SC28-1333-1
File No. S370-39

Program Product

TSO Extensions User's Guide

Program Number 5665-285

IBM

Second Edition (February, 1986)

This is a major revision of, and obsoletes, SC28-1333-0 and Technical Newsletter SN28-1039. See the Summary of Amendments following the Contents for a summary of the changes made to this book. Technical changes to the text are indicated by a vertical line to the left of the change.

This edition applies to TSO Extensions Release 2, Program Number 5665-285 and to all subsequent releases until otherwise indicated in new editions or technical newsletters. Changes are made periodically to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370 Bibliography*, GC20-0001, for the editions that are applicable and current.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM program product in this publication is not intended to state or imply that only IBM's program product may be used. Any functionally equivalent program may be used instead.

Publications are not stocked at the address given below. Requests for 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, Information Development, Department D58, Building 921-2, PO Box 390, Poughkeepsie, N.Y. 12602. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Preface

This publication describes how to use the Time Sharing Option (TSO) command language to handle data and to compile, link edit, load and execute programs at a terminal. It is intended as a guide for all TSO users, from those unfamiliar with TSO to experienced users who need further information to perform a specific task. This manual can be used at a terminal or as a tutorial manual before a new user begins work with TSO. System programmers responsible for generating and maintaining a system with TSO should consult *System Programming Library: TSO Extensions.*

Your installation may have the Interactive System Productivity Facility/Program Development Facility (ISPF/PDF). These IBM program products provide panels that prompt you for information as an interface between you and TSO. Behind the ISPF/PDF panels, the TSO commands are processed as if you had entered them individually at your terminal. Most TSO tasks can be accomplished with or without ISPF/PDF. If you are using ISPF/PDF, see *Interactive System Productivity Facility/Program Development Facility for MVS: Reference.*

Organization

This book is divided into three sections. Section One describes information that every TSO user must understand, such as the concept of data sets and the data set naming conventions, how to specify TSO commands, and how to communicate with other TSO users. Section Two describes the use of the TSO command language to allocate, rename, copy, protect, free and delete data sets. Section Three describes how to compile, link edit, load and execute programs in the foreground and background.

Related Publications

The publications referred to in this book are:

MVS/Extended Architecture

Refer to *MVS/System Product Version 2 General Information Manual,* GC28-1118 for the order numbers of these MVS/XA books on the level that you are using.

MVS/Extended Architecture Access Method Services
MVS/Extended Architecture JCL
MVS/Extended Architecture System Programming Library: System Macros and Facilities, Volume 1

MVS/Extended Architecture System Programming Library: System Macros and Facilities, Volume 2
MVS/Extended Architecture Message Library: System Messages
MVS/Extended Architecture System Programming Library: TSO
MVS/Extended Architecture Message Library: TSO Terminal Messages

MVS/370

Refer to *MVS/System Product Version 1 General Information Manual*, GC18-1025 for the order numbers of the MVS/370 books on the level that you are using.

MVS JCL
OS/VS Message Library: VS2 System Messages, GC28-1002
OS/VS Assembler Language, GC33-4010
OS/VS2 Access Method Services, GC26-3841
OS/VS2 System Programming Library: Supervisor, GC28-1046
OS/MVT and OS/VS2 TSO Data Utilities: COPY, FORMAT, LIST, MERGE User's Guide and Reference, SC28-6765

MVS/370 or MVS/Extended Architecture

System Programming Library: TSO Extensions Planning and Installation Volume 1, SC28-1379
System Programming Library: TSO Extensions User Exits and Modifications Volume 2, SC28-1380
System Programming Library: TSO Extensions Command and Macro Reference Volume 3, SC28-1381
TSO Extensions CLISTs: Implementation and Reference, SC28-1304
TSO Extensions Command Language Reference, SC28-1307
TSO Extensions Guide to Writing a Terminal Monitor Program or Command Processor, SC28-1136
TSO Extensions Session Manager Program Reference, SC28-1306
TSO Extensions Session Manager Terminal User's Guide, SC28-1305

IBM 3790 Communication System Operator's Guide for the 2741 Communications Terminal, GA27-2783
IBM 3270 Information Display System Operator's Guide, GA27-2742
IBM 3290 Information Panel Description and Reference GA23-0021
IBM 3767 Communication Terminal Operator's Guide, GA18-2000
Introduction to the IBM 3770 Data Communication System, GA27-3144
OS/MVT and OS/VS2 TSO Terminals, GC28-6762

Assembler H Version 2 Application Programming: Language Reference, GC26-4037
OS/TSO Assembler Prompter User's Guide, SC26-3740
IBM OS (TSO) COBOL Prompter Terminal User's Guide and Reference, SC28-6433
IBM System/360 OS (TSO) Terminal User's Supplement for FORTRAN IV (G1) Processor and TSO FORTRAN Prompter, SC28-6855
OS PL/I Optimizing Compiler: TSO User's Guide, SC33-0029

Interactive System Productivity Facility/Program Development Facility for MVS: Reference, SC34-2089

Resource Access Control Facility Command Language Reference, SC28-0733

Program Products

The IBM Program Products referred to in this book are:

- Interactive System Productivity Facility (ISPF), 5668-960 or 5665-319
- Interactive System Productivity Facility/Program Development Facility (ISPF/PDF), 5665-268 or 5665-317

- OS/MVT and OS/VS2 TSO Data Utilities, 5734-UT1

- Resource Access Control Facility (RACF), 5740-XXH

- TSO Extensions (TSO/E), 5665-285

- Assembler H Version 2, 5668-962
- COBOL TSO Prompter, 5734-CP1
- OS/VS COBOL Release 2.4, 5740-CB1
- PL/I OS Optimizing Compiler, 5734-PL1
- PL/I OS Checkout Compiler, 5734-PL2
- TSO Assembler Prompter, 5734-CP2
- TSO FORTRAN Prompter, 5734-CP3
- VS BASIC, 5748-XX1
- VS FORTRAN, 5748-F03

Contents

Section III: Preparing and Running a Program

Figures

Summary of Amendments

Summary of Amendments
for SC28-1333-1
for TSO Extensions Release 2

There are minor technical and editorial changes throughout the book. Information about the TSO service facility has been rewritten and the following information has been added: handling error conditions, using the RECEIVE command in background jobs, and using the SMCOPY command.

Summary of Amendments
for SC28-1333-0
as Updated December 14, 1984
By Technical Newsletter SN28-1039

This technical newsletter contains changes to the description of attention interrupt.

Section I: General TSO Functions

TSO is a tool you can use to interact with MVS to process data. Programs, commands, and command procedures are the vehicles you use in a TSO environment to process data. This section describes:

- Using your terminal to communicate with TSO
- The data sets that contain the information you want to process
- The general format and syntax of the TSO commands
- Communicating with other TSO users.

Chapter 1. Using Your Terminal

Your terminal is your link to the computer system using TSO. Before you start to use TSO, you need to understand how to use your terminal and how to tailor its capabilities to your needs. This chapter discusses:

- Where to look for information on using your terminal
- How to log on and off of TSO
- How using session manager can record your terminal session
- What system messages can tell you
- How to use an attention interrupt
- How to define or alter your terminal characteristics
- How to define or alter your user profile.

Learning About Your Terminal

You can use TSO on any device which supports MVS. For example, TSO is supported on the following types of commonly used terminals:

IBM 2741 Communication Terminal
IBM 3270 Information Display System
IBM 3290 Information Panel
IBM 3767 Communication Terminal
IBM 3770 Data Communications System

A terminal session is relatively simple. A terminal user identifies himself to the system and then issues commands to request work from the system. In order to conduct a TSO terminal session you need to:

- Understand how to use your terminal, including how to enter data, correct mistakes, add or delete characters and lines, and how to interrupt operations from the terminal
- Log on to the computer system you will be using
- Contact TSO
- End the TSO terminal session (logoff).

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. To learn how to use your terminal, consult the terminal operators manual that

accompanies your device. Some frequently used terminal operators guides are listed below:

- *IBM 3790 Communication System Operator's Guide for the 2741 Communications Terminal*
- *IBM 3270 Information Display System Operator's Guide*
- *IBM 3290 Information Panel Description and Reference*
- *IBM 3767 Communication Terminal Operator's Guide*
- *Introduction to the IBM 3770 Data Communication System* - contains list of operating procedures guides for the various devices in this series
- *OS/MVT and OS/VS2 TSO Terminals*

Because each installation differs in its access methods, configuration, and procedures, you will have to consult your system programmer or system support center to learn how you can contact your computer system, and begin using TSO.

LOGON/LOGOFF Command

Use the LOGON command to identify yourself to the system and request use of its resources.

When you use the LOGON command, the system requires two pieces of information. They are:

1. Your user ID, which tells the system who wants to use it.
2. The password, which authorizes you to use the systems resources.

Accessing the System

You can issue the LOGON command with no operands. This results in the system prompting you for your user ID and your password, if one is used with your system. For example, to simply request access to the system, enter:

```
LOGON
```

You can also issue the LOGON command followed by your user ID. Separate your user ID from the command with a blank character. Once you enter this command, the system prompts you for your password. For example, if your user ID is D00ABC1, you would enter:

```
LOGON D00ABC1
```

Each installation determines what appears on your screen after you issue the LOGON command. You will see one of three things. They are:

- A full screen menu, where you can type in required information. See Appendix A, "Full Screen Logon Processing" in this book for further information on full screen menus.
- A message or messages, prompting you for required information.
- A READY message, indicating that the system recognized your user ID and password, and is now at your disposal.

Your installation might require you to specify other information with the LOGON command; see your system programmer for LOGON instructions.

Use the LOGOFF command to sign off from the system and end your terminal session. For example, to sign off from the system, enter:

```
LOGOFF
```

The system releases your user ID until the next time you issue the LOGON command.

You can also sign off from the system by issuing the LOGON command. When you sign off using the LOGON command, the system terminates your previous session and starts a new one using the options specified on the LOGON command. For more information about the LOGON and LOGOFF commands, see *TSO/E Command Language Reference*.

Using Session Manager

The TSO session manager is part of the TSO/E program product. The session manager keeps a complete journal of everything that happens during your terminal session. It records everything you type in and everything the system displays. Any time during your terminal session, you can look at any of the work you did in the beginning, middle, or at the end of your session. The session manager also lets you print a copy of this information.

The session manager operates as an interface to TSO. You invoke session manager as part of your logon procedure. Consult your system programmer to learn how to invoke session manager at your installation.

For more information about using session manager, see *TSO/E Session Manager Program Reference* and *TSO/E Session Manager Terminal User's Guide.*

Messages

You can receive four types of messages at your terminal:

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

Mode Messages

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

```
READY
```

The EDIT, OUTPUT and TEST commands have subcommands associated with them which you can use only after you issue the command. After you enter EDIT, OUTPUT or TEST and the system is ready to accept that command's subcommands, the mode message is the name of the command, instead of READY. The mode messages indicating the system is ready to accept a valid subcommand are:

```
EDIT
OUTPUT
TEST
```

The TEST message also appears after each TEST subcommand has been processed. If the system has to display any output or other messages as a result of the previous command or TEST subcommand, it does so before displaying the mode message. The EDIT and OUTPUT messages are not displayed after **each** subcommand executes. For more information about when mode messages for subcommands appear, see *TSO/E Command Language Reference.*

As long as you enter commands and subcommands correctly, you can sometimes save time by entering two or more commands in succession without waiting for

the intervening mode message. The system then prints the mode messages in succession after the commands. For example, if you enter the DELETE, FREE and RENAME commands without waiting for the intervening mode messages, your display is:

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

Use care when entering commands without waiting for the intervening mode messages. If you make a mistake in one of the commands, the system sends you messages telling you of your mistake, then cancels the remaining commands you entered. After you correct the error, you have to reenter the other commands. Therefore, unless you are sure your input is correct, wait for a READY message before entering a new command.

Some terminals "lock" the keyboard after you enter a command, so that when you press the keys, you are not communicating with the system. You cannot enter commands until you get a READY message. Terminals that do not normally lock the keyboard might occasionally do so, for example, when all buffers allocated to the terminal are full. See your terminal operator's guide for information about the terminal you work on.

Prompting Messages

A prompting message indicates that you did not supply required information or that you supplied incorrect information. A prompting message asks you to supply or correct that information. For example, a partitioned data set name is a required operand of the CALL command. If you enter the CALL command without that operand, the system prompts you for the data set name. Your display looks like this:

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

Respond by typing the requested information, in this case the data set name, and pressing the ENTER key. For example, if the data set name is **ALPHA.DATA**, you would respond to the prompting message as follows:

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

You can choose to receive prompting messages when appropriate. You can use the PROFILE command to control prompting. For more information on the PROFILE command, see "PROFILE Command" later in this chapter.

Some messages have additional information available that you can choose to request.

To request an additional level of a message:

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

If the second message is not enough, you can request a further message to give you even more detail. An indication that a second or additional message levels are available is a plus sign (+) at the end of the message. However, not all prompting messages that have further levels of prompting information are followed by a plus sign (+). When unsure of how to respond to a message, continue requesting additional messages, regardless of whether or not the previous message ended with a plus sign (+).

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

```
NO INFORMATION AVAILABLE
```

You can stop a prompting sequence by entering the requested information or by requesting an attention interrupt to cancel the command.

Informational Messages

An informational message tells you about the status of the system or your terminal session. For example, an informational message can tell you how much processor 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, as described in "Prompting Messages" in this section. Like prompting messages, informational messages may have two or more levels.

Broadcast Messages

Broadcast messages are messages that an operator or a user sends to another user via the SEND command. Broadcast messages are messages of general interest to users of the system. The system operator can send messages to all users of the system or to individual users. For example, an operator might send the following message to all users:

```
DO NOT USE TERMINALS #4, 5 AND 6 ON 6/30.  THEY ARE
RESERVED FOR DEPARTMENT D04.
```

When you receive a broadcast message on a display terminal, the screen you are working on disappears and a screen with the message appears. After reading the message, you can return to your work screen without losing any of your previous work by pressing the ENTER key. If you are using session manager, the message will be incorporated into the record of your terminal session.

You can also display broadcast messages at your terminal by issuing a LISTBC command, which lists the contents of the SYS1.BRODCAST data set. The SYS1.BRODCAST data set contains messages of general interest (NOTICES) that are sent from the system to all terminals and messages directed to a particular user (MAIL).

You, or any other user, can send messages to other users or to the system operator.

A message sent by another user shows the user identification so you know who sent the message. For more information on how to communicate with other TSO users, see Chapter 4, "Communicating With Other Users" later in this book.

The Attention Interrupt

The attention interrupt allows you to interrupt processing at any time so that you can enter a command or subcommand, postpone further processing, or terminate processing altogether. For instance, if you are executing a program and the program gets in a loop, you can use the attention interrupt to halt execution. As another example, when you are having data listed at your terminal and you would like to know what time it is, you can request an attention interrupt and issue the TIME command, and then resume execution.

You can also request an attention interrupt while the system processes a command. When the system is processing a command and you request an attention interrupt, you can enter the TIME or TEST command, and then resume with the interrupted operation by pressing the ENTER key. Entering any command other than TIME or TEST causes current processing to terminate. One output record from the interrupted program may be displayed at the terminal after you enter your next command. This is normal for some programs.

If, after causing an attention interrupt, you want to continue with the operation that you interrupted, simply press the ENTER key before typing anything else. Keep in mind however, that data you were entering or displaying at the time of the attention interrupt might be lost.

If you request an attention interrupt after issuing a CANCEL, STATUS, or SUBMIT command, the attention interrupt might terminate that command's processing. In this case, pressing the ENTER key would not cause command processing to continue. For example, if the attention interrupt terminated processing by the SUBMIT command, you would not see the SUBMIT job name displayed.

Note: If you are using session manager, press the **ERASE EOF** key and then press the **ENTER** key to enter a null line to resume execution.

PROFILE Command

Your user profile defines how you want the system to respond to information sent to or from your terminal when performing data processing tasks. Use the PROFILE command to change or list your user profile.

An authorized system programmer creates your user ID and your user profile. The typical user profile is defined by the default values of the PROFILE operands discussed under this command. A typical user profile might look like this:

```
CHAR(0) LINE(0) PROMPT INTERCOM NOPAUSE NOMSGID NOMODE
NOWTPMSG NORECOVER PREFIX(USERID)
DEFAULT LINE/CHARACTER DELETE CHARACTERS IN EFFECT FOR THIS
TERMINAL
```

You can use the PROFILE command to change or list your user profile. When you use the PROFILE command to change your profile, only the characteristics that you explicitly specify using the operands of this command are changed, the others remain as they are. The new characteristics you specify usually remain unchanged from one session to another. If the changes you requested do not remain in effect from one session to another, your installation might have a special feature that prevents this. For example, you installation might have a LOGON pre-prompt exit that prevents the system from saving any changes to your user profile. You can check with the system programmer to learn if this applies at your installation.

Using the PROFILE command, you can specify:

- On terminals without a delete key, the characters that when typed in, cause the system to delete a single character or an entire line
- Whether or not you want to be prompted for message responses
- Whether or not you want to accept messages sent from other terminal users
- Whether or not you want the opportunity to obtain additional information about messages issued to your terminal while you execute a command procedure
- Whether or not you want the message numbers associated with diagnostic messages that may be displayed at your terminal
- A particular user ID or character string to be used instead of your own user ID as the first qualifier of all non-fully qualified data set names
- Whether or not you want to display at you terminal all all messages issued to you by a program (write-to-programmer messages).

In the following examples, you will see either no operands or one operand entered with each example of the PROFILE command. However, you may enter any number of non-conflicting operands on a PROFILE command.

For more information about the PROFILE command, see *TSO/E Command Language Reference.*

Displaying Your Current User Profile - the LIST Operand

You might want to display your current user profile to find out what options it contains. To list the current characteristics of your user profile, enter the PROFILE command with no operands or use the LIST operand with no other operands.

EXAMPLE

```
To list the characteristics of your current profile, enter:

PROFILE
    or
PROFILE LIST
```

If you specify other operands with the LIST operand, the system first changes your user profile according to the other operands you specify, and then lists the changed user profile.

Specifying a Character Deletion and a Line Deletion Character - the CHAR and LINE Operands

Some terminals have a special key set up to tell the system to delete the character above the cursor. For example, on 3270 Display Stations, the delete key has **DEL** written on it. On terminals without a specific character deletion key, the backspace key is specified for character deletion by the default value on the PROFILE command of CHAR(BS). You can choose a different character to use for character deletion, by using the CHAR operand. To specify a character deletion key, enter the PROFILE command:

- Specify the CHAR operand.
- Immediately enclose in parentheses the character to function as a delete key.

EXAMPLE

```
To use $ as the character deletion key, enter:

PROFILE CHAR($)
```

To specify that no character be used as a character deletion key, use the NOCHAR operand:

```
PROFILE NOCHAR
```

Some terminals have a special key set up to tell the system to delete the remainder of a line. For example, on 3270 Display Stations, the line delete key has **ERASE EOF** written on it. On terminals that do not have a specific line deletion key, the attention key is specified as the default for line deletion by the operand LINE(ATTN). You can chose a different character for line deletion by using the LINE operand. To specify a character deletion key, enter the PROFILE command:

- Specify the LINE operand
- Immediately enclose in parentheses the character to function as a line deletion key.

