto mode: for 3771 and 3773, press the CODE and 1 keys; for 3774 and 3775, press the top of the Auto switch.
2. Type a line of input. You can type as much as will fit on one line.
3. Correct any typing errors in the line.
4. Press the Return (↵) key. This transmits the data to TSO, which starts processing it.
5. If you want to enter more input, return to step 2 after the Proceed light goes on and the keyboard unlocks.

Note: If the terminal is not in auto mode, data is transmitted automatically after 256 characters are entered, or when the EOB or EOM key is pressed.

Buffered SDLC Transmission

To transmit multiple lines of data do the following:

1. Make sure that your terminal is not in auto mode. If the Auto light is on for 3771 and 3773, press the CODE and 1 keys: for 3774 and 3775, press the bottom of the Auto switch.
2. Type a line of input. You can type as much as will fit on one line.
3. Correct any typing errors (see "Correcting Typing Errors" below).
4. Press the Return (↵) key to stack the line in your terminal's buffer. Return to step 2 if you want to stack more lines in this group.
5. Press the EOB key to transmit the stacked lines to TSO and to have TSO hold the lines for processing, or press the EOM key to transmit the stacked lines to TSO and to signal TSO to start processing them.
6. If you want to enter more input, return to step 2 after the Proceed light goes on and the keyboard unlocks.

   Note: The audible alarm sounds each time a character is entered in the last ten positions of the buffer.

The number of lines you can stack in your buffer (before you press the EOB or EOM key) depends on the size of the buffer in your terminal. Your system programmer should give you this information.

Basic SDLC Data Correction

Correction is limited to the current line if the Auto/Off switch is set to Auto.

To correct the current line:

1. For 3771/3/4, repeatedly press the BUFFR BKSP key until the vertical line on the cut forms guide aligns with the error character. For 3775, repeatedly press the BUFFR BKSP key until the print-position indicator aligns with the error character; print-position lights will go out as you backspace.

2. Advance the form by rotating the platen knob if you want to avoid overprinting.
3. Key in the correct character. If you are replacing one incorrect character with more than one new character, you must reenter the remainder of the line.

To reenter the line:

1. Press the BUFFR RTN key . The print head returns to the left margin and one line feed takes place.
2. Key in the line again.

Buffered SDLC Data Correction

Corrections can be made only if the terminal was *not* in auto mode when the data was entered. Make sure that your terminal is not in auto mode. If the Auto light is on: for 3771 and 3773, press the CODE and 1 keys; for 3774 and 3775, press the bottom of the Auto switch.

To correct the current line:

1. Repeatedly press the BUFFR BKSP key until the vertical line on the cut forms guide (form 3771/3/4) or print-position indicator (for 3775) aligns with the error character. If you backspace too far, repeatedly press (or press and hold) the PRINT CHAR key until the proper position is aligned.
2. Advance the form by rotating the platen knob if you want to avoid overprinting.
3. Key in only the corrections.
4. Press the PRINT LINE key.
5. Continue keying in.

To correct a previous line:

1. Press the BUFFR RTN key to return to the beginning of the buffer.
2. Press the PRINT LINE key once for each line before the error line.
3. Repeatedly press (or press and hold) the PRINT CHAR key until you come to the first error position of the error line.
4. Key in only the corrections.
5. Press the PRINT LINE key if you made a character-for-character correction. If you add or delete data, you must key in the remainder of the line.
6. Press the Return key to indicate the end of editing for the line.
7. Press the PRINT BUFR (or PRINT LINE) key to advance the edit action to the end of the data originally stored.
8. The buffer now contains correct data; continue the operation.

To reenter the entire buffer:

1. Press the BUFFR RTN key. The print head returns to the left margin and one line feed takes place.

2. Key in the data from the beginning of the data block.

To print the current line:

1. Press the BUFFR LINE RTN key. The carrier returns to the left margin and a line feed takes place.
2. Press the PRINT LINE key. The contents of the current buffer line are printed. (Instead of pressing the PRINT LINE key, you can press and hold the PRINT CHAR key.)