EXAMPLE

To use # as the line deletion key, enter:

```
PROFILE LINE(#)
```

When you specify a character or line deletion character, avoid specifying a blank, tab, comma, asterisk or parentheses because these characters are used when entering commands. Do not use alphabetic or other commonly-used characters as character deletion or line deletion characters.

If you enter an invalid character and/or line deletion character on the PROFILE command, an error message informs you which character is invalid; the character or line deletion field in the user profile table is not changed. You may continue to use the old character or line deletion character.

To specify that the system not use a character as a line deletion character, enter:

```
PROFILE NOLINE
```

Requesting to be Prompted by the System - the PROMPT/NOPROMPT Operand

If you enter invalid information with a command, or the system requires further information, the system will prompt you for the correct information. The PROMPT operand, which allows you to receive these messages, is a default with the PROFILE command.

To request that the system not prompt you for missing or invalid information, use the NOPROMPT operand. For example:

```
PROFILE NOPROMPT
```

Receiving Messages from Other Users - the INTERCOM/NOINTERCOM Operand

By using the INTERCOM or NOINTERCOM operands, you have the choice of specifying whether or not you want to receive messages sent from other terminal users. INTERCOM, allowing you to receive messages from other users, is the default value with the PROFILE command.

To specify that you do not want to receive messages from other users, specify the NOINTERCOM operand. For example:

```
PROFILE NOINTERCOM
```

Obtaining Additional Diagnostic Information - the PAUSE/NOPAUSE Operand

Use the PAUSE operand to obtain additional levels of information about a message. When the system issues a message with additional levels of information, the system displays the word **PAUSE**, and waits for you to either press the ENTER key or enter a question mark (?). When you enter a question mark, the system displays the next level of information associated with the message. When you simply press the ENTER key, the system resumes processing where it left off. For example, to request that the system give you the opportunity to obtain additional diagnostic information while executing a CLIST, enter:

```
PROFILE PAUSE
```

If you do not want the opportunity to obtain additional information about messages issued to your terminal use the NOPAUSE operand. Note that NOPAUSE is the default value for your profile when your profile is created.

Displaying Message IDs with Messages- the MSGID/NOMSGID Operand

To include the message identifier for the diagnostic messages that appear at your terminal, enter the MSGID operand. With the message identifier, you can look up the correct message in *TSO Terminal Messages*.

EXAMPLE

If the message, **ENTER RESTORE PARAMETERS OR 'DELETE' OR 'END'** appears on your screen, and you wanted the message identifier (INMR906A) displayed also, enter:

```
PROFILE MSGID
```

Note that NOMSGID is the default value for your profile when your profile is created.

Specifying a Data Set Name Prefix - the PREFIX/NOPREFIX Operand

To request the system use a specific user ID or other valid string as the first qualifier of all non-fully qualified data set names, use the PREFIX operand followed by the user ID or string enclosed in parentheses.

EXAMPLE

To use D00ABC1 as the first-level qualifier of all non-fully qualified data sets, enter:

```
PROFILE PREFIX(D00ABC1)
```

If you do not want to append a prefix to all non-fully qualified data sets, enter:

```
PROFILE NOPREFIX
```

The default prefix for foreground processing is your user ID. No prefix is the default for background processing.

Receiving Write-to-Programmer Messages - the WTPMSG/NOWTPMSG Operand

Some programs issue messages to the programmer. To receive all write-to-programmer messages at your terminal, use the WTPMSG operand. For example, to ensure that you can receive all write-to-programmer messages, enter:

```
PROFILE WTPMSG
```

NOWTPMSG, which specifies that you do not want to receive any write-to-programmer messages, is the default value for your profile when your profile is created.

TERMINAL Command

Most installations provide a default TERMINAL command through the logon procedure (PROC) in the LOGON command. The options specified with a TERMINAL command remain in effect until the end of the session or until another TERMINAL command is issued. Use the terminal command to define or alter the operating characteristics of your terminal. The values you can specify with the operands on the TERMINAL command depend on the type of terminal you are using.

Using the TERMINAL command, you can specify:

- The maximum number of characters allowed per line of your terminal
- The screen dimensions of an IBM 3270 display station.

When you define or alter your terminal characteristics, the options you request remain in effect until you enter the LOGOFF command. If you terminate a TSO session and begin a new session by entering the LOGON command, (instead of a LOGOFF command followed by a LOGON command) the terminal characteristics defined in the earlier session are used during the subsequent session.

If a line disconnection interrupts your session, and you log on again using the LOGON command with the RECONNECT option, the terminal characteristics defined earlier in the session are not saved. You must redefine your terminal characteristics because all records for defined data are lost as a result of a line disconnection.

The TERMINAL command is not allowed as a TSO command in the background. For a description of foreground and background execution, see Chapter 17, "Executing Foreground Commands from a Background Job."

In the following examples, you will see only one operand entered with each example of the TERMINAL command. You may enter any number of non-conflicting operands on a TERMINAL command.

For more information about the TERMINAL command, see *TSO/E Command Language Reference.*

Specifying the Maximum Characters Per Line - the LINESIZE Operand

To specify the maximum number of characters allowed on one line of your terminal, use the LINESIZE operand followed by a number enclosed in parentheses. For example, to specify 40 as the maximum number of characters the system can display on one line of your terminal, enter:

```
TERMINAL LINESIZE(40)
```

The LINESIZE operand is not applicable to IBM 3270 Display Stations.

If you enter LINESIZE(80) and try to view a data set with its record format attribute (RECFM) sat to U, specifying records of undefined length, the line is

truncated at the 79th character. The truncated byte (80) is reserved for an attribute character.

Specifying Your Terminal's Screen Size - the SCRSIZE Operand

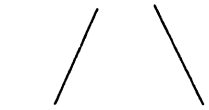
To specify the maximum number of horizontal rows of characters per screen and the maximum number of characters per row, enter the TERMINAL command:

- Specify the SCRSIZE operand
- Enclose in parentheses the number of rows per screen, and the number of characters per row, separated by a comma.

EXAMPLE

To specify a maximum of 32 rows of data where each row is a maximum of 80 characters as the dimensions of the screen of an IBM 3270 Display Station, enter:

```
TERMINAL SCRSIZE(32,80)
```



```
rows      maximum characters
```

Your installation defines the default values for the screen sizes when it defines the terminal characteristics.

Chapter 2. Data Sets

A data set is a unit of information that can be stored and retrieved. It is organized in one of several prescribed arrangements and is described by control information which the system can access. The two types of data sets most often used with TSO are sequential data sets and partitioned data sets. In a sequential data set, the data is arranged in a sequence, from top to bottom or beginning to end. A partitioned data set is divided into separately named, independent partitions called members, each of which can contain data.

Examples of types of data a TSO user might put in a data set are:

- A program's source code
- Job control language statements
- Input to a program
- Output from a program
- Memos
- A CLIST
- Text of a report.

This chapter describes how to name data sets, how to use data set names when processing data, and how to edit data sets.

TSO Data Set Naming Rules and Conventions

Each data set is identified by a unique data set name. When naming data sets, you must follow certain rules:

- A data set name consists of one or more parts connected by periods. Each part is called a qualifier.
- Each qualifier must begin with an alphabetic character (A - Z) or national character (@, #, $).
- The remaining characters in each qualifier can be alphabetic characters, digits (0-9) or national characters.
- Each qualifier must be between one and eight characters in length.
- The maximum length of a complete data set name is 44 characters, including the periods.

Some examples of valid data set names are:

```
PARTS
PARTS.DATA
D00ABC1.DA.PARTS.DATA
```

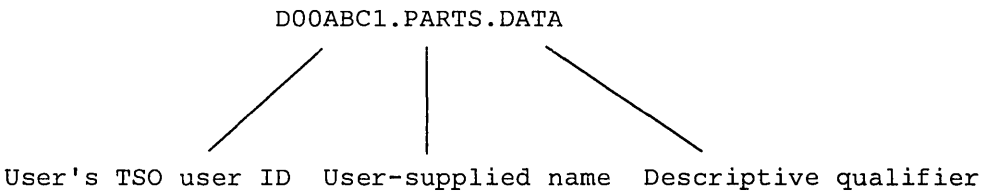
In addition to the rules, you can make use of certain naming conventions that will make TSO easier for you to use. These conventions are:

- Data set names consist of three qualifiers.
- The first qualifier of each data set name is your TSO user ID, or a prefix specified using the PROFILE command.
- The second qualifier of each data set name is your choice; it should be a meaningful name to you.
- The third qualifier is a descriptive qualifier implying certain characteristics of the data. See the list of descriptive qualifiers below:

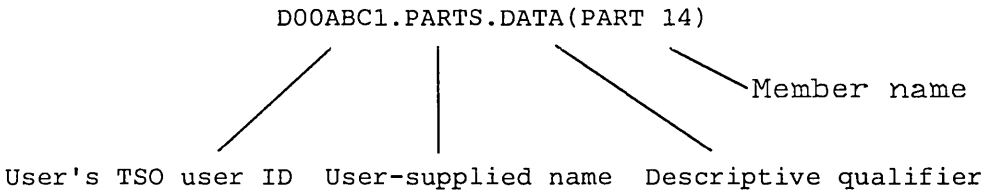
Descriptive Qualifier	Data Set Contents
ASM	Assembler (F) input
CLIST	TSO commands and CLIST statements
CNTL	JCL and SYSIN for SUBMIT command
COBOL	American National Standard COBOL statements
DATA	Uppercase text
FORT	FORTRAN (G1, H) statements
LINKLIST	Output listing from linkage editor
LIST	Listings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

Figure 2-1. Descriptive Qualifiers

A data set name that consists of a user ID, a user-supplied name, and a descriptive qualifier is a "fully qualified" data set name. A fully qualified data set name looks like:



When you refer to partitioned data sets, enclose the member name in parentheses immediately following the descriptive qualifier. A fully qualified partitioned data set name looks like:



You do not have to use the conventional descriptive qualifiers when naming a TSO data set. However, when a data set name adheres to the conventions, you can refer to the data set by an abbreviated version of the name, and the system will supply the rest of the name. Also, certain TSO procedures expect to find data in specific types of data sets.

You must enclose the data set name in single quotation marks if you specify:

- A fully qualified data set name with a user identification (leftmost qualifier) that is not your own
- A data set name that does not conform to the naming conventions.

For example:

'D00ABC1.PROG.LIST'

where D00ABC1 is **not your** user identification

'D00ABC1.PROG.FIRST'

where FIRST is **not** a valid descriptive qualifier in this case

How to Enter Data Set Names

The data set naming conventions simplify the use of data set names. If the data set name conforms to the conventions, you need to specify only the user-supplied name field when you refer to the data set on commands. The system adds the necessary qualifiers to the beginning and to the end of the name that you specify. In some cases, however, the system prompts you for a descriptive qualifier.

When you specify only the user-supplied name, the system adds the prefix specified in your user profile, usually your user identification and, whenever possible, a descriptive qualifier. The system attempts to derive the descriptive qualifier from available information.

If you do not specify enough information, the system issues a message at your terminal requesting the required information. For example, when user D00ABC1 enters the following:

```
EDIT SAMPLE
```

the system responds with:

```
ENTER DATA SET TYPE -
```

The user should then respond by typing in a valid data set qualifier. For example:

```
TEXT
```

The user can now edit data set D00ABC1.SAMPLE.TEXT.

The system will not append qualifiers to any name enclosed in single quotation marks.

EDIT Command

Use the EDIT command and its subcommands to modify the contents of a data set. Using the EDIT command, you can:

- Add data
- Delete data
- Change data
- Move data
- Copy data
- Control line numbers
- Locate data
- Display data
- Control tab settings.

There are two main facilities available to you for modifying the contents of a data set. They are the TSO EDIT command and the EDIT option of the Interactive System Productivity Facility (ISPF). If you have ISPF installed at your location, you will probably want to use the ISPF EDIT option because it is a full-screen editor. The TSO EDIT command is a line mode editor, and most often used in CLISTs. Line mode means that the data is displayed at the terminal one line at a time and you can access data only through commands. In full screen editing, an entire screen of data is displayed at once and you can access data through commands or by using the cursor.

For more information on the TSO EDIT command, see *TSO/E Command Language Reference.*

For more information on how to perform edit functions when executing a CLIST, see *TSO Extensions CLISTs: Implementation and Reference.*

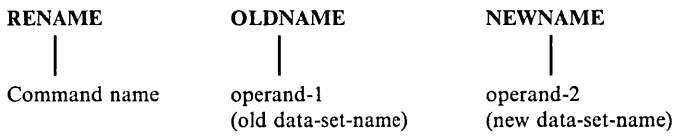
For information on how to use the ISPF EDIT facility, see *Interactive System Productivity Facility/Program Development Facility for MVS: Reference.*

Chapter 3. 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 command 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 form.

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



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

Positional Operands

Positional operands immediately follow the command name in a certain order. If you enter positional operands incorrectly, you will get an error message. Usually you supply a name or value for a positional operand. In some cases, you must specify both the operand and its value. The examples in this publication show actual values in place of positional operands. For example:

```
EDIT TFKNU47.DATA
```

```
where TFKNU47.DATA is the data-set-name positional operand
with the EDIT command
```

To enter a positional operand that is a list of several names or values, enclose the list within parentheses. The names or values must not include unmatched right parentheses. For example,

```
LISTDS (PARTS.DATA TEST.DATA)
```

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 some cases you may specify values with a keyword. You must enter the value within parentheses following the keyword. A typical keyword operand with a value could be described as the LINESIZE operand of the TERMINAL command:

```
TERMINAL 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:

```
TERMINAL LINESIZE(80)
```

If you enter conflicting, mutually exclusive 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. (If you supply an abbreviation which is not specific enough, the system will prompt you.) For instance, the LISTBC command has four keywords:

```
MAIL
NOMAIL
NOTICES
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)
```

Delimiters

When you type a command, separate the command name from the first operand by one or more blanks. Separate operands by one or more blanks or a comma. For example, you can type the LISTBC command like this:

```
LISTBC NOMAIL NONOTICES
        or
LISTBC NOMAIL,NONOTICES
        or
LISTBC NOMAIL     NONOTICES
```

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

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

Line Continuation

You might want to issue a command that does not fit on one line, making it necessary to continue the command on the next line. Use a plus or minus sign as the last character of the first line to continue the command onto the next line. A plus sign causes leading delimiters to be removed from the continuation line. For example, typing in:

```
ALLOC DA(DSN1.PLI) NEW   /* allocate the first of +
                        my data sets */
```

causes the line to appear to the system as:

```
ALLOC DA(DSN1.PLI) NEW  /* allocate the first of my data sets */
```

A minus sign causes leading delimiters to be kept with the continuation line. For example, typing in:

```
ALLOCATE DATASET(OUTPUTDS.TEXT)-
         NEW VOLUME(TSOMAR2)
```

causes the line to appear to the system as:

```
ALLOCATE DATASET(OUTPUTDS.TEXT)          NEW VOLUME(TSOMAR2)
```

Note: When TSO encounters a line plus sign or minus sign at the end of a line with no following line, it waits for you to add a new line.

Comments

You can include comments in a command anywhere a blank might appear. Simply enter them within the comment delimiters /* and */. To continue a comment, use a line continuation character (+ or -) at the end of the line.

```
listd (data-set-list) /* my data sets */

            or

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

You do not need to end a comment with */. Ending a comment with */ is a convention, not a requirement.

Note: To continue a line that contains a comment, use a continuation character *after* the comment:

```
ALLOCATE DATASET(OUTDS.TEXT) /* data set name */ +
         NEW VOLUME(TSOMAR2)
```

Subcommands

Three TSO commands, EDIT, OUTPUT, and TEST have subcommands associated with them. When you issue EDIT, OUTPUT, or TEST, the system responds by putting you into either EDIT, OUTPUT, or TEST mode and issuing a mode message. You can then enter a subcommand to specify the particular operation that you want performed. You can continue entering subcommands until you enter the END subcommand. Subcommands are very similar to commands in that they can have positional and keyword operands.

After you receive the mode message from the system, you can execute any of the subcommands associated with the command that put you in that mode. For example, your terminal session could look like this:

```
READY
edit book.text emode
EDIT
top
save
EDIT
end
READY
```

where

- READY and EDIT are mode messages.
- EDIT is a command. EMODE is an operand of the EDIT command specifying that EDIT should start in EDIT mode, not INPUT mode.
- Top, save and end are EDIT subcommands.

Executing TSO Commands

You execute TSO commands or subcommands by:

- Entering them at your terminal
- Submitting them in a job using JCL statements
- Including them in a CLIST.

To enter a command at your terminal, type the command and the operands (if any), then press the ENTER key.

To enter a command using JCL statements, see Chapter 17, "Executing Foreground Commands from a Background Job."

CLISTs are groups of TSO commands and CLIST statements that are stored in data sets having a descriptive qualifier of CLIST. You execute a CLIST by using the EXEC command or by typing the CLIST name preceded by a percent sign (%) on the command line. The system executes the commands within a CLIST in the order in which they appear in the data set. For more information concerning CLISTs, refer to *TSO Extensions CLISTs: Implementation and Reference.*

Examples of the commands and their operands in this book are in upper case type. In examples of terminal sessions, responses from the system are in upper case, and commands you type in are in lower case type.

TSOEXEC Command

Some installations distinguish between authorized and unauthorized commands and programs. The system programmer designates which commands and programs are authorized.

The TSOEXEC command allows a TSO user to invoke all unauthorized and some authorized commands and programs from an unauthorized environment. This is especially helpful when using TSO in an unauthorized environment, such as ISPF. When you invoke an authorized command with the TSOEXEC command, TSO processes the authorized command as if you invoked it from an authorized environment. You can invoke some authorized programs by using the TSOEXEC command with the CALL command. The TSOEXEC command can be invoked in both foreground and background TSO sessions.

The TSOEXEC command is based on the TSO service facility. For further information on how the TSO service facility functions, see Chapter 18, "Invoking Programs or Commands from a Program with the TSO Service Facility" in this book and *TSO/E Guide to Writing a Terminal Monitor Program.*

To invoke an authorized command from an unauthorized environment, issue the TSOEXEC command followed by the authorized command.

EXAMPLE

```
USER1 at node NODEA using TSO under ISPF wants to use the
TRANSMIT command to send a copy of data set SYS1.PARMLIB
to another user (USER2 at NODEB).

TSOEXEC TRANSMIT NODEB.USER2 DA('SYS1.PARMLIB')
```

For more information on the TSOEXEC command, see *TSO/E Command Language Reference.*

HELP Command

Use the HELP command to receive all the information available on the system on how to use any TSO command. The requested information is displayed at the user's terminal.

Using the HELP command with its operands, you can request the system to display:

- The syntax notation used to request the HELP information
- A list of all TSO commands in the system, along with an explanation of each
- All the available information in the system associated with a command
- The function and operation of a command
- The syntax for issuing a command
- More information about a command's operands
- More information associated with messages that result from executing a program coded in VSBASIC, or from issuing the TRANSMIT or RECEIVE commands.

The EDIT, TEST, and OUTPUT commands have a HELP subcommand. This subcommand acts similarly to and uses similar syntax as the TSO HELP command. When using the HELP subcommand, follow the instructions for the HELP command, but substitute the subcommand names.

For more information about the HELP command, see *TSO/E Command Language Reference*.

Syntax of HELP Information

You can get the syntax descriptions for commands by entering the HELP command followed by the HELP operand. For example:

```
HELP HELP
```

Listing All TSO Commands

To receive a list of all the TSO commands in the system along with a description of each, enter the HELP command with no operands.

EXAMPLE

```
HELP
```

An installation may also place its own information about installation-written commands on the system. The system treats this information in the same way as it treats the IBM-supplied information.

Requesting All Available Information about a Command

You may request specific information about commands by specifying operands with the HELP command. You can request additional information about the function, syntax, operands, subcommands or certain messages associated with all commands or a particular command.

To request all available information about a particular command, specify the command name.

EXAMPLE

```
To request all of the information available in the system
about the LISTCAT command, enter:

HELP LISTCAT
```

This results in the system displaying information about the syntax, function, and operands associated with the LISTCAT command.

Requesting Information about a Command's Function - the FUNCTION Operand

To request only the information pertaining to a command's function and operation, specify the command name followed by the FUNCTION operand.

EXAMPLE

```
To request the information about the function and operation
of the SEND command, enter:

HELP SEND FUNCTION
```

Requesting Information about a Command's Syntax - the SYNTAX Operand

To request only the information pertaining to a command's syntax, specify the command name followed by the SYNTAX operand.

EXAMPLE

```
To request the information about the syntax of the RENAME
command, enter:

HELP RENAME SYNTAX
```

Requesting More Information about a Command's Operands - the OPERANDS Operand

To request more information pertaining to all operands of a command, specify the command name followed by the OPERANDS operand.

EXAMPLE

```
To request more information about all the operands of the
LISTDS command, enter:

HELP LISTDS OPERANDS
```

To request more information about a specific operand of a command, specify that operand in parentheses after the OPERANDS operand.

EXAMPLE

```
To request more information about the STATUS operand of the
LISTDS command, enter:

HELP LISTDS OPERANDS(STATUS)
```

Requesting More Information about a Message - the MSGID Operand

To request more information than that provided in the text of a TRANSMIT, RECEIVE, or VSBASIC error message, use the MSGID operand. Issue the HELP command :

- Specify the command
- Specify the MSGID operand
- Immediately enclose in parentheses the message identifiers separated by commas.

EXAMPLE

```
To request information about the TRANSMIT message INMX090A,
enter:

HELP TRANSMIT MSGID(INMX090A)
```

The information you receive includes what caused each error and how to prevent reoccurrences. You cannot use the FUNCTION, SYNTAX, OPERANDS, or ALL operands with MSGID.

List of TSO Commands

Following is a list of all the TSO commands that are documented in this manual and the major function each command performs.

The commands in this publication are not documented individually. They are documented according to the task each command performs. For an individual description of each command and its operands, see *TSO/E Command Language Reference*.

Command	Function
ALLOCATE	Allocating data sets.
CALL	Loading and executing load modules.
CANCEL	Halting a submitted job.
COPY	Copying all or part of one data set into another.
DELETE	Deleting one or more data set entries or one or more members of a partitioned data set.
EDIT	Entering data into data sets, or directly modifying data that is already in a data set.
FREE	Releasing (unallocating) a previously allocated data set.
HELP	Obtaining information about the function, syntax, and operands of commands and subcommands and information about certain messages.
LINK	Invoking the linkage editor service program.
LIST	Listing the contents of a data set.
LISTALC	Listing of the data sets that were allocated during the current TSO session.
LISTBC	Listing of the contents of the SYS1.BRODCAST data set.
LISTCAT	Listing the entries in a catalog.
LISTDS	Listing certain attributes of specific data sets at your terminal.
LOADGO	Loading a compiled or assembled program into real storage and beginning execution.
LOGOFF	Ending a terminal session.
LOGON	Accessing the system.
PROFILE	Establishing, changing, or listing your user profile.
PROTECT	Preventing unauthorized access to your non-VSAM data set.
OUTPUT	Directing held output.
RECEIVE	Receiving a transmitted message or data set.
RENAME	Changing the name of a non-VSAM cataloged data set or a member of a partitioned data set, or creating an alias name for a member of a partitioned data set.
RUN	Compiling, loading, and executing source statements in a data set.
SEND	Sending messages to another terminal user or the system operator on your system.
STATUS	Checking the progress of a background job.
SUBMIT	Submitting a background job for processing.
TERMINAL	Defining the operating characteristics of your terminal.
TRANSMIT	Sending messages or data sets to users on your system or on another system.
TSOEXEC	Invoking authorized commands or programs from an unauthorized environment.

Chapter 4. Communicating With Other Users

To communicate with the operator or other users, use the SEND, TRANSMIT, and RECEIVE commands. You can send and receive messages, sequential data sets, partitioned data sets, or select members of a partitioned data set. Use the SEND command to send a message to another user or to the operator within your system. Use the TRANSMIT command to send a message or a data set to another user within your system or on another system. Use the RECEIVE command to retrieve transmitted messages or data sets and restore them to their original formats. You can keep a record of your TRANSMIT and RECEIVE activity in a LOG data set. You can avoid repetitive typing of commonly used names and addresses when you use TRANSMIT by using a NAMES data set. The SEND, TRANSMIT, and RECEIVE commands and the NAMES and LOG data sets are discussed in this chapter.

SEND Command

Using the SEND command, you can:

- Send a message to the master console operator
- Send a message to specific users
- Ensure that a user receives the message you send, even though he or she may not be logged on when you send the message the LISTBC or LOGON command
- Specify to which operator or operator console you send a message to.

In the following examples, you will see one to three operands coded with each example of the SEND command. You must enclose the text of each message in single quotation marks.

When you send a message, the text cannot exceed 115 characters, including blanks. If you want to display an apostrophe in your message, enter two to get one. For example, when you send **'I''M READY'** as the message text, the receiver sees **I'M READY**.

For more information about the SEND command, see *TSO/E Command Language Reference*.

Sending a Message to the Master Console Operator

When you send any message, the minimum you can specify is the SEND command followed by the message text enclosed in single quotation marks. If you specify no operands, the system sends the message to the master console operator.

EXAMPLE

```
To send the message "Please cancel my ID" to the master
console operator, enter:

SEND 'Please cancel my ID'
```

Sending a Message to Specific Users - the USER Operand

To send a message to a specific user, specify the SEND command:

- Enter the message text
- Specify the USER operand followed by the destination user ID enclosed in parentheses.

EXAMPLE

```
To send the following message to user ID D00ABC1, enter:

SEND 'Don''t forget the meeting at 2:00.' USER(D00ABC1)
```

To send a message to two or more users, enter the SEND command:

- Enter the message text
- Specify the USER operand followed by the destination user IDs separated by commas, and enclosed in parentheses.

EXAMPLE

```
To send the following message to user IDs D00ABC1 and D99XYZ1,
enter:

SEND 'All you need is on CRSPK1.' USER(D00ABC1,D99XYZ1)
```

You can send a message to a maximum of 20 different users.

To send a message to your own user ID, enter an asterisk instead of a user ID when specifying the USER operand.

EXAMPLE

```
To send the following message to your user ID, enter:

SEND 'MAP has completed.' USER(*)
```

Specifying When to Receive a Message - the LOGON, SAVE, and NOW Operands

To ensure that the user gets the message you send, regardless of whether he or she is logged on or is receiving messages, enter the SEND command and specify:

- The message text
- The USER operand
- The LOGON operand.

EXAMPLE

```
To send the following message to D00ABC1, and to ensure that
he gets it, regardless of whether or not he is able to
receive the message immediately, enter:

SEND 'Have you gotten FRANK''s new disk?' USER(D00ABC1) LOGON
```

If D00ABC1 is logged on and receiving messages, the system displays the message at his terminal immediately. If the user is not logged on or is not receiving messages, the system displays the message at his terminal the next time he logs on.

To send a message to a user, but not have it displayed at the user's terminal until the next time he issues the LOGON or LISTBC command, enter the SAVE operand following the USER operand.

EXAMPLE

```
To send the following message to user D00ABC1, and not
have it displayed at his terminal until he either logs
on or issues the LISTBC command, enter:

SEND 'JUMBO has gone away' USER(D00ABC1) SAVE
```

NOW is a default operand with the SEND command. If you do not specify the LOGON or SAVE operands with the SEND command, the system will immediately send the message to the user specified. If the user is not logged on the system notifies you and deletes the message.

Sending Messages to a Specific Operator or Operator Console - the OPERATOR and CN Operands

Specific operators are known to the system by a route-code used to identify them. To send a message to a specific operator, enter the SEND command and specify:

- The message text
- The OPERATOR operand with the operator's specific route-code enclosed in parentheses.

EXAMPLE

```
To send the following message to the operator identified by
route-code 7, enter:

SEND 'PLEASE MOUNT TAPES' OPERATOR(7)
```

If you omit the route-code, the default value is 2, which indicates that the system routes the message to the master console operator. If you omit both the USER and OPERATOR operands, the system also routes the message to the master console operator.

Specific operator consoles are known to the system by a number assigned to identify them. To send a message to a specific operator console, enter the SEND command and specify:

- The message text
- The CN operand with the operator console identifier enclosed in parentheses.

EXAMPLE

```
To send the following message to the operator console
identified by number 29, enter:

SEND 'DID YOU MOUNT THE TAPES?' CN(29)
```

The integer you specify with this operand must be between 0 and 64.

TRANSMIT Command

The TRANSMIT command allows you to send to one or more users:

- A message
- A data set
- Selected members of a partitioned data set
- A message with a data set.

If you specify the MESSAGE operand when you transmit data, the TRANSMIT command prompts you for a message that will accompany the data. The message is displayed when the receiving user issues the RECEIVE command. The receiver sees the message before actually getting the data set, in case you want to include special processing instructions. You may enter the messages in either full-screen mode or single line mode.

The TRANSMIT command does not support:

- Data sets with keys
- Data sets with user labels
- ISAM data sets
- VSAM data sets.

For further information on the TRANSMIT command see *TSO/E Command Language Reference*.

Transmitting a Message

To transmit a message to another user, enter TRANSMIT and specify:

- The destination node, followed immediately by a period
- The destination user ID.

EXAMPLE

```
To transmit a message to user USER1 at node NODEID, enter:

TRANSMIT NODEID.USER1
```

The TERMINAL operand, a default with the TRANSMIT command, causes the system to prompt you to enter the text in either full screen mode or line mode, depending on the default for the type of terminal you are using.

Transmitting a Data Set

To transmit a data set to another user, enter the TRANSMIT command and specify:

- The destination node, followed immediately by a period
- Immediately specify the destination user ID
- The DATASET operand with the data set name enclosed in parentheses.

EXAMPLE

```
To transmit data set SAMPLES.TEXT to user USER1 at node NODEID,
enter:

TRANSMIT NODEID.USER1 DATASET(SAMPLES.TEXT)
```

Transmitting Selected Members of a Partitioned Data Set - the MEMBER Operand

To transmit selected members of a partitioned data set to another user, enter the TRANSMIT command and specify:

- The destination node, followed immediately by a period
- Immediately specify the destination user ID
- The DATASET operand with the data set name enclosed in parentheses
- The MEMBER operand with the member names enclosed in parentheses.
  - Within the parentheses, separate the member names with commas.

EXAMPLE

```
To send members FIRST and SECOND of data set SAMPLES.TEXT to
user USER1 at node NODEID enter:

TRANSMIT NODEID.USER1 DATASET(SAMPLES.TEXT) MEMBER(FIRST,SECOND)
```

Transmitting a Data Set and a Message - the MESSAGE Operand

To transmit a data set with an accompanying message to another user, enter the TRANSMIT command and specify:

- The destination node, followed immediately by a period
- The destination user ID
- The DATASET operand with the data set name enclosed in parentheses
- The MESSAGE operand.

EXAMPLE

```
To send data set SAMPLES.TEXT with an accompanying message
to user USER11 at node NODEID, enter:

TRANSMIT NODEID.USER11 DATASET(SAMPLES.TEXT) MESSAGE
```

The system then prompts you to enter the message text in either full-screen mode or in line mode, depending on the default for the type of terminal you are using.

If you specify MESSAGE along with the default, TERMINAL, the system prompts you twice for the message text.

Transmitting to More than One Receiver

To transmit a message or a data set to more than one receiver, enter the TRANSMIT command and:

- Enclose in parentheses the following:
  - The destination node, followed immediately by a period, followed by the destination user ID
  - Separate each nodeid.userid by commas
- Specify the data set and/or message as described previously.

EXAMPLE

```
To send a message to users USER1 and USER2 at node NODEID
enter:

TRANSMIT (NODEID.USER1,NODEID.USER2)
```

Using Nicknames and the Names Data Set

You can reduce repetitive typing of nodes and user IDs by using nicknames and the NAMES data set. The nickname is a 1 to 8 character name that is a synonym for the node and user ID. The system finds the actual node and user ID by looking up the nickname in tables provided in the NAMES data set.

The NAMES data set allows you to perform the following general functions:

- Abbreviate addresses by associating a nickname with a node name and user ID or with a list of other nicknames (distribution list).
- Provide standard openings and closings for messages
- Control defaults for logging and notification of transmissions
- Use the nicknames section of other NAMES data sets.

The NAMES data set is composed of two parts:

- The control section. The control section must precede the nicknames section and is terminated by the first nick tag. The control section is used to set defaults for the LOG/NOLOG and NOTIFY/NONOTIFY tags, set prolog or epilog lines, set the default LOG data set name, and identify other NAMES data sets which the system can use.
- The nicknames section. The nicknames section is a directory of nicknames comprised of one nickname with its supporting data and tags for each entry that you wish to define.

The NAMES data set in Figure 4-1 contains commonly used tags. Your own NAMES data set may have these tags as well as any tags listed below.

The tags are:

Tag	Description
:ALTCTL.dsname	The :ALTCTL tag is used in the control section of the userid.NAMES.TEXT data set to provide the name of secondary NAMES data sets. Your first NAMES data set must be userid.NAMES.TEXT, but the ALTCTL tags can point to another user's NAMES data set or another data set that you own. The :ALTCTL tag may appear up to ten times.
:CC.name name...	The :CC tag is used in a nickname entry to specify a list of users that make up the distribution list. A distribution list can contain up to 100 names. Each name specified may be either a nickname or the name of another distribution list. The explicit user identification (userid and node name) can not be used with this tag. The :CC tag is identical to the :LIST tag.
:EPILOG.text	The :EPILOG tag is used in the control section to specify a text line to be appended at the end of any transmitted message. Typically, this will be a standard memo closing. The maximum length of an epilog line is 72 characters. Up to ten epilog lines may be specified.
:LIST.name name...	The :LIST tag is used in a nickname entry to specify a list of users that make up the distribution list. A distribution list can contain up to 100 names. Each name specified may be either a nickname or the name of another distribution list. The explicit user identification (userid and node name) can not be used with this tag. The :LIST tag is identical to the :CC tag. If you want to be notified when addressees on a distribution list receive your mail, you must specify :NOTIFY in the distribution list entry or specify NOTIFY(ALL) on the TRANSMIT command.
:LOGSEL.name	The :LOGSEL tag is used in the control section to specify the second (middle) qualifier(s) of all LOG data set names. The value specified with LOGSEL may be 1 to 40 characters (for example: *userid.name. name2.name3...* Remember that the middle qualifiers can be 1 to 40 characters, but each 8 characters must be separated by a period. If no middle qualifier is specified, LOG is used as the middle qualifier (userid.LOG.MISC)
:LOGNAME.name	The :LOGNAME tag may be used either in the control section or in a nickname entry. The value specified in the control section serves as a default third qualifier for the LOG data set name. For example *userid.LOG.logname* If :LOGNAME. is specified in a nickname entry, the value provided overrides the default set in the control section. If no LOGNAME is specified, the default MISC is used (userid.LOG.MISC).
:LOG/:NOLOG	The :NOLOG and :LOG tags may be used either in the control section or in the nickname section. When used in the control section, the LOG or NOLOG tag controls logging for any addressee specified by node and userid and also controls logging for any nickname which does not also specify :LOG or :NOLOG. If the nickname entry contains the :LOG or :NOLOG tag, this value overrides any value in the control section, but in turn may have been overridden by the LOG or NOLOG parameter on the TRANSMIT command. When receiving a message, if NOLOG is specified in the receiver's NAMES data set, either: ● in the control section ● on a NICK tag describing the sender the receiver is prompted with a message asking to receive dataset 'A.MAIL.userid'. When the message is stored it will be placed in 'myid.MAIL.userid'. "Myid" is the receiver's TSO user-ID and "userid" is the sender's TSO user-ID.

:LOGLST/:NOLOGLST — The :LOGLST or :NOLOGLST tag is used in a nickname entry that defines a distribution list. The tags indicate whether a log entry should be made for each user in the list (:LOGLST) or not (:NOLOGLST).

:NAME.username — The :NAME tag specifies the name of the user being defined. This name will appear in the copylist and in any log entries for this nickname. The specified name value may be up to 30 characters long.

:NICK.nickname — The :NICK tag is used to begin each nickname entry in the NAMES data set. It must be the first nonblank (except for line numbers) character of the record. The nickname can be any string of nonblank alphanumeric (A-Z and 0-9) characters and one to eight characters in length. The first :nick tag separates the control section of a NAMES data set from the nicknames section.

:NODE.nodeid — The :NODE tag is used within a nickname entry to specify the computer system that you want to TRANSMIT to. If the :NODE tag is not present in a nickname entry, the computer system that you are transmitting from is assumed. This tag can not be included in a distribution list (a nickname entry that includes either a :CC tag or :LIST tag).

:NOTIFY/:NONOTIFY — The :NOTIFY and :NONOTIFY tags can be used either in the control section or in a nickname entry. When used in the control section, the NOTIFY or NONOTIFY tag controls notification for any addressee specified by node and userid and for any nickname where the nickname entry does not contain :NOTIFY or :NONOTIFY. The value of a :NOTIFY or :NONOTIFY tag can be overridden by the NOTIFY or NONOTIFY parameter on the TRANSMIT command.

Receipt notification is the default for any addressee entered individually on the TRANSMIT command, but not for addressees derived from distribution lists. If you want to be notified for addressees on distribution lists, you must have specified :NOTIFY on the distribution list in the control data set or specified NOTIFY(ALL) on the TRANSMIT command.

:PROLOG.text — The :PROLOG tag is used in the control section to specify a text line to be inserted at the beginning of any transmitted message. The maximum length of a prolog line is 72 characters. Up to ten prolog lines may be specified.

:USERID.userid — The :USERID tag identifies the userid for a nicknamed user. The :USERID tag may not be used in the same entry as the :LIST or :CC tags.

Example of a NAMES data set

```
*Control Section
:altctl. OTHER.NAMES.DATA.SET
:prolog. MY NAME
:prolog. MY ADDRESS
:prolog. MY PHONE NUMBER
:epilog. SINCERELY YOURS,
:epilog. MY NAME

*Nicknames Section
:nick.DISTRIB :name. Distribution List
              :list. BILL
                     HOWARD
                     RUTH
                     SUE
                     TOM

:nick.DISTRO :name. Other Distribution List
             :cc. DISTRIB
                  SHERRY
                  MARK
                  JOHN

:nick.DISTRO2 :name. Other Distribution List
              :cc. BILL,DISTRO,ALEX

:nick.TOM :name. Tom Jones
          :node. PLPSC
          :user id. D01TXJ
          :logname.INM.LOG.DATA

:nick.SUE :name. Susan Smith
          :node. ABCXYZ
          :user id. A11SXS

:nick.BILL :name. Bill West
           :node. TDCSYS5
           :user id. H01WWW

:nick.RUTH :name. Ruth Ryan
           :node. ARDNIA
           :user id. T35RYR
```

Figure 4-1. Example of a NAMES Data Set

EXAMPLE

To transmit a message to four of the users in the NAMES data set, enter:

```
TRANSMIT (TOM,SUE,BILL,RUTH)
```

RECEIVE Command

Use the RECEIVE command to retrieve files that have been transmitted to you. When you issue the RECEIVE command, the system retrieves the transmitted data set or message that has been waiting longer than any other. The system then displays any descriptive information about the data set and prompts you for information the system uses to control the restore operation.

Using the RECEIVE command, you can:

- Retrieve transmitted data: messages, data sets, or both.
- Control how you save the data set under your user ID.

The RECEIVE command enables you to receive data sets or messages that are transmitted to you from the same system that you are using or another one. Procedures to initiate receiving will vary with different installations. At your installation:

- A RECEIVE command may be issued automatically when you enter LOGON

- A message may be sent to notify you when a message or data set is available to be received.

Contact your system support center to determine what option is in effect on your system. If neither of these two options is used on your system, you may have to enter a RECEIVE command periodically to determine if transmissions are available for you.

What happens when the RECEIVE command is entered depends on what (if any) transmissions are available. The possibilities are:

- If no transmissions have arrived, the system issues a message and terminates the process

- If multiple transmissions are available, the system processes them one after another without your having to enter additional RECEIVE commands

- If a transmission is a message, the system displays it immediately at your terminal.

- If a transmission is a data set, you have the following options:

  - Receive the data set with either the default name or a name that you supply

  - Delete the data set

  - Postpone receiving the data set.

The RECEIVE command cannot, in general, reformat data sets. Be sure that you tell the system to write the transmitted data into a data set that has the same record format as the original data set. Be sure that the record length is compatible (equal for fixed length records and equal or longer for variable length

records.) Also, be sure to specify a block size that is compatible with both the record length and record format of the transmitted data. If the system detects a mismatch in block size, record length, or record format, the system terminates the RECEIVE command and issues the appropriate error messages.

The system warns you if you receive a data set that was RACF or PASSWORD protected. The system takes no further action to protect the received data.

Receiving a Transmitted Data Set

To retrieve and restore a transmitted data set, use RECEIVE with no operands.

EXAMPLE

```
If data set EXAMPLES.TEXT is transmitted to your user ID,
to retrieve and restore the data set, enter:

RECEIVE
```

To retrieve a data set accompanied by a message, or a message itself, enter RECEIVE with no operands, as in the preceding example.

When receiving a data set, the system prompts you for information to control how the data set is restored. You can accept the system defaults offered with the prompts, or supply additional operands with the RECEIVE command, as described in the *TSO/E Command Language Reference*.

The LOG Data Set

The LOG data set is a journal of all TRANSMIT and RECEIVE activity. It keeps a record of:

- What you transmitted
- To whom you transmitted
- When you transmitted
- The success or failure of your attempt to transmit
- What you received
- From whom you received
- When you received
- The success or failure of your attempt to receive

Example of a LOG data set

```
TRANSMIT         D01TXJ.INM.LOG.DEFAULT   16 SEP 1981 10:26
  TO:   JONES     PLPSC     D01TXJ  Tom Jones
This is the text of a message that was
transmitted
```

```
RECEIVE          D01TXJ.INM.LOG.DEFAULT   16 SEP 1981 10:34
  FROM: JONES     PLPSC     D01TXJ  Tom Jones   16 SEP 1981 10:26
This is the text of a message that was
received along with a data set (SEQ.DATA)
   DSN: D32JAD3.SEQ.DATA
```

```
RECEIVE          ** ACKNOWLEDGMENT **     16 SEP 1981 10:34
  FROM: JONES     PLPSC     D01TXJ  Tom Jones   16 SEP 1981 10:34
   STORED        ** MESSAGE **   16 SEP 1981 10:26
```

Figure 4-2. Example of a LOG Data Set

The three entries in the above LOG data set show:

- A transmitted message
- A message received along with a data set
- Notification that a message was received.

You may have several LOG data sets or just one. The default data set name is userid.LOG.MISC but can be specified by use of the :LOGSEL and :LOGNAME tags in your NAMES data set. For further information concerning these tags see "The NAMES Data Set" in *TSO/E Command Language Reference.*

Section II: Using Data Sets

TSO allows you to use data sets in the following ways:

- Create data sets and specify access to them
- Rename data sets
- Delete data sets
- Copy the data from one data set to another
- Request information about data sets
- Protect data sets from unauthorized use
- Release allocated data sets
- Delete data sets from the system.

This section shows you how to perform the preceding tasks using TSO commands.

Chapter 5. Allocating Data Sets

Allocating a data set is the way you request MVS to allow you to use a data set. If the data set does not exist, the system creates it and then gives you access to it. You can allocate data sets at your terminal, or you can dynamically allocate data sets during program execution.

ALLOCATE Command

Using the ALLOCATE command you can:

- Create data sets
- Model the attributes of a new data set after an existing data set
- Request two types of access to an existing data set. They are:
  - Exclusive access, such that no other user can access the data set until you are finished using it
  - Non-exclusive access, such that other users can access the data set while you are using it.
- Specify how much space a data set can use on a device
- Specify on what tape or disk the data set resides
- Specify the disposition of the data set when you are finished using it
- Designate a data set as a system output data set
- Concatenate library data sets - allocate a number of data sets to be associated with a single ddname.

For more information on ALLOCATE and its operands, see the *TSO/E Command Language Reference.*

The Operands Used with ALLOCATE

Because the allocation process describes the attributes of a data set, the ALLOCATE command has many operands associated with it. To help you decide which of the operands you will need to use in a particular situation, and what you can do with the operands, look through the following list of the more commonly used operands. Also, Figure 5-1 can help you choose which operands you need for a specific task.

TASK	DATASET	DISP		VOLUME	SPACE
		OLD,SHR or MOD	NEW		
Create A Data Set	X		X	Optional	X
Retrieve a Cataloged Data Set	X	X			
Retrieve an Uncataloged Data Set	X	X		X	

Figure 5-1. Operands Needed to Allocate a Data Set

Naming the data set - DATASET, FILE

One of these operands must be specified. The DATASET operand names the particular data set you allocate. The FILE operand names the data definition (DD) statement that defines the data set you allocate.

Modeling one data set after another - LIKE

This operand specifies the name of an existing data set whose attributes are to be used as the attributes of the new data set being allocated.

Requesting type of access to data sets - NEW, MOD, OLD, SHR

The NEW operand tells the system that the data set does not exist, that you want to create it, and that you will have exclusive access to it.

The MOD operand tells the system that you want to add data to the end of a data set, and gives you exclusive access to the data set. The data set must be sequential.

The OLD operand gives you exclusive access to a data set.

The SHR operand gives you non-exclusive access to a data set that already exists.

Requesting an output data set - SYSOUT

The SYSOUT operand indicates that the data set is to be a system output data set. This operand is mutually exclusive with the NEW, MOD, OLD, and SHR operands.

With the SYSOUT operand, you can also specify one of the SYSOUT classes used at your installation. Consult your system support center to find out what classes you can use.

Requesting space for a data set - SPACE

When you allocate a new data set you can make specific requests for the amount of storage the data set can use. You need to request space attributes for a new data set only if you do not want to use the system defaults. For more information about the system default space allocations for a data set, see *TSO/E Command Language Reference*. You need not request any space attributes for an existing data set.

Accessing a data set on a specific volume - VOLUME

This operand specifies the serial number of a direct access volume on which a new data set is to reside or on which an existing data set is located. If you specify VOLUME for an existing data set, the data set must reside on the specified volume, or you get an error. If you do not specify VOLUME for a new data set, the system places the data set on any eligible volume.

Disposing of a freed data set - KEEP, DELETE, CATALOG, UNCATALOG

The disposition of a data set specifies what happens to the data set when you are finished using it.

- KEEP specifies that the system retain the data set.
- DELETE specifies that the system delete the data set.
- CATALOG specifies that the system retain descriptive information about the data set in a catalog. Cataloging allows you to keep track of and retrieve data sets. You can catalog data sets in the system master catalog or in user (private) catalogs. When retrieving a cataloged data set, you do not have to specify volume information.
- UNCATALOG specifies that the system remove descriptive information from the catalog.

The disposition of a data set only takes effect when the data set is freed. You free the data sets when you issue the FREE command or when you terminate your TSO session with either the LOGOFF or LOGON commands. When you terminate your session, the system automatically frees all data sets allocated to your user ID.

You can use any one of the four preceding operands when specifying a data set's disposition, regardless of the data set's disposition before your program began executing.

See Figure 5-2 for help in choosing a disposition in a specific situation.

TASK	Initial Status		Disposition at Normal Termination of Program			
	OLD,SHR	NEW	KEEP	DELETE	CATLG	UNCATLG
To keep a new data set		X	X			
To delete a new data set		X		X		
To catalog a new data set		X			X	
To keep an existing data set	X		X			
To delete an existing data set	X			X		
To catalog an existing data set	X				X	
To uncatalog a cataloged data set	X					X

Figure 5-2. Choosing a Disposition for a Data Set

Creating a Data Set

To create a data set issue the ALLOCATE command and specify:

- The DATASET operand.
- The name of the data set enclosed in parentheses.

  When you specify a data set name on the ALLOCATE command, the data set is kept and cataloged when you free it, unless you specify otherwise. When you do not specify a data set name, the data set is deleted when you free it or log off.

- Either the NEW or MOD operand.

  The MOD operand indicates that if the data set already exists you want to change it. If the data set does not exist, you want to create it.

- If you are creating a new partitioned data set, the DIR operand.

  DIR specifies the number of directory blocks that the system allocates for a new partitioned data set.

EXAMPLE

```
To create the partitioned data set, TEST.DATA, with five
directory blocks, and space of ten cylinders primary
allocation and five cylinders secondary space allocation,
enter:

ALLOCATE DATASET(TEST.DATA) NEW DIR(5) SPACE(10,5) CYL
```

Although you can specify record format, data set type, and other information for a data set, the system uses default values when you omit this information. For more information on the defaults, see *TSO/E Command Language Reference.*

Creating a Data Set Modeled After Another Data Set - the LIKE Operand

To create a data set that has the same data set attributes as an existing data set, issue the ALLOCATE command and specify:

- The DATASET operand followed by the name of the data set to be created enclosed in parentheses
- The LIKE operand followed by the name of the model enclosed in parentheses.

EXAMPLE

```
To create data set TEST.DATA, and to ensure that it has the
same attributes as MODEL.DATA, specify:

ALLOCATE DATASET(TEST.DATA) LIKE(MODEL.DATA)
```

Requesting Access to an Existing Data Set - the OLD, SHR, MOD Operands

When using these operands, you must specify a data set name. If you are accessing an existing data set that is not cataloged, specify the VOLUME operand. The system retains existing data sets when you free them unless you specify the DELETE operand.

Requesting Exclusive Access:

To request exclusive access to a data set that already exists, use the OLD operand. While you are using the data set, no one else can use it or change it. Issue the ALLOCATE command and specify:

- The DATASET operand with the name of the data set enclosed in parentheses. You may also specify an associated file name.
- The OLD operand.

EXAMPLE

```
Assume data set TEST.DATA associated with the file name
SYSPRINT already exists, and you want to write to it.
You want to prevent any other user from also writing to it.
Request exclusive access to the data set by entering:

ALLOCATE DATASET(TEST.DATA) FILE(SYSPRINT) OLD
```

Requesting Non-exclusive Access:

The SHR operand indicates that you are willing to share the data set with any other user who also requests non-exclusive access. If you are only going to read from a data set, you can request non-exclusive access. To request non-exclusive access to a data set that already exists, issue the ALLOCATE command and specify:

- The DATASET operand with the name of the data set enclosed in parentheses. You may also specify an associated file name.
- The SHR operand.

EXAMPLE

```
To request non-exclusive access to data set TEST.DATA,
associated with the file name SYSIN, enter:

ALLOCATE DATASET(TEST.DATA) FILE(SYSIN) SHR
```

Requesting Access to Add to a Sequential Data Set

If your program adds records to the end of a data set, use the MOD operand when requesting access to the data set. The MOD operand requests exclusive access to the data set. Issue the ALLOCATE command and specify:

- The DATASET operand with the name of the data set enclosed in parentheses. You may also specify an associated file name.
- The MOD operand.

EXAMPLE

```
If your program adds records to the end of data set
TEST.DATA, associated with file name SYSPRINT, enter:

ALLOCATE DATASET(TEST.DATA) FILE(SYSPRINT) MOD
```

Making Specific Space Requests for a Data Set - The SPACE Operand

When you allocate a data set, you may not always be sure how much or what type of space to specify. You can copy space information from another data set using LIKE or, if you are enlarging an existing data set, look at the previous space allocation and increase it. When you request a specific amount of space, you can specify two quantities: a primary quantity and a secondary quantity. The primary quantity is how much space the system originally allots to the data set. The secondary quantity is how much more space the system allots each time the data set's current allotment gets filled. The secondary quantity will be allocated to the data set up to fifteen times. You can also define the units of space the system allots. You can request units of space in blocks, tracks, or cylinders.

Some points to keep in mind when choosing values for the SPACE operand are:

- Tracks and cylinders represent different storage values depending on the device the data set resides on. Either consult a system programmer or the manual that describes the particular device your data set will reside on for the number of bytes in a track or cylinder.
- It is often better to request more secondary space than you might need as opposed to running out of allotted space for the data set.
- If you have not allocated enough space for a data set, to enlarge it, you can:

  1. Create another data set with the same attributes as the full data set, except that you should request more space.
  2. Copy the full data set into the new data set.
  3. Delete the first data set, and, if you want, rename the new data set using the name of the original data set.

To specify the space allocation for a data set, issue the ALLOCATE command and specify:

- The DATASET operand with the data set name enclosed in parentheses
- The NEW operand
- The SPACE operand
  - Enclose in parentheses the primary and secondary space quantities, separated by a comma
  - Specify the type of space you want. The choices are BLOCK, AVBLOCK, TRACKS, or CYLINDER.

EXAMPLE

```
To create data set TEST.DATA, and allocate 10 cylinders
of primary space and 5 cylinders of secondary space, enter:

ALLOCATE DATASET(TEST.DATA) NEW SPACE(10,5) CYL
```

Locating a Data Set on a Specific Volume - the VOLUME Operand

To create or access data sets that are uncataloged, use the Volume operand. Use the NEW operand with VOLUME to create an uncataloged data set on a specified volume. Use the OLD operand with VOLUME to request exclusive access to an existing uncataloged data set. Issue the ALLOCATE command and specify:

- The DATASET operand with the name of the data set enclosed in parentheses
- The VOLUME operand with the volume's serial number enclosed in parentheses.

EXAMPLE

```
To create a data set TEST.DATA, modeled after the data set
MODEL.DATA and to reside on volume TSO349, enter:

ALLOCATE DATASET(TEST.DATA) NEW VOLUME(TSO349) LIKE(MODEL.DATA)
```

You can also make a specific volume request for a data set that resides on more than one volume (a multivolume data set). This is very similar to the preceding example. The only difference is that you specify the volume serial numbers of each volume the data set resides on.

EXAMPLE

```
To gain exclusive access to data set TEST.DATA, which resides
on volumes TSO349, TSO350, and TSO359, enter:

ALLOCATE DATASET(TEST.DATA) OLD VOLUME(TSO349,TSO350,TSO359)
```

Specifying a Data Set's Disposition When Freed - the KEEP, CATALOG, UNCATALOG, and DELETE Operands

To ensure that the system retains the data set after the data set has been freed, use the KEEP operand. Issue the ALLOCATE command and specify:

- The DATASET operand with the name of the data set enclosed in parentheses. You may also include an associated file name.
- The KEEP operand.

EXAMPLE

```
To retain the SYSOUT data set associated with the file
SYSPRINT after it has been freed, enter:

ALLOCATE SYSOUT FILE(SYSPRINT) NEW KEEP
```

To ensure that the system retain the data set's descriptive information in a catalog after the data set has been freed, use the CATALOG operand. Issue the ALLOCATE command and specify:

- The DATASET operand with the name of the data set enclosed in parentheses
- The CATALOG operand

EXAMPLE

```
To retain the new data set TEST.DATA's descriptive information
in a catalog after it has been freed, enter:

ALLOCATE DATASET(TEST.DATA) NEW CATALOG
```

To ensure that the system remove the data set's descriptive information from a catalog after the data set has been freed, use the UNCATALOG operand. Issue the ALLOCATE command and specify:

- The DATASET operand with the name of the data set enclosed in parentheses
- The UNCATALOG operand.

EXAMPLE

```
To remove data set TEST.DATA's descriptive information from a
catalog after the data set has been freed, enter:

ALLOCATE DATASET(TEST.DATA) OLD UNCATALOG
```

To ensure that the system deletes the data set from the system and uncatalogs the data set after the data set has been freed, use the DELETE operand. Issue the ALLOCATE command and specify:

- The DATASET operand with the name of the data set enclosed in parentheses
- The DELETE operand.

EXAMPLE

```
To delete data set TEST.DATA from the system after it has
been freed, enter:

ALLOCATE DATASET(TEST.DATA) OLD DELETE
```

Designating a System Output Data Set - The SYSOUT Operand

The system directs output data to the job entry subsystem (JES2 or JES3). The job entry subsystem routes the data set to a final output device, based on the class you specify with the SYSOUT operand. After the job entry subsystem routes the data set, the system deletes it.

To allocate a data set as a system output data set, issue the ALLOCATE command and specify:

- The FILE operand with the name of the data definition (DD) statement that defines the output data set enclosed in parentheses.
- The SYSOUT operand.

  You can also specify the output class the system uses when processing the output data set. The output class is a single letter following the SYSOUT operand and enclosed in parentheses.

EXAMPLE

```
To allocate the data set defined by DD statement SYSPRT
as a system output data set, and to request the system
to process as output class A, enter:

ALLOCATE FILE(SYSPRT) SYSOUT(A)
```

Concatenating Data Sets into a Library

You can concatenate related data sets in a library under a single ddname, to establish the order in which the system searches the data sets and to simplify data set invocation. For example, if you concatenate CLIST data sets in the file SYSPROC, you can invoke individual CLISTs by typing *%CLISTname* on the command line.

To concatenate a number of data sets, specify a list of the data set names with the DATASET operand.

EXAMPLE

```
To concatenate existing data sets A.CLIST, B.CLIST, and
C.CLIST to the data definition statement named SYSPROC,
specify:

ALLOCATE FILE(SYSPROC) DATASET(A.CLIST,B.CLIST,C.CLIST) +
         SHR REUSE
```

There are two ways to add another data set to a data set concatenation:

1. Use the FREE command to unallocate or free the data sets in the concatenation. Then reallocate the entire concatenation, including the data set to be added, using the ALLOCATE command.

2. Specify the REUSE operand with the ALLOCATE command when you concatenate. The REUSE operand specifies that the filename being allocated is to be freed and reallocated if it is currently in use. Then reallocate the entire concatenation, including the data set to be added, using the ALLOCATE command.

When data sets are concatenated, the system treats the group as a single data set. It is important to consider the characteristics of the individual data sets that you are working with. Concatenated partitioned data sets use the attributes of the first data specified only. If you are concatenating data sets with unlike attributes, you should specify the data set with the largest block size first to ensure that you will not lose any data during the concatenation. If you need to concatenate the data sets in a certain order regardless of block size, see *TSO/E Command Language Reference* for information on specifying sufficient buffer space.

For more information about data set concatenation, see *MVS JCL*. For more information about the FREE command, see "FREE Command."

Chapter 6. Renaming Data Sets

RENAME Command

Use the RENAME command to:

- Change the name of one or more non-VSAM cataloged data set
- Change the name of a member of a cataloged partitioned data set
- Create an alias for a member of a cataloged partitioned data set.

You can use the Access Method Services ALTER command to change the name of a VSAM data set. For more information on the ALTER command, see *Access Method Services.*

Do not use the RENAME command to create an alias for a load module created by the linkage editor.

When you rename a password protected data set, the data set does not retain the password. If you want to password protect the renamed data set, you must use the PROTECT command to assign a password to the data set before you access it.

In the following examples, assume that the user issuing the RENAME command owns the data set.

For more information on the RENAME command, see *TSO/E Command Language Reference.*

Changing a Data Set's Name

To change the name of a data set, specify on the RENAME command:

- The data set name you want to change
- The data set's new name.

EXAMPLE:

```
To change the name of data set TEST.DATA to PARTS.DATA,
enter:

RENAME TEST.DATA PARTS.DATA
```

Renaming a Group of Data Sets

To rename a group of data sets that have at least one qualifier in common, issue the RENAME command and specify asterisks in place of the qualifiers that vary in both the new and old data set names. Remember that you are renaming all data sets with the specified common qualifier.

EXAMPLE

```
To change the descriptive qualifier of the following data sets
from DATA to TEXT:

TEST.DATA
PARTS.DATA
OLD.DATA
WORK.DATA

to

TEST.TEXT
PARTS.TEXT
OLD.TEXT
WORK.TEXT

enter the command as

RENAME *.DATA *.TEXT
```

Renaming a Member

To change the name of a member of a partitioned data set, issue the RENAME command and:

- Specify the data set name and specify the old member name enclosed in parentheses
- Optionally, specify the data set name again
- Enclose the new member name in parentheses.

EXAMPLE

```
To rename MEMBER1 of data set TEST.DATA to MEMBER2, enter:

RENAME TEST.DATA(MEMBER1) TEST.DATA(MEMBER2)
                  or
RENAME TEST.DATA(MEMBER1) (MEMBER2)
```

Creating an Alias Name for a Member - the ALIAS Operand

To create an alias name for a member of a partitioned data set, issue the RENAME command and:

- Specify the data set name and enclose the member name parentheses
- Optionally, specify the data set name again
- Enclose the member's alias in parentheses
- Specify the ALIAS operand.

EXAMPLE

To create AUXMEM as an alias for MEMBER1 of data set TEST.DATA, enter:

```
RENAME TEST.DATA(MEMBER1) TEST.DATA(AUXMEM) ALIAS
                  or
RENAME TEST.DATA(MEMBER1) (AUXMEM) ALIAS
```

Chapter 7. Copying Data Sets

The following are some of the facilities you can use to copy information from one data set to another:

- The UTILITIES option of the Interactive System Productivity Facility (ISPF).
- The TSO Data Utilities COPY command, which is described in *TSO Data Utilities: COPY, FORMAT, LIST, MERGE User's Guide and Reference.* Your installation must have the TSO Data Utilities package for you to use the COPY command.
- The TSO/E command SMCOPY. Although SMCOPY is primarily intended for use under the session manager, you can use it to copy some data set types regardless of whether or not you're logged on under the session manager. (See "Data Sets You can Copy Using SMCOPY" below for information about the types of data sets you can copy.) This chapter provides some information on using SMCOPY. For a description of the syntax of the SMCOPY command, see *TSO Extensions Session Manager Program Reference.*

Data Sets You can Copy Using SMCOPY

You can use the SMCOPY command to copy:

- A member of a partitioned data set into another member or into a sequential data set.
- A sequential data set into another sequential data set or into a member of a partitioned data set.

The data sets or members you copy must have:

- A variable (V) or fixed (F) record format. (The records can be either blocked or unblocked.)
- A logical record length of no more than 256 bytes.

To copy other types of data sets—such as load libraries, which have a record format of U—you can use ISPF (Interactive System Productivity Facility); for more information, see *Interactive System Productivity Facility/Program Development Facility for MVS: Reference.*

Some Examples of Using SMCOPY

In general, when using the SMCOPY command, specify the following operands:

- FROMDATASET(dsname) or FDS(dsname) to designate the originating sequential data set or member of a partitioned data set
- TODATASET(dsname) or TDS(dsname) to designate the destination sequential data set or member of a partitioned data set
- NOTRANS, if you do not want characters in the data set to be translated.

You can copy a data set to either a new or existing data set. When you copy to a new data set, the system allocates one for you, using the attributes of the data set being copied. When you copy to an existing data set, the system uses the attributes of the data set to which you're copying the data; you can copy data sets that have different logical record lengths or block sizes. You cannot, however, copy data sets that have different record formats. For example, you can't copy a data set with a record format of FB to a data set with a record format of VB.

Copying a Member of a Partitioned Data Set

For example, to copy member MEM1 of the partitioned data set PDS1.DATA into member MEM2 of PDS2.DATA, specify:

```
SMCOPY FDS(PDS1.DATA(MEM1)) TDS(PDS2.DATA(MEM2)) NOTRANS
```

To copy member MEM1 of the partitioned data set PDS1.DATA into the sequential data set SEQ1.DATA, specify:

```
SMCOPY FDS(PDS1.DATA(MEM1)) TDS(SEQ1.DATA) NOTRANS
```

Copying a Part of a Data Set

You can copy specific records from one sequential data set or member to another. Using the SMCOPY command, specify:

- The FDS and TDS operands, as shown above.
- The LINE operand with the range of records to be copied enclosed in parentheses. Specify a starting record and an ending record.

For example, to copy the first five records of text from the member MEM1 of PDS1.DATA to MEM2 of PDS2.DATA, specify:

```
SMCOPY FDS(PDS1.DATA(MEM1)) LINE(1:5) TDS(PDS2.DATA(MEM2)) NOTRANS
```

You cannot use the LINE operand to copy specific line numbers within a numbered data set; the system treats the numbers you specify as offsets from the beginning of the data set.

Chapter 8. Listing Data Set Information

You can request that the system return various types of information about a data set or a list of data sets by using the following commands:

LIST
Displays all or part of the contents of one or more data sets at your terminal. Your installation must have the TSO Data Utilities package for you to use the LIST command. The LIST command is not described in this chapter; for more information, see *TSO Data Utilities: COPY, FORMAT, LIST, MERGE User's Guide and Reference.*

LISTALC
Displays a list of the data sets allocated to your user ID during the current TSO session.

LISTCAT
Displays a list of entries from a catalog.

LISTDS
Displays the attributes of specific data sets at your terminal.

When you display information about a data set you own, you don't need to fully qualify the data set name; the system appends your user ID (or a prefix specified with the PROFILE command) to the beginning of the data set for you. For example, to list the attributes of a data set you own, PARTS.TEXT, enter:

```
LISTDS PARTS.TEXT
```

To list the attributes of a data set you do not own, be sure to specify the data set's fully-qualified name and enclose the name in apostrophes. For example:

```
LISTDS 'D00ABC1.PARTS.TEXT'
```

For all of the following examples of the LISTALC, LISTCAT, and LISTDS commands, assume that you want to display information pertaining to data sets you own.

LISTALC Command

Use the LISTALC command to list and optionally display information about the data sets and attribute-lists allocated to your user ID during the current TSO session. The list includes the data sets and attribute-lists you dynamically allocated and any data sets previously allocated for temporary use by a command.

Using the LISTALC command you can display:

- The data sets allocated to your user ID during the current TSO session.
- The attribute list names created by your use of the ATTRIB command.
- The data definition name (DDNAME) associated with a data set.
- The disposition of a data set.
- The history of a data set. The history of a data set includes:
  - The date the data set was created.
  - The date the data set will expire (retention period).
  - Whether or not the data set is password-protected.
  - The organization of the data set (sequential or partitioned).
- The names of the members of a partitioned data set.
- The fully-qualified name of a system-generated data set name.

When you see the list, data sets allocated to your terminal are followed by the word **TERMINAL** and attribute-lists are followed by the word **NULLFILE**. If an asterisk precedes the data set name in the list, the data set is allocated but not currently in use.

For more information on the LISTALC command, see *TSO/E Command Language Reference*.

Listing Attribute lists and Data Sets

There are times when you might want to find out exactly what data sets the system has currently allocated to your user ID. To list all attribute list names and all data sets allocated to your user ID during the current TSO session, enter:

```
LISTALC
```

Listing Attribute Lists, DDNAMEs, and Data Set Disposition - the STATUS Operand

When writing programs and JCL statements, it is often useful to know if any other steps or programs use the same data sets you do, and when and how they intend to do so. You can find this type of information by examining the attribute-list name, data definition names (DDNAME), and the data set's disposition.

A data set's disposition is indicated by one of the following operands discussed in detail under the ALLOCATE command:

CATLG
Requests that the system retain the data set and enter the data set's name into the system catalog.

UNCATLG
Requests that the system remove the data set's name from the system catalog, but still retain the data set.

KEEP
Requests that the system retain the data set.

DELETE
Requests that the system remove the data set's name from the system catalog and release the storage space the data set formerly occupied.

EXAMPLE

```
To list all attribute list names, dispositions, and all
DDNAMEs associated with each data set currently allocated
to your user ID, enter:

LISTALC STATUS
```

Listing the History of a Data Set - the HISTORY Operand

Before you access a data set, you might want to know more about it. For example, you might want to know when the data set was created, or when it will expire (be removed from the system). To avoid errors, you would also want to know if the data set is password-protected and how the data set is organized. The system
displays the data set's organization under the title, DSORG. The possible values for DSORG and their meanings are:

DSORG	Meaning
PS	A sequential data set
PO	A partitioned data set
IS	An indexed sequential data set
DA	A direct access data set
VSAM	A virtual sequential access method (VSAM) data set
**	An unspecified type of data set
??	Any other type of data set

EXAMPLE

```
To find out each data set's creation and expiration dates,
whether or not the data set is password-protected, and its
organization, enter:

LISTALC HISTORY
```

Listing the Members of a Partitioned Data Set - the MEMBERS Operand

A partitioned data set can have many members, often too many to remember by name. When using TSO, there are times when you need to use a specific member name within a partitioned data set. TSO provides a way for you to list the names of the members of each partitioned data set currently allocated to your user ID. Enter:

```
LISTALC MEMBERS
```

The list also includes each member's alias name(s), if any exist.

Listing the System-generated Data Set Names - the SYSNAMES Operand

To list the fully-qualified system-generated data set names for each data set currently allocated to your user ID, enter:

```
LISTALC SYSNAMES
```

LISTCAT Command

Use the LISTCAT command to obtain a list of entries from a catalog. You can select entries by name, by type, or by specific fields within an entry.

Using the LISTCAT command you can request that the system:

- List the entries of a particular catalog.
- Write the output to a data set rather than to the terminal screen.
- List the information associated with only specific entries.
- List the entries in a particular level.
- List alias entries only.
- List the entries that were created a specific number of days before the present.
- List the entries that expire within a certain number of days.
- Return all or a part of the following information stored in a catalog entry:
  - Data set name
  - Owner identification
  - Creation date
  - Expiration date
  - Entry type
  - Volume serial number
  - Device types.

For information on how to list catalog information for VSAM data sets, refer to *Access Method Services.*

When you enter LISTCAT with no operands, your user ID, or the last prefix specified with the PROFILE command, becomes the highest level entry name qualifier. The system only displays information about those entries associated with that user ID or prefix.

For more information on the LISTCAT command, see *TSO/E Command Language Reference.*

Listing the Entries From a Specific Catalog - the CATALOG, OUTFILE Operands

You can list all of the entries from a specific catalog at your terminal, or write them to a data set. To list all of the entries of a specific catalog at your terminal, enter the LISTCAT command:

- Specify the CATALOG operand
- Immediately enclose in parentheses the catalog name.

EXAMPLE

```
To list all of entries in catalog USERCAT1 at your terminal,
enter:

LISTCAT CATALOG(USERCAT1)
```

To write all of the entries of a specific catalog to a data set, enter the LISTCAT command and specify:

- The CATALOG operand, enclosing the catalog name in parentheses.
- The OUTFILE (abbreviated OFILE) operand.

  - The OUTFILE operand specifies the ddname of the DD statement, or the name specified with the FILE operand of the ALLOCATE command, that defines the data set to which the system writes the list of catalog entries. To find out which DD statement defines the data set you want to write to, use the LISTALC command. For more information on how to do this, see "Listing Attribute Lists, DDNAMEs, and Data Set Disposition - the STATUS Operand," earlier in this chapter.

  - When using the OUTFILE operand, enter the ddname or file name immediately after the operand, and enclose it in parentheses.

  - Be sure to separate the operands by either a comma or a space.

EXAMPLE

```
To list all of entries in catalog USERCAT1 in data set
CATLIST1.DATA, where CATLIST.DATA is defined by DD
statement CATDD, enter:

LISTCAT CATALOG(USERCAT1),OUTFILE(CATDD)
```

If USERCAT2 is a password-protected catalog, and the password is VGTBLS, enter a slash (/) and the catalog's password immediately following the catalog name field.

EXAMPLE

```
To list all of the entries in password-protected catalog
USERCAT2 at your terminal, enter:

LISTCAT CATALOG(USERCAT2/VGTBLS)
```

Listing the Entries for Specific Data Sets - the ENTRIES Operand

You can list the catalog information about specific data sets using the ENTRIES operand. To list specific data sets' information, enter the LISTCAT command and:

- Specify the ENTRIES operand

- Immediately enclose in parentheses the list of data sets separated by a comma or blank.

EXAMPLE

```
To list the catalog information about data sets named
TEST.DATA and TEST.TEXT, enter:

LISTCAT ENTRIES(TEST.DATA,TEST.TEXT)
```

If the data sets whose catalog information you want to obtain are password-protected, enter the password following the data set name and separate them using a slash(/).

EXAMPLE

```
TEST.DATA and TEST.TEXT have passwords of TESTD and TESTF,
respectively.  To list the catalog information associated
with the entries for the data sets named TEST.DATA and
TEST.TEXT enter:

LISTCAT ENTRIES(TEST.DATA/TESTD,TEST.TEXT/TESTF)
```

If you enter neither ENTRIES nor LEVEL (discussed later in this section) the system displays only the entries associated with the user ID.

Listing the Entry Names in a Catalog by Level - the LEVEL Operand

To list the entry names in a catalog associated with a particular entry name level, enter the LISTCAT command:

- Specify the LEVEL operand
- Immediately enclose in parentheses the level of the entry names.

EXAMPLE

```
To list all of the entry names in a catalog associated with
user ID D00ABC1, enter:

LISTCAT LEVEL(D00ABC1)

To list all of the entry names in a catalog associated with
user ID D00ABC1 and qualifier PROJ2, enter:

LISTCAT LEVEL(D00ABC1.PROJ2)
```

If neither LEVEL nor ENTRIES is entered, only the entries associated with the user's prefix are listed.

Listing Alias Entries in a Catalog - the ALIAS Operand

Catalogs can contain aliases, or as they are sometimes known, alternate entries. An alias entry is a copy of an entry that already exists, but with a different name. You can list all the alias entries in a catalog by specifying the ALIAS operand. You need not specify any other information with this operand.

EXAMPLE

```
To list all the alias entries in a catalog, enter:

LISTCAT ALIAS
```

Listing Entries in a Catalog By Creation and Expiration Dates - the CREATION, EXPIRATION Operands

You can list all the entries in a catalog for data sets that were created on or before a specified number of days from the present by using the CREATION operand followed immediately by a decimal number enclosed in parentheses.

EXAMPLE

```
To list the entries that were created in the last ten days,
enter:

LISTCAT CREATION(10)
```

You can list all the entries in a catalog for data sets that expire on or before a specified number of days from the present by using the EXPIRATION operand followed immediately by a decimal number enclosed in parentheses.

EXAMPLE

```
To list the entries that expired in the last eight days,
enter:

LISTCAT EXPIRATION(8)
```

Listing Specific Data Set Information in a Catalog - the ALL, NAME, and HISTORY Operands

Use the ALL operand to list each data set's:

- Name
- Owner identification
- Creation date
- Expiration date
- Entry type
- Volume serial number
- Device type.

You need not specify any further information. To list only the name of each data set, use the NAME operand. To list only the name of each data set, the owner identification, creation date and expiration date, use the HISTORY operand.

EXAMPLES

```
To list each data set's name, owner identification, creation
date, expiration date, entry type, volume serial number(s),
and device type(s) in a catalog, enter:

LISTCAT ALL

To list only each data set's name, enter:

LISTCAT NAME

To list only each data set's name, owner identification,
creation date, and expiration date, enter:

LISTCAT HISTORY
```

If you don't enter either LEVEL or ENTRIES, the system displays only the entries associated with the user ID.

LISTDS Command

Use the LISTDS command to display the attributes of specific data sets at your terminal. Each data set you specify must be currently allocated or available from the system catalog, and must reside on a currently-active volume.

Using LISTDS, you can display the attributes of one or more data sets. For example, you can use an asterisk (*) in place of any qualifier except the first. When you specify an asterisk, the system searches for all catalogued data sets whose names contain the qualifiers you specified. For example, if you wanted to list the attributes of all the data sets starting with MYDATA.TEXT, specify:

```
LISTDS MYDATA.TEXT.*
```

Using the LISTDS command, you can display:

- The volume identifier (VOLID) of the volume the data set resides on.
- The logical record length (LRECL), the block size (BLKSIZE), and for non-VSAM data sets, the record format (RECFM) of the data set.
- The data set organization (DSORG). For an explanation of the possible values for DSORG and their meanings, see "Listing the History of a Data Set - the HISTORY Operand" earlier in this chapter.
- The date the data set was created.
- The date the data set will expire.
- For non-VSAM data sets, whether or not the data set is password-protected.
- File name and disposition.
- The members in a partitioned data set, including any aliases.
- Non-VSAM data set control blocks (DSCBs)
- The information from the data sets in a particular user catalog.
- A list of information about data sets with a common high-level qualifier.

You can specify a list of data set names when using any of the operands of the LISTDS command.

For more information on the LISTDS command, see *TSO/E Command Language Reference*.

Listing Data Set VOLID, LRECL, BLKSIZE, RECFM and DSORG

You can list the VOLID, LRECL, BLKSIZE, RECFM, and DSORG of a specific data set by using the LISTDS command followed by a data set name.

EXAMPLE

```
To list the VOLID, LRECL, BLKSIZE, RECFM, and DSORG of
data set PARTS.DATA, enter:

LISTDS PARTS.DATA
```

You can also list the VOLID, LRECL, BLKSIZE, RECFM, and DSORG of a number of specific data sets by entering the LISTDS command followed by the data sets' names enclosed in parentheses and separated by commas or blanks.

EXAMPLE

```
To list the VOLID, LRECL, BLKSIZE, RECFM, and DSORG of
data sets PARTS.DATA, ART.TEXT, JONES.CNTL, enter:

LISTDS (PARTS.DATA,ART.TEXT,JONES.CNTL)
```

Listing Data Sets' Creation and Expiration Dates - the HISTORY Operand

You can list the date a data set was created, the date the data set expires, and for non-VSAM data sets, whether or not the data set is password-protected by specifying the data set name followed by the HISTORY operand. Be sure that you separate the operands by a comma or a blank.

EXAMPLE

```
To list the creation and expiration dates, and whether or
not non-VSAM data set PARTS.DATA is password-protected,
enter:

LISTDS PARTS.DATA HISTORY
```

Listing Data Sets' Associated DDNAMEs and Dispositions - the STATUS Operand

You can list the DDNAME currently associated with a data set, and that data set's disposition by specifying the data set name followed by the STATUS operand.

EXAMPLE

```
To list the DDNAME currently associated with data set
PARTS.DATA, and its disposition, enter:

LISTDS PARTS.DATA STATUS
```

Listing the Members of a Partitioned Data Set - the MEMBERS Operand

You can list all of the members in a partitioned data set, including any aliases, by specifying the data set name followed by the MEMBERS operand.

EXAMPLE

```
To list all of the members in partitioned data set PARTS.DATA,
including any aliases, enter:

LISTDS PARTS.DATA MEMBERS
```

Listing the DSCB for a Non-VSAM Data Set - the LABEL Operand

You can list the data set control block (DSCB) associated with a non-VSAM data set by specifying the data set name followed by the LABEL operand.

EXAMPLE

```
To list the DSCB associated with non-VSAM data set PARTS.DATA,
enter:

LISTDS PARTS.DATA LABEL
```

This operand is only applicable to non-VSAM data sets on direct access devices. The list will be displayed in hexadecimal notation.

Listing the Information in a User Catalog for a Data Set - the CATALOG Operand

To list the information associated with a data set in a user catalog, enter the LISTDS command and specify:

- The data set name
- The CATALOG operand followed by the catalog name enclosed in parentheses.

EXAMPLE

```
To list the information associated with data set PARTS.DATA
in user catalog, USECAT5, enter:

LISTDS PARTS.DATA CATALOG(USECAT5)
```

This operand is only required for data sets whose information is in catalogs other than the ones specified in the STEPCAT JCL statements or the catalogs implied by the data sets' highest-level qualifier.

Listing Information for Commonly Owned Data Sets - the LEVEL Operand

You can list information for data sets that have a common high-level qualifier by specifying the high-level qualifier followed by the LEVEL operand. These could be data sets that are owned by a particular user.

EXAMPLE

```
To list all the data sets whose high-level qualifier is
D00ABC1, that is, all data sets owned by the user
associated with user ID D00ABC1, enter:

LISTDS 'D00ABC1' LEVEL
```

When you specify this operand, the data set name cannot contain an asterisk. You may specify only one data set list with this operand.

Chapter 9. Protecting Data Sets

You can protect a data set by using the PROTECT command, which allows you to specify a type of access through the use of a password. You can request that the system restrict access to a data set to only those who meet specific access requirements.

The system allows four types of access to a data set, which are represented by four operands on the PROTECT command. The operands are PWREAD, PWWRITE, NOPWREAD, and NOWRITE.

You protect a data set by assigning it a password that a user must specify to gain access to the data set. You can assign one or more passwords to a data set. A password consists of one through eight alphameric characters. The system allows you two attempts to supply a correct password. After two attempts, the system issues a message indicating that the password is invalid and you cannot access the data set.

PROTECT Command

Use the PROTECT command to prevent unauthorized access to a non-VSAM data set. (Use the Access Method Services ALTER and DEFINE commands to protect VSAM data sets. These commands are described in *Access Method Services.*)

Using the PROTECT command, you can:

- Specify the type of access you want to allow to a data set
- Specify a password with which to protect your data set
- Add an extra password to a data set
- Replace an existing password with a new password
- Delete an existing password or access type
- Display the number of times the data set has been accessed, the access type, and any optional security information for a data set.

Each password is stored as part of an entry in a password data set, which resides on the system residence volume. The first password entered in the password data set is called the control entry. This control entry must be specified for each access

of the password data set via the PROTECT command, with one exception: the LIST operand of the PROTECT command does not require the control entry.

If you have allocated a data set either during a LOGON procedure or via the ALLOCATE command, you cannot protect it by specifying the PROTECT command. If you want to protect an allocated data set, you have to deallocate it via the FREE command before you can protect it via the PROTECT command.

If you omit a required password when using the PROTECT command, the system will prompt you for it. If your terminal is equipped with the print-inhibit feature, the system will disengage the printing mechanism at your terminal while you enter the password in response. However, the print-inhibit feature is not used if the prompting is for a new password.

For more information on the PROTECT command, see *TSO/E Command Language Reference*.

Specifying the Type of Access to a Data Set - the PWREAD, NOPWREAD, PWWRITE, NOWRITE Operands

You can specify one of the four types of access to a data set: password read (PWREAD), no password read (NOPWREAD), password write (PWWRITE), or no password write (NOWRITE). To specify a particular type of access to a data set, enter the PROTECT command and specify:

- The data set name.
- The type of access - PWREAD, NOPWREAD, PWWRITE, or NOWRITE:

  PWREAD
  You must specify the password before you can read the data set.

  NOPWREAD
  You can read the data set without specifying the data set's password.

  PWWRITE
  You must specify the data set's password before you can write to the data set.

  NOWRITE
  You cannot write to the data set.

  If you specify a password but do not specify a type of access, the default is:

  - NOPWREAD PWWRITE if the data set does not have any existing access restrictions
  - The existing type of access if a type of access has already been established.

  The system does not support a combination of NOPWREAD and NOWRITE. If you specify NOPWREAD and NOWRITE the system uses the values NOPWREAD and PWWRITE as defaults.

EXAMPLES

```
To specify that a password is needed to read PARTS.DATA,
enter:

PROTECT PARTS.DATA PWREAD

To specify that a password is not needed to read PARTS.DATA,
enter:

PROTECT PARTS.DATA NOPWREAD

To specify that a password is needed to write to PARTS.DATA,
enter:

PROTECT PARTS.DATA PWWRITE


To specify that a password is not needed to write to PARTS.DATA,
enter:

PROTECT PARTS.DATA NOWRITE
```

Specifying a Password for a Data Set - the ADD Operand

To specify a password with which to protect a data set that is not already protected, enter the PROTECT command and specify:

- The data set name
- The ADD operand with the password enclosed in parentheses.

EXAMPLE

```
To protect data set PARTS.DATA for the first time using
POODLE as the password, enter:

PROTECT PARTS.DATA ADD(POODLE)
```

To add an additional password to a data set that is already password- protected, enter the PROTECT command as shown in the previous example.

EXAMPLE

```
To add password PHYDEAUX to the password data set for
PARTS.DATA, enter:

PROTECT PARTS.DATA ADD(PHYDEAUX)
```

Replacing a Password for a Data Set - the REPLACE Operand

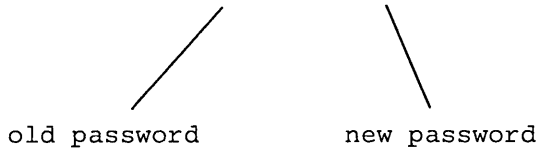
To replace one of a data set's passwords with a new password, enter the PROTECT command:

- Specify the data set name
- Enter the REPLACE operand
- Immediately enclose in parentheses:
  - The password you want to replace, followed by a blank
  - The new password.

EXAMPLE

To replace password POODLE for data set PARTS.DATA with the new password RUTABAGA, enter:

```
PROTECT PARTS.DATA REPLACE(POODLE RUTABAGA)
```



When you specify the REPLACE function of the PROTECT command, the default type of access is that of the entry being replaced.

Deleting Password Information for a Data Set - the DELETE Operand

For this and the following two examples, assume that the control entry for PARTS.DATA is RAKE. To delete one of a data set's passwords, enter the PROTECT command:

- Specify the data set name
- Enter a slash (/) immediately following the data set name
- Immediately specify the control entry
- Specify the DELETE operand with the password enclosed in parentheses.

EXAMPLE

To delete data set PARTS.DATA's password, RUTABAGA, enter:

```
PROTECT PARTS.DATA/RAKE DELETE(RUTABAGA)
```

To delete an access type from those specified for a data set, follow the instructions above, specifying the access type with the DELETE operand.

EXAMPLE

To delete data set PARTS.DATA's access type NOWRITE, enter:

```
PROTECT PARTS.DATA/RAKE DELETE(NOWRITE)
```

Displaying Password Information for a Data Set - the LIST Operand

To display the number of times a data set has been accessed, the access type, and any optional security information for the data set, enter the PROTECT command and:

- Specify the data set name
- Enter the LIST operand with the data set's password enclosed in parentheses.

EXAMPLE

To list the password information for data set PARTS.DATA, enter:

```
PROTECT PARTS.DATA LIST(POODLE)
```

RACF Commands

You can also protect a data set using the Resource Access Control Facility (RACF). RACF is a program product and you can obtain it for a license fee.

Using RACF you protect a data set by issuing the ADDSD command and its appropriate operands. There are a number of other protection-related tasks you can perform using RACF commands. For more information on the how to protect resources using RACF, see *MVS Resource Access Control Facility (RACF) Command Language Reference*. Below are brief discussions on how to RACF protect a data set, and how to list the protection attributes of a data set.

Protecting a Data Set Using RACF

You can RACF protect a data set by issuing the ADDSD RACF command followed by the name of the data set. To protect data set PARTS.DATA using the default values for the ADDSD command, enter:

```
ADDSD PARTS.DATA
```

Listing a Data Set's RACF Protection Attributes

You can list the RACF protection attributes for a data set at your terminal by issuing the LISTDSD RACF command, followed by the DATASET operand with the name of the data set enclosed in parentheses. To list data set PARTS.DATA's RACF protection attributes, enter:

```
LISTDSD DATASET(PARTS.DATA)
```

Chapter 10. Releasing Data Sets

Use the FREE command to release (deallocate) previously allocated data sets that you no longer need to use.

There is a maximum number of data sets that the system can allocate to your user ID at any one time. The system does not automatically release any data sets that are allocated as a result of issuing the LOGON or ALLOCATE commands. To avoid reaching the maximum limit of allocated data sets, and being denied access to desired resources, use the FREE command to release data sets that you no longer need.

FREE Command

Using the FREE command, you can request that the system:

- Release all dynamically allocated data sets, files, and attribute lists that are not currently in use.
- Release one or more specific data sets.
- Send SYSOUT data sets to a specific printer or destination.
- Place a data set in a HOLD queue to print later or view only.
- Not place a data set in a HOLD queue.
- Retain the data set after you release it.
- Delete the data set after you release it.
- Retain the data set in a catalog after you release it.
- Remove the data set from a catalog after you release it. The system still retains the data set.

When a command terminates, the system does not automatically deallocate data sets that the command dynamically allocated. You must explicitly release dynamically allocated data sets.

When using the FREE command, you must use one of the following operands when specifying a data set:

- DATASET
- DSNAME
- FILE
- DDNAME

For more information about the FREE command, see *TSO/E Command Language Reference*

Releasing All Your Data Sets not Currently in Use - the ALL Operand

Use the ALL operand to release all data sets, files, and attribute lists that were dynamically allocated to your user ID, but are not currently in use.

EXAMPLE

```
A data set allocated to your user ID is tying up the system, and
you are not sure which data set is causing the problem.
You decide to free all your data sets until you can correct
the problem.  You enter:

FREE ALL
```

Be aware that after issuing a FREE ALL, you may be unable to use ISPF or other programs that are dependent on the data sets just released. In this situation, you will have to LOGON again to continue your terminal session.

Releasing Specific Data Sets

You can release data sets by specifying:

- The data set name
- A group of specific data set names
- The file or ddname associated with a specific data set
- A group of file or ddnames associated with a specific data set.

To release a specific data set by specifying the data sets name, enter the FREE command:

- Specify the DATASET or DSNAME operand.
- Enclose the data set name in parentheses.
  - You must include the descriptive qualifier in the data set name.
  - You may include a member name in parentheses when specifying a partitioned data set.

EXAMPLE

```
To release data set PARTS.DATA, enter:

FREE DATASET(PARTS.DATA)
```

To release a group of specific data sets, enter the FREE command:

- Specify the DATASET or DSNAME operand
- Enclose in parentheses the data set names separated by commas.

EXAMPLE

```
To release data set PARTS.DATA, REPORT.TEXT, and
SIMIAN.CNTL(HAPE), enter:
FREE DATASET(PARTS.DATA,REPORT.TEXT,SIMIAN.CNTL(HAPE))
```

To release the data set(s) associated with a specific data definition name, enter the FREE command and specify:

- The FILE or DDNAME operand
- The data definition name enclosed in parentheses.

EXAMPLE

```
To release the data set associated with data definition name
SYSUT1, enter:
FREE FILE(SYSUT1)
```

To release the data sets associated with a specific group of data definition names, enter the FREE command and specify:

- The FILE or DDNAME operand
- The data definition names enclosed in parentheses.

EXAMPLE

```
To release the data set associated with data definition names
SYSUT1, SYSUT3, SYSPRINT, and SYSIN, enter:
FREE FILE(SYSUT1,SYSUT3,SYSPRINT,SYSIN)
```

Releasing SYSOUT Data Sets and Sending Them to a Work Station - the DEST Operand

To release a SYSOUT data set and send it to another work station for output processing, enter the FREE command and specify:

- The SYSOUT data set and the associated file name
- The DEST parameter with the name of the work station enclosed in parentheses.

EXAMPLE

```
To send the SYSOUT data set associated with the file name
SYSPRINT to work station LONDONW3 after releasing the data
set, enter:
FREE FILE(SYSPRINT) SYSOUT DEST(LONDONW3)
```

You can send more than one SYSOUT data set to a destination by following the same procedure for releasing multiple data sets.

If you omit this operand when releasing SYSOUT data sets, the data sets are directed to the work station you specified at the time you allocated the data set.

Releasing Data Sets and Placing Them in a HOLD Queue - the HOLD, NOHOLD Operands

You can send data to the HOLD queue if you want to view it with the option of printing it later or not printing it. To place a data set on a HOLD queue, enter the FREE command and specify:

- The data set
- The HOLD operand.

EXAMPLE

```
To place the data set associated with data definition name
SYSUT1 in a HOLD queue, enter:

FREE FILE(SYSUT1) HOLD
```

When you specify this operand, it overrides any HOLD/NOHOLD specification made when the data set was originally allocated, and it also overrides the default HOLD/NOHOLD specification associated with a SYSOUT class.

To ensure that the system not place a data set on a HOLD queue, enter the FREE command and specify:

- The data set
- The NOHOLD operand.

EXAMPLE

```
To keep the data set associated with data definition name
SYSUT1 from being placed on a HOLD queue, enter:

FREE FILE(SYSUT1) NOHOLD
```

Releasing Data Sets and Specifying Their Disposition

When you release a data set, you can also tell the system what you want done with the data set once it is released. You can specify one of four data set dispositions. They are:

- KEEP
- DELETE
- CATALOG
- UNCATALOG

For an explanation of the meaning of each of these four dispositions, see, "Specifying a Data Set's Disposition When Freed - the KEEP, CATALOG, UNCATALOG, and DELETE Operands" under the discussion of the ALLOCATE command.

To specify that the system is to retain the data set after you release it, enter the FREE command and specify:

- The data set or file
- The KEEP operand.

EXAMPLE

To retain file name SYSPRINT in the system after you release it, enter:

```
FREE FILE(SYSPRINT) KEEP
```

To specify that the system is to delete the data set after you release it, enter the FREE command and specify:

- The data set
- The DELETE operand.

EXAMPLE

To delete data set PARTS.DATA from the system after you release it, enter:

```
FREE DSNAME(PARTS.DATA) DELETE
```

The DELETE operand is not valid when releasing a data set that was allocated with a disposition of SHR, or when releasing a member of a partitioned data set.

The DELETE operand is the only valid disposition operand when you release a SYSOUT data set. It deletes the data set without printing it.

To specify that the system is to retain the data set in a catalog after you release it, enter the FREE operand and specify:

- The data set
- The CATALOG operand.

EXAMPLE

To retain the data set(s) associated with ddname SYSUT2 in a catalog after you release it, enter:

```
FREE FILE(SYSUT2) CATALOG
```

Note: if you use FREE FILE(SYSUT2) DELETE, all the data sets associated with it are deleted.

To specify that the system is to remove the data set(s) from a catalog after you release it, enter the FREE command and specify:

- The data set
- The UNCATALOG operand.

EXAMPLE

To remove the data set associated with ddname SYSUT2 from a catalog after you release it, enter:

```
FREE FILE(SYSUT2) UNCATALOG
```

Chapter 11. Deleting Data Sets

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

The system removes a catalog entry for a partitioned data set only after the entire partitioned data set is deleted. The system deletes members of a partitioned data set by removing the member name from the directory of the partitioned data set.

You must explicitly delete both members of a partitioned data set and their aliases. When you delete a member, the system does not delete any aliases for that member. When you delete an alias, the system does not delete its associated member.

Before you delete a protected non-VSAM data set, use the PROTECT command to delete the data set's password from the password data set. This will prevent you from finding insufficient space in the password data set when you want to add future entries.

For more information about the DELETE command, see *TSO/E Command Language Reference*.

DELETE Command

Using the DELETE command, you can specify:

- The name of the entry you want to delete. You can also specify an entry's password, if the entry is password-protected.
- The name of the catalog that contains the entries you want to delete.
- The name of the DD statement that identifies either the volume on which the entry to be deleted resides or the entry itself.
- Whether or not to delete a data set depending on its retention period.
- Whether or not to scratch (remove) a non-VSAM data set's entry from the volume table of contents (VTOC) on the volume on which the data set resides.
- That the system is to delete an alias entry.

Deleting a Data Set from the System

You can delete a data set by specifying either:

- The data set name
- The data set name and its passwords
- A number of data sets
- The volume identifier from the data set's DD statement.

To delete from the system a data set that doesn't have a password, enter the DELETE command and specify the data set name.

EXAMPLE

```
To delete data set PARTS.DATA, enter:

DELETE PARTS.DATA
```

To delete a data set that is password-protected, enter the DELETE command:

- Specify the data set name
- Immediately enter a slash, followed by the data set's password.

EXAMPLE

```
To delete data set PARTS.DATA with the password BLOBULNT,
enter:

DELETE PARTS.DATA/BLOBULNT
```

To delete a number of data sets, enter the DELETE command:

- Specify each data set name separated by a comma or blank
- Enclose the group of names in parentheses.

EXAMPLE

```
To delete data sets PARTS.DATA, PARTS.TEXT, PARTS.ASM, and
PARTS.CNTL, enter:

DELETE (PARTS.DATA,PARTS.TEXT,PARTS.ASM,PARTS.CNTL)
```

To delete PARTS.DATA using the volume identifier from the data definition statement associated with the data set, enter the DELETE command and specify:

- The data set name
- The FILE operand.

EXAMPLE

```
To delete data set PARTS.DATA if the DD statement SYSIN
contains the volume identifier on which the volume
PARTS.DATA resides, enter

DELETE PARTS.DATA FILE(SYSIN)
```

Deleting a Data Set Entry from a Catalog - the CATALOG Operand

To delete a data set entry from a catalog, enter the DELETE command:

- Specify the CATALOG operand
- Enclose the name of the catalog in parentheses.

EXAMPLE

```
To delete data set PARTS.DATA from the catalog USERCAT1,
enter:

DELETE PARTS.DATA CATALOG(USERCAT1)
```

When deleting an entry for a password-protected data set, you may either specify the password for the data set itself, or specify the password to the catalog that contains the entries you want to delete. If catalog USERCAT1 had a master password of PFEFFER, you could specify:

```
DELETE PARTS.DATA CATALOG(USERCAT1/PFEFFER)
```

Deleting a Data Set Based On Its Retention Period - the PURGE/NOPURGE Operands

When you allocated a particular data set, you may have specified a retention period after which the system would automatically delete the data set. To delete a data set regardless of its retention period, use the PURGE (abbreviation PRG) operand.

EXAMPLE

```
If data set PARTS.DATA still has 30 days left in its retention
period, but you want to delete it now,enter:

DELETE PARTS.DATA PURGE
```

The NOPURGE operand (abbreviation NPRG) specifies that the system should delete a data set only if its retention period has expired. NOPURGE is a default with the DELETE command.

Deleting and Scratching a Data Set's VTOC Entry - the SCRATCH/NOSCRATCH Operand

The volume table of contents (VTOC) is a data set that describes the contents of a direct access volume. It is used to account for each data set and available space on the volume. The VTOC includes the name, location, organization, and other control information about each data set stored on the volume.

When you delete a data set, you scratch (remove) its entry from the volume table of contents (VTOC) on the volume on which it resides. The SCRATCH operand is a default on the DELETE command.

If you do not want to scratch (remove) an entry from the VTOC, enter the DELETE command and specify:

- The data set name
- The NOSCRATCH (abbreviation NSCR) operand.

EXAMPLE

```
To delete data set PARTS.DATA, but not remove its entry from
the VTOC, enter:

DELETE PARTS.DATA NOSCRATCH
```

If you use the NOSCRATCH operand with the DELETE command, you remove the catalog entry for the data set. The data set still exists, and to reaccess it, you must specify the volume on which it resides.

Deleting an Alias Entry - the ALIAS Operand

To delete an alias entry, issue the DELETE command and specify:

- The alias name of the member of the partitioned data set
- The ALIAS operand.

EXAMPLE

```
To delete alias entry, JCL.CNTL(ALIAS1), enter:

DELETE JCL.CNTL(ALIAS1) ALIAS
```

Section III: Preparing and Running a Program

After you have written a program, or source code, the next step is to execute it. To execute a program, the processor needs to have the program's source statements changed into a format that the processor can interpret. This process is called compiling or assembling. You assemble assembler language statements and you compile any other language's statements. The compiler or assembler creates certain data sets for you that assist you in diagnosing problems, known as listings, and data sets that contain the compiled or assembled code. The data sets that contain the compiled or assembled code are called **object modules**.

Some programs require data from system libraries or from other programs in order to complete execution. When this data is in object module form, you can link edit a number of object modules together using TSO. A group of object modules linked together is known as a **load module**. The linkage editor also produces diagnostic listings. You can use TSO to bring the load or object modules into real storage and execute them while you control the process at your terminal. This is known as executing a program in the foreground.

You can use JCL statements to set up the proper conditions for running your program, and then use TSO to submit the JCL statements to the processor. The program is put on a queue and executed at a later time, leaving you free to do other work at your terminal. This process is called executing a job in the background, also known as a batch environment.

This section describes how to:

- Compile or assemble source code
- Link object modules together to form load modules
- Load programs into real storage and execute them
- Submit and monitor a batch job
- Execute TSO commands in a background or batch environment
- Invoking commands or programs from an unauthorized environment.

Chapter 12. Compiling and Assembling Code

The following figures show the steps and commands you use when executing a program. Figure 12-1 presents the commands you can use when executing a program, showing what each command accomplishes. These commands are discussed later in this section. Figure 12-2 shows in diagram form the compile, link edit and execution of a PLI language program. The commands, highlighted in dark print, are discussed later in this section. The modules and listings produced by these commands are shown in the boxes. Figure 12-3 shows what you would see at your terminal when you execute this program. The commands you enter are in lower case type and the system responses are in upper case type.

Command	Compiles	Link Edits	Loads	Begins Execution	Notes
RUN	X	X	X	X	For use ONLY with certain program products
ASM, FORT, PLI, COBOL	X				Command used to compile; depends on programming language used.
LINK		X			Produces linkage editor listings to help test and debug a program
LOADGO		X	X	X	
CALL			X	X	Program must be in executable load module form and must be a member of a PDS

Figure 12-1. Commands Used to Execute a Program

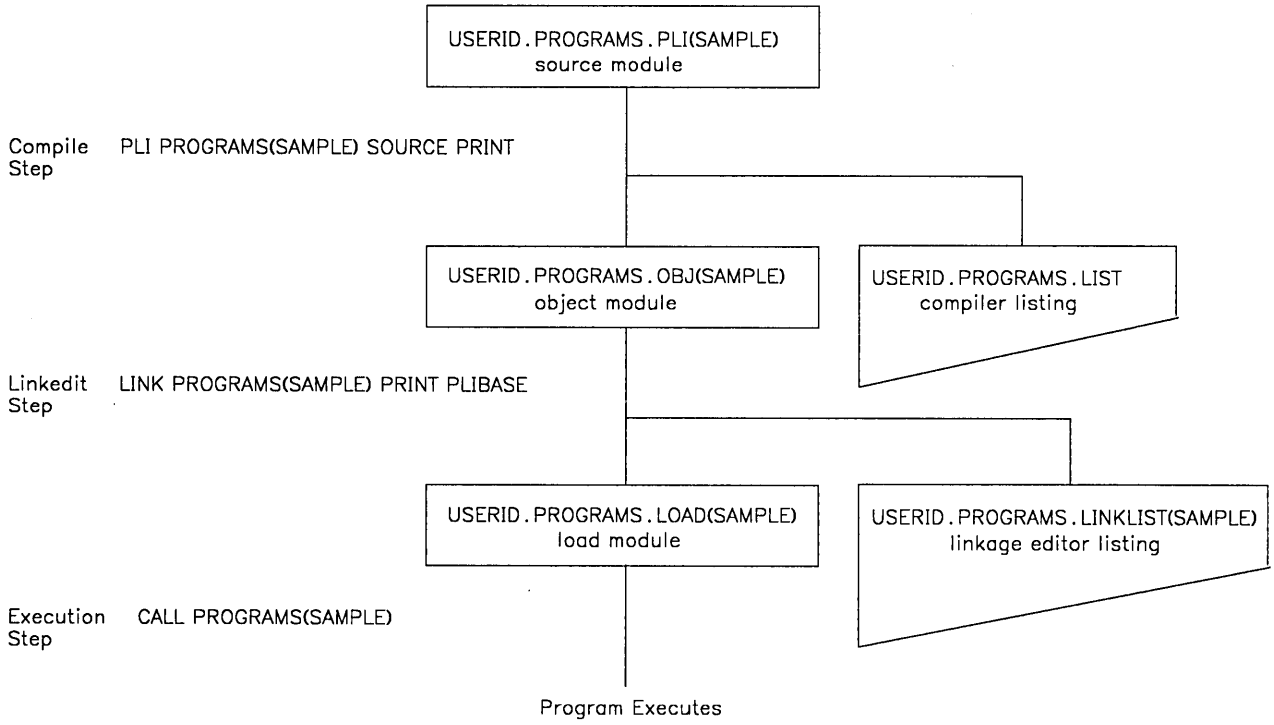


Figure 12-2. Compiling and Link-Editing a Single Program

```
READY
pli programs(sample) source print
PL/I OPTIMIZER V1 R4.0
OPTIONS SPECIFIED
S;

NO MESSAGES PRODUCED FOR THIS COMPILATION
COMPILE TIME  0.00 MINS    SPILL FILE=      0 RECORDS, SIZE  4051
READY
list programs.list
IKJ52827I PROGRAMS.LIST
PL/I OPTIMIZING COMPILER      VERSION 1  RELEASE 4.0
OPTIONS SPECIFIED
S;
PL/I OPTIMIZING COMPILER      TSOCALL =
          SOURCE LISTING
NUMBER
10000    TSOCALL:
           PROCEDURE OPTIONS(MAIN);
63500    DECLARE (FILEOUT) FILE;/* PLI OUTPUT FILE       */
71000       PUT FILE (FILEOUT) EDIT ('THIS PLI PROGRAM IS EXECUTING')
                          (A)
90000     END TSOCALL;
PL/I OPTIMIZING COMPILER         TSOCALL:
NO MESSAGES PRODUCED FOR THIS COMPILATION
COMPILE TIME      0.00 MINS     SPILL FILE=      0 RECORDS, SIZE 4051

READY
link programs(sample) print plibase

READY
list programs.linklist(sample)
IKJ52827I  PROGRAMS.LINKLIST(SAMPLE)

     H96-LEVEL LINKAGE EDITOR OPTIONS SPECIFIED TERM
           DEFAULT OPTION(S) USED _SIZE=(262144,49152)
     ****SAMPLE   NOW REPLACED IN DATA SET    AMODE 24
     RMODE IS 24
     AUTHORIZATION CODE IS        0.
READY
alloc f(fileout) dsn(*)
READY
call programs(sample)

  THIS PLI PROGRAM IS EXECUTING

READY
```

Figure 12-3. Terminal Session Showing Execution of a Single Program

The Compile Commands

After you write your source code, you must compile it into object code and place it in an object module (see Figure 12-2). The command you use to compile your source code depends on which programming language you are using. The third, or descriptive qualifier in the source code data set name should identify the programming language used to help the compiler find and process the data.

There are a number of versions of some compilers. You can usually find information on how to use your version of a compiler with the compiler code or in its accompanying reference material. This manual will describe the commands used with several common compilers.

The RUN command, which is designed specifically for use with certain program products, will compile, load, and execute source statements. These program products are listed in Figure 12-4. If you cannot use the RUN command, need the diagnostic information provided by the other commands, or need to link your program to other modules, you will have to issue separate commands to link edit your object modules into load modules, load these modules into main storage, and begin program execution. These commands, listed in Figure 12-1, are described in the following chapters.

Some commands used to compile or assemble source statements are:

- ASM
- COBOL
- FORT
- PLI

ASM command

The ASM command is provided as part of the optional TSO ASM Prompter program product, which is available for a license fee. Use the ASM command to process assembler language data sets and produce object modules. The prompter requests required information and enables you to correct your errors at the terminal. See *OS/TSO Assembler Prompter User's Guide*, for detailed information on this command.

For those who use an MVS/XA system, you need to use the Assembler H compiler to process your assembler language statements. For information on how to use and invoke this compiler, see *Assembler H Version 2 Application Programming: Language Reference.*

COBOL Command

The COBOL command is provided as part of the optional COBOL Prompter program product, which is available for a license fee. Use the COBOL command to compile American National Standard (ANSI) COBOL programs. This command reads and interprets parameters for the OS Full American National Standard COBOL Version 3 or Version 4 compiler and prompts you for any information that you have omitted or entered incorrectly. It also allocates required data sets and passes parameters to the compiler. See *IBM OS (TSO)*

COBOL Prompter Terminal User's Guide and Reference, for detailed information on this command.

FORT command

The FORT command is provided as part of the optional TSO FORTRAN Prompter program product, which is available for a license fee. Use the FORT command to compile a FORTRAN IV (G1) program. You will be prompted for any information that you have omitted or entered incorrectly. The FORT command also allocates required data sets and passes parameters to the FORTRAN IV (G1) compiler. See *IBM System/360 OS (TSO) Terminal User's Supplement for FORTRAN IV (G1) Processor and TSO FORTRAN Prompter*, for detailed information on this command.

PLI command

The PLI command is provided as part of the optional PL/I Optimizing compiler program product, which is available for a license fee. Use the PLI command to invoke the PL/I Optimizing compiler. The prompter will allocate required data sets and prompt you for any information that you have omitted or entered incorrectly, then it will pass control to the compiler. See *OS PL/I Optimizing Compiler: TSO User's Guide*, for detailed information on this command. The program product includes the PL/I Prompter.

RUN Command

Use the RUN command to compile, load, and execute the source statements in the data set that you are editing. The RUN command is designed specifically for use with certain program products; it selects and invokes the particular program product needed to process your source statements. Figure 12-4 shows which program product is selected to process each type of source statement.

SOURCE	PROGRAM PRODUCT
ASSEMBLER	MVS/370: Assembler F and TSO Assembler Prompter MVS/XA: Assembler H Version 2
COBOL	OS/VS COBOL Release 2.4 and TSO COBOL Prompter
FORTRAN	TSO FORTRAN Prompter and FORTRAN IV (G1) VS FORTRAN
PL/I	PL/I Checkout Compiler or PL/I Optimizing Compiler
VSBASIC	VSBASIC
Note: User-defined data set types can be executed under the RUN subcommand of the EDIT command if a prompter name was specified at system generation time. The RUN command will not recognize these same data set types.	

Figure 12-4. Source/Program Product Relationship

Using the RUN command, you can specify:

- Which data set contains the source code or object module you want to process and which assembler or compiler you want to use to process your source statements.
- A string of up to 100 characters that is to be passed as parameters to the program you are running.
- The libraries that contain subroutines your program will use during its execution.
- What options you want to use with a VSBASIC program.

The RUN subcommand of the EDIT command is very similar to the RUN command, and you may use it similarly to the way you use the RUN command. Before you use the RUN command, keep in mind:

- Any data sets required by your problem program may be allocated before you enter EDIT mode or may be allocated using the ALLOCATE subcommand.
- If you wish to enter a value for 'parameters,' you should enter this prior to any of the other keyword operands.

For more information about the RUN command, see *TSO/E Command Language Reference*.

Compiling Source Code Statements

To compile a data set, issue the RUN command:

- Specify the data set name
- Optionally specify the compiler.

If the data set name follows standard naming conventions and includes the name of the programming language used in the descriptive qualifier, you do not need to specify a compiler with the RUN command. If the system cannot determine which compiler to use, you will be prompted for more information.

EXAMPLE

```
To compile the source code statements in member SAMPLE of
data set PROG1.PLI, enter:

RUN PROG1.PLI(SAMPLE)
```

Passing Parameters When Compiling

To pass parameters to a program, issue the RUN command and specify:

- The data set name
- The parameters enclosed in single quotes
- The compiler to be used, if the data set name does not follow standard naming conventions.

EXAMPLE

```
To specify 13, FRIDAY, CAT as parameters to pass to program
PROG1, which was written in PLI, enter:

RUN PROG1 '13 FRIDAY CAT' PLI
```

Specifying a Subroutine Library When Compiling - the LIB Operand

If you use subroutines that reside in a library data set, use the LIB operand. Issue the RUN command and specify:

- The data set name
- The LIB operand with the name of the library data set enclosed in parentheses.

EXAMPLE

```
When running COBOL program PROG1, you need to use library
SUBS.LOAD because it contains the subroutines your program
calls:

RUN PROG1 COBOL LIB(SUBS.LOAD)
```

You can also use the LIB operand when running the assembler, Fortran, and PLI compilers.

Specifying VSBASIC Compiler Options

When running the VSBASIC compiler, you can use a number of options with the TSO RUN command. To do this, use the options you want following the data set's descriptive qualifier (in this case, VSBASIC). Following is a list of operands and a description of each:

LPREC
Specifies that long precision arithmetic calculations are required by the VSBASIC program.

SPREC
Default specifying that short precision arithmetic calculations are adequate for a VSBASIC program.

TEST
Specifies that testing of a VSBASIC program is to be performed.

NOTEST
Default specifying that the TEST function is not desired with a VSBASIC program.

STORE
Specifies that the VSBASIC compiler is to store an object program.

NOSTORE
Default specifying that the VSBASIC compiler is not to store an object program.

GO
Default specifying that the VSBASIC program is to receive control after compilation.

NOGO
Specifies that the VSBASIC program will not receive control after compilation.

SIZE(value)
Specifies the number of one thousand-byte blocks of VSBASIC user area where value is an integer of one-to-three digits.

PAUSE
Specifies that the VSBASIC compiler is to prompt to the terminal between program chains, giving the user the chance to change certain compiler options.

NOPAUSE
Default specifying no prompting between program chains.

SOURCE
Default specifying that new VSBASIC source code is to be compiled.

OBJECT

Specifies that the data set name entered is a fully-qualified name of an object data set to be executed by the VSBASIC compiler.

Chapter 13. Link Editing Programs

The linkage editor, invoked by the LINK command, provides a great deal of information to help you test and debug a program. This information includes a cross-reference table and a map of the module that identifies the location of control sections, entry points, and addresses. You can specify which of these types of information you want by including the LIST, MAP, and XREF operands with the LINK command. You can have this information listed at your terminal or saved in a data set.

LINK Command

Use the LINK command to invoke the linkage editor service program. Basically, the linkage editor converts one or more object modules (the output modules from a compiler or the assembler) into a load module that is suitable for execution.

You can specify all the linkage editor options explicitly or you can accept the default values. The default values are satisfactory for most uses. By accepting the default values, you simplify the use of the LINK command. This chapter does not describe every operand available with the LINK command. For more information on the LINK command, see *TSO/E Command Language Reference.*

In some cases, you might want to use the LOADGO command as an alternative to the LINK command. If the module that you want to process has a simple structure (that is, it is self contained and does not pass control to other modules), you do not require the extensive listings produced by the linkage editor, or you do not want a load module, the LOADGO command might better serve your needs.

Using the LINK command, you can specify:

- The names of one or more data sets containing your object modules and/or linkage editor control statements
- The name of the partitioned data set that will contain the load module after processing by the linkage editor
- One or more names of library data sets the linkage editor is to search to locate object modules referred to by the module being processed; that is, to resolve external references
- Whether or not you want the system to produce linkage editor listings and place them in a data set you specify or at your terminal

- Whether or not you want the system to produce and include in the PRINT data set:
  - A map of the output module consisting of the control sections, the entry names, and for overlay structures, the segment number
  - A list of all linkage editor control statements
  - A cross-reference table
- Whether or not you want the system to place the symbol tables created by the assembler and contained in the input modules into the output module
- Whether or not you want error messages directed to your terminal as well as to the PRINT data set.

Creating a Load Module

To create a load module by link editing an object module, issue the LINK command and specify the object module name. If you want to link more than one object module, issue the LINK command and list the object modules' names separated by commas and enclosed in parentheses. For example, to link edit sequential data set object modules FIRST.OBJ, SECOND.OBJ and FIFTH.OBJ, enter:

```
LINK (FIRST,SECOND,FIFTH)
```

The specified data sets will be concatenated within the output load module in the sequence that they are included in the list of data set names. If there is only a single name in the data-set-list, parentheses are not required around the name. To link edit object modules in members, enclose the member names in parentheses.

EXAMPLE

```
To link members ONE, TWO, and THREE of data set
D00ABC.SAMPLE.OBJ, enter:

LINK (SAMPLE(ONE),SAMPLE(TWO),SAMPLE(THREE))
```

If the data set name is D00ABC.OBJ, enclose the member names within two pairs of parentheses, for example:

```
LINK ((ONE),(TWO),(THREE))
```

You may substitute an asterisk (*) for a data set name to indicate that you will enter linkage editor control statements from your terminal. The system prompts you to enter the control statements, which should begin in column 2. Press the **ENTER** key after you type in each control statement. Enter a null line to indicate the end of your control statements.

When you link edit an object module that is a sequential data set, the linkage editor generates a load module in a partitioned data set and assigns the load module the member name of TEMPNAME.

To store the output from the LINK command in a data set, use the LOAD operand with the user-specified qualifier of the data set enclosed in parentheses.

EXAMPLE

```
To store the results of the previous example in data set
SALES.LOAD(ONE), enter:

LINK (FIRST,SECOND,FIFTH) LOAD(SALES(ONE))
```

If you omit the LOAD operand, the system generates a name according to the data set naming conventions.

Resolving External References - the LIB Operand

To have the system search specific library data sets to resolve any external references in your program, issue the LINK command and specify:

- The data set name
- The LIB operand, followed by the name of the library data set(s) enclosed in parentheses.

When you specify more than one name, the names must be separated by a valid delimiter. If you specify more than one name, the data sets are concatenated to the file name of the first data set in the list. For control statements, the first data set in the list must be preallocated with the ddname or file name SYSLIB before issuing the LINK command. If you specify more than one name, the data sets concatenated to the file name of the first data set lose their individual identity. See *MVS/XA System Programming Library: System Macros and Facilities* for details on dynamic concatenation.

EXAMPLE

```
To have the system search library data sets MYLIB, YOURLIB,
and OURLIB to resolve any external references, enter:

LINK SECOND LIB(MYLIB,YOURLIB,OURLIB)
```

Producing Output Listings - the PRINT Operand

To have the system produce linkage editor output listings and place them into a data set, use the PRINT operand with the name of the data set enclosed in parentheses.

EXAMPLE

```
To have the system produce linkage editor listings from
object modules FIRST.OBJ, SECOND.OBJ, and FIFTH.OBJ and
place them in data set PRINT.DATA, enter:

LINK (FIRST,SECOND,FIFTH) LOAD(SALES) PRINT(PRINT.DATA)
```

Use the PRINT operand with an asterisk enclosed in parentheses to have the system produce linkage editor output listings and display them at your terminal.

EXAMPLE

```
To have the system produce linkage editor listings from
object modules FIRST.OBJ, SECOND.OBJ, and FIFTH.OBJ and
display them at your terminal, enter:

LINK (FIRST,SECOND,FIFTH) LOAD(SALES) PRINT(*)
```

When you omit the data set name on the LOAD operand, the data set that is generated is named according to the data set naming conventions. Naming this data set is the default value if you specify the LIST, MAP, or XREF operand. If you want to have the listings displayed at your terminal, you may substitute an asterisk (*) for the data set name.

To have the system produce no linkage editor output listings, use the NOPRINT operand.

EXAMPLE

```
To have the system not produce linkage editor listings
from object modules FIRST.OBJ, SECOND.OBJ, and FIFTH.OBJ
enter:

LINK (FIRST,SECOND,FIFTH) LOAD(SALES) NOPRINT
```

NOPRINT causes the MAP, XREF, and LIST options to become invalid. NOPRINT is the default value if both PRINT and NOPRINT are omitted, and you do not use the LIST, MAP, or XREF operand.

Creating a Map of the Load Module - the MAP Operand

To have the system include a map of the output module use the MAP and PRINT operands. The output module consists of the control sections, the entry names, and (for overlay structures) the segment number in the PRINT data set. NOMAP, specifying that you do not want a map of the output module, is the default.

EXAMPLE

```
To have the system include a map of the output module from
object module SECOND into PRINT data set PRINT.DATA, enter:

LINK SECOND PRINT(PRINT.DATA) MAP
```

Producing a List of All Linkage Editor Control Statements - the LIST Operand

To have the system include a list of all linkage editor control statements and place them in the PRINT data set, use the PRINT and LIST operands. NOLIST, which specifies that you do not want a list of all linkage editor control statements, is the default.

EXAMPLE

```
To produce a list of all the linkage editor control
statements from object module SECOND.OBJ, and place them in
PRINT data set PRINT.DATA, enter:

LINK SECOND PRINT(PRINT.DATA) LIST

To produce a list of all the linkage editor control
statements from object module SECOND.OBJ, and place them in
data set SECOND.LINKLIST, where SECOND.OBJ contains both
linkage editor control statements and the object module,
enter:

LINK SECOND LIST
```

Producing a Cross Reference Table - the XREF Operand

To have the system create a cross reference table and place it in the PRINT data set, use the PRINT and XREF operands. The cross reference table includes the module map and a list of all address constants referring to other control sections. Since the XREF operand includes a module map, both XREF and MAP cannot be specified for a particular LINK command. NOXREF, which specifies that you do not want a cross reference table in the PRINT data set, is the default.

EXAMPLE

```
To produce a cross reference table from object module
SECOND.OBJ, and place it in PRINT data set PRINT.DATA,
enter:

LINK SECOND PRINT(PRINT.DATA) XREF
```

Producing a Symbol Table - the TEST Operand

To have the system place a symbol table created by the assembler and contained in the input modules into the output module, use the TEST operand. NOTEST, specifying that you do not want a symbol table, is the default.

EXAMPLE

```
To have the system place a symbol table created by the
assembler and contained in the input module SECOND into
the output module, enter:

LINK SECOND TEST
```

Sending Error Messages to Your Terminal - the TERM/NOTERM Operand

You can choose whether or not you want error messages directed to your terminal as well as the PRINT data set. TERM, which specifies that the system direct error messages to your terminal as well as to the PRINT data set, is a default. If you do not want error messages directed to your terminal as well as to the PRINT data set, use the NOTERM operand.

EXAMPLE

```
LINK SECOND NOTERM
```

Chapter 14. Loading and Executing Programs

When you run a program, the program must be placed into main storage, because programs reside most often on auxiliary storage. Placing a program into main storage is called **loading** the program. To load and execute a program, use either the LOADGO or CALL command. The LOADGO command loads object modules produced by a compiler or assembler, and load modules produced by the linkage editor. If you want to load and execute a single load module, the CALL command is more efficient.

LOADGO Command

Use the LOADGO command to load a compiled or assembled program into main storage and begin execution. The LOADGO command invokes the system loader to accomplish this function. The loader combines basic editing and loading services of the linkage editor and program fetch in one job step. Therefore, the *load* function is equivalent to the *link edit and go* function.

The LOADGO command does not produce load modules for program libraries, and it does not process linkage editor control statements such as INCLUDE, NAME, or OVERLAY. If you need to use these control statements, use the LINK and CALL commands.

The LOADGO command also searches a specified call library (SYSLIB) or a resident link pack area, or both, to resolve external references.

Using the LOADGO command, you can specify:

- The names of one or more data sets containing your object modules and/or load modules
- Parameters to pass to the program you execute
- Whether or not the system is to produce output listings and place them in a data set you specify or display them at your terminal
- One or more names of library data sets to be searched by the linkage editor to locate object modules referred to by the module being processed; that is, to resolve external references
- Whether or not the system is to send error messages to your terminal as well as to the PRINT data set

- Whether or not the system is to include a list of external names and their addresses in the PRINT data set
- Whether or not the system is to search the data set(s) you specified on the LIB operand to locate load modules to which the executing load module refers
- The external name for the loaded program's entry point
- The name you want to assign to the loaded program.

For more information about the LOADGO command see *TSO/E Command Language Reference.*

Loading and Executing Programs with No Operands

To load and execute the source code in a data set or a group of data sets, issue the LOADGO command and specify the data set name. The rightmost qualifier of the data set name must be OBJ or LOAD. When loading and executing the code in a single data set, specify the data set name without parentheses.

EXAMPLE

```
To load and execute the code in data set PAYROLL.LOAD, enter:

LOADGO PAYROLL.LOAD
```

When loading and executing the code in a group of data sets, specify the data set names in parentheses, separating each name with a comma. The names may be data set names, names of members of partitioned data sets, or both.

EXAMPLE

```
To load and execute the code in data sets FIRST.OBJ,
SECOND.OBJ and THIRD.LOAD, enter:

LOADGO (FIRST.OBJ,SECOND.OBJ,FIFTH.LOAD)
```

Passing Parameters when Loading and Executing Programs

To pass parameters to a program when loading and executing it, specify the parameter enclosed in single quotes following the data set name field.

EXAMPLE

```
To pass THE RAIN IN SPAIN as a parameter to the program in
data set FIFTH.LOAD, enter:

LOADGO FIFTH.LOAD 'THE RAIN IN SPAIN'
```

Requesting Output Listings when Loading and Executing Programs - the PRINT/NOPRINT and TERM/NOTERM Operands

To produce output listings and place them in a data set, use the PRINT operand with the data set name enclosed in parentheses. The NOPRINT operand, suppressing output listings, is the default. Note that the NOPRINT and MAP operands are mutually exclusive. The MAP operand, discussed below, puts data in the PRINT data set. Therefore, if you want the MAP information, you must also specify the PRINT operand with the LOADGO command.

EXAMPLE

```
To send the output listings from the program in data set
FIFTH.LOAD to data set OUT5.DATA, enter:

LOADGO FIFTH.LOAD PRINT(OUT5.DATA)
```

To produce output listings and send the output to your terminal, use the PRINT operand with an asterisk enclosed in parentheses.

EXAMPLE

```
To send the output listings from the program in data set
FIFTH.LOAD to your terminal, enter:

LOADGO FIFTH.LOAD PRINT(*)
```

All error messages are directed to your terminal as well as to the PRINT data set, as the TERM operand is a default with the LOADGO command. To direct all error messages only to the PRINT data set, not to your terminal, use the NOTERM operand.

EXAMPLE

```
To direct all error messages only to the print data set
OUT5.DATA, enter:

LOADGO FIFTH.LOAD PRINT(OUT5.DATA) NOTERM
```

Resolving External References when Loading and Executing Programs - the CALL/NOCALL and LIB Operands

To resolve external references, you must know in which data sets the code or data being referred to is kept. If you know the load module library data set(s) name, use the LIB operand with the name enclosed in parentheses to tell the system where to look to resolve any external references.

EXAMPLE

```
To search library data sets MYLIB, YOURLIB, and OURLIB,
in order to resolve any external references when executing
the program in FIFTH.LOAD, enter:

LOADGO FIFTH.LOAD LIB(MYLIB,YOURLIB,OURLIB)
```

The CALL operand is a default with the LOADGO command. The system will search for the data set(s) you specified on the LIB operand to locate the load modules to which the executing code refers.

Use the NOCALL operand following the LIB operand to prevent the system from searching the data set(s) you specified on the LIB operand.

EXAMPLE

```
To suppress searching for load modules within data set
MYLIB, YOURLIB, or OURLIB when executing the program
in FIFTH.LOAD, enter:

LOADGO FIFTH.LOAD LIB(MYLIB,YOURLIB,OURLIB) NOCALL
```

To include a list of external names and their addresses in the PRINT data set, use the PRINT, LIB and MAP operands. The NOMAP operand is the default and does not include the MAP information in the PRINT data set.

EXAMPLE

```
To resolve external references found in data set PROLIB,
and list them in data set OUT5.DATA, enter:

LOADGO FIFTH.LOAD PRINT(OUT5.DATA) LIB(PROLIB) MAP
```

Note that MAP and NOPRINT are mutually exclusive operands.

Specifying an Entry Point when Loading and Executing Programs - the EP Operand

To specify an external name for a program's entry point when loading and executing the program, use the EP operand with the entry point name enclosed in parentheses.

EXAMPLE

```
To specify START as the external name for the entry point
into the program in data set FIFTH.LOAD, enter:

LOADGO FIFTH.LOAD EP(START)
```

If the entry point of the loaded program is a load module, you must specify this operand.

Specifying Names when Loading and Executing Programs - the NAME Operand

To assign a name to a program in a data set, use the NAME operand followed by the name of the program.

EXAMPLE

```
To assign the name PROG3 to the program in FIFTH.LOAD,
enter:

LOADGO FIFTH.LOAD NAME(PROG3)
```

CALL Command

Use the CALL command to load and execute a program that exists in executable (load module) form. The program may be user-written or it may be owned by the system, for example, a compiler, sort, or utility program.

You must specify the name of the program (load module) to be processed. It must be a member of a partitioned data set. Also, you can pass parameters to the program.

For more information about the CALL command see, *TSO/E Command Language Reference.*

Loading and Executing Load Modules

To load and execute a load module in a member of a partitioned data set, specify the name of the data set with the name of the member enclosed in parentheses.

EXAMPLE

```
To load and execute the load module in member JOYCE of data
set PUBS.LOAD, enter:

CALL PUBS(JOYCE)
```

If JOYCE is the only member of data set PUBS.LOAD, you need not specify the member name when issuing the CALL command.

EXAMPLE

```
CALL PUBS
```

If the partitioned data set does not conform to data set naming conventions described in Chapter 2, "Data Sets," then you must specify the member name that contains the program you want to execute. If you specify a fully-qualified data set name, enclose it in single quotation marks in the following manner:

```
CALL 'DOOABC1.MYPROG.LOADMOD(DISCHARG)'
                or
CALL 'SYS1.LINKLIB(IEUASM)'
```

Passing Parameters when Loading and Executing Load Modules

To load and execute the load module in a partitioned data set and pass it parameters, specify the parameters enclosed in single quotation marks following the data set name.

EXAMPLE

```
To pass PRANCE as a parameter to the load module in member
UDOIT of data set DLW.LOAD, enter:

CALL DLW(UDOIT) 'PRANCE'
```

Chapter 15. Submitting and Monitoring a Background Job

Some TSO tasks require a great deal of system resources, or keep the user from doing other TSO tasks until one particular task is completed. To increase productivity and improve system performance, you can submit such tasks to run in the background. Jobs executed in the background are also known as batch jobs. Examples of batch jobs are compiles and link edits.

Batch jobs consist of

- Job control language (JCL) statements
- The user's program
- The data for the program

You must first prepare the JCL and then submit the job. The system executes the job and directs the results to an output queue.

The JCL for a batch job can be user created under EDIT, or system created using the SUBMIT command. User created JCL is discussed in Chapter 17, "Executing Foreground Commands from a Background Job." Generally, the JCL data set should have the descriptive qualifier CNTL.

When the job is submitted, the system gives it a job number, which you can use to identify the job. Using the job number, you can display the status of the job to see what stage of processing it is in. You can request the system to notify you when the job terminates. If you want to stop the processing of your batch job, you can cancel the job.

Use SUBMIT, STATUS, CANCEL and OUTPUT commands primarily to control the submission and processing of jobs in a batch environment.

Submitting Batch Jobs

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

The JOB Statement

Before you submit a batch job with the SUBMIT command, you must know which data set (or a member of a partitioned data set) contains the job or jobs you want to submit. Each job consists of job control language (JCL) statements and program instructions and data.

The first JCL statement in the data set is usually a JOB statement. The job name in the JOB statement can be up to eight characters in length and should consist of your user identification followed by one or more letters or numbers, for example, D58ELM5 or D04DMB74.

If the job name does not begin with your user identification, you can submit the job with the SUBMIT command and request its status with the STATUS command, but you cannot refer to it with the CANCEL or OUTPUT command unless your installation has replaced the IBM-supplied installation exit.

If the job name consists of only your user identification, the system prompts you for one or more characters to complete the job name. This allows you to change job names without re-editing the data. For example, you may submit the same job several times, and supply a different character for the job name each time you are prompted.

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

```
//userid JOB accounting info,
//   userid, ** JOB STATEMENT GENERATED BY SUBMIT **
//   NOTIFY=userid,
//   MSGLEVEL=(1,1)
```

TSO prompts you for a character to complete the job name. The job accounting information is the information specified by the user when logging on to the system.

When you enter the SUBMIT command, you must give the name of a data set (or data sets) containing the batch job (or jobs). You can also use the NONOTIFY operand to specify that you do not want to be notified when the batch job with a generated JOB statement terminates. The SUBMIT command performs best if you enter the fully-qualified data set name in quotation marks. Submitted data sets must have a logical record length of 80 bytes, a record format of fixed-blocked (FB), and must not contain lowercase characters.

SUBMIT Command

Use the SUBMIT command to submit one or more batch jobs for processing. Each job submitted must reside in either a sequential data set, a direct-access data set, or in a member of a partitioned data set, unless you use the asterisk (*) function of the SUBMIT command. The asterisk (*) function tells the system that you will enter the data in some way other than a permanent data set. For more information on the asterisk function, see "The SUBMIT * Function" later in this chapter. Submitted data sets must be fixed blocked, 80 byte records.

Any of these data sets can contain part of a job, one job, or more than one job that can be executed via a single entry of SUBMIT. Each job must comprise an input job stream, that is, JCL plus data. Do not submit data sets with descriptive qualifiers