To print any line:

1. Count backward from the current line until you reach the line where you want to start printing.
2. Press the BUFFR LINE RTN key as many times as you counted in step 1. The carrier returns to the left margin and a line feed takes place.
3. Press the PRINT BUFFR key, or the PRINT LINE key, or press and hold the PRINT CHAR key. The contents of the buffer are printed, from the beginning of the selected line to the last position keyed in.

To print the entire buffer:

1. Press the BUFFR RTN key. The carrier returns to the left margin and a line feed takes place.
2. Press the PRINT BUFFR key. The contents of the buffer are printed from the beginning.

How to Interrupt Operations from the Terminal

Press the ATTN key (causing an attention interruption) or the CNCL key to signal TSO that you want to interrupt the operation that is taking place. The following operations can be interrupted:

- To stop the transmission of input being sent from your terminal to TSO, press the CNCL key. (If using buffered SDLC data transmission, pressing CNCL also causes data entered since the last time EOM was pressed to be lost.)
- To stop the listing of output being sent to your terminal from TSO, press the ATTN key or the CNCL key.
- To interrupt a command or program that is executing, press the ATTN key.

You can press the ATTN key any time that input can be entered. The CNCL key never locks, so you can press it at any time. After you press the ATTN key, TSO replies by printing the following response:

```
!
```

You can then proceed with any operation you want to. TSO does not reply when you press the CNCL key, and you can proceed immediately with any operation.

If you want to continue with the operation that you interrupted, press the Return (–) key before typing anything else; however, input data that was being typed or output data that was being printed at the time of the interruption may be lost.

How to End a TSO Terminal Session

To end a terminal session with TSO, you may either log off or log on. Logging off ends the terminal session with TSO; logging on ends the terminal session with TSO and automatically starts a new session with TSO. If your terminal is supported by TSO/TCAM, your installation can choose to end the SNA session with TCAM automatically when the TSO session ends. If this is the case, follow the procedure below. If your installation selects another option, the installation should provide the proper sequence of commands for logging off.

To log off:

1. Press the SYS REQ key if you are using a SNA logoff command.
2. Enter a logoff command. This command will consist of the word LOGOFF or its installation-defined substitute. Your system programmer should tell you the format of the logoff command for your terminal.
3. Transmit the logoff command to TSO by pressing the Return (–) key.

Whether logging off or logging on, TSO will type:

```
userid LOGGED OFF TSO AT time ON date
```

Index

OS/VS2 TSO Terminal User's Guide
GC28-0645-4

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate.

IBM shall have the nonexclusive right, in its discretion, to use and distribute all submitted information, in any form, for any and all purposes, without obligation of any kind to the submitter. Your interest is appreciated.

Note: ***Copies of IBM publications are not stocked at the location to which this form is addressed. 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.***

Possible topics for comments are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If comments apply to a Selectable Unit, please provide the name of the Selectable Unit ________.

If you wish a reply, give your name and mailing address:

____

____

____

**Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.**

Cut or Fold Along Line

Please circle the description that most closely describes your occupation.

Customer	(Q) Install Mgr.	(U) System Consult.	(X) System Analyst	(Y) System Prog.	(Z) Applica. Prog.	(F) System Oper.	(I) I/O Oper.	(L) Term. Oper.	(O) **Other**

IBM	(S) System Eng.	(P) Prog. Sys. Rep.	(A) System Analyst	(B) System Prog.	(C) Applica. Prog.	(D) Dev. Prog.	(R) Comp. Prog.	(G) System Oper.	(J) I/O Oper.	(E) Ed. Dev. Rep.	(N) Cust. Eng.	(T) Tech. Staff Rep.

Number of latest Newsletter associated with this publication: ________________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)

GC28-0645-4

Reader's Comment Form

Cut or Fold Along Line

Fold and tape | Please Do Not Staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department D58, Building 706-2
PO Box 390
Poughkeepsie, New York 12602

Fold and tape | Please Do Not Staple | Fold and tape

IBM®

OS/VS2 TSO Terminal User's Guide (S370-39) Printed in U.S.A. GC28-0645-4

GC28-0645-4

OS/VS2 TSO Terminal User's Guide (S370-39) Printed in U.S.A. GC28-0645-4

IBM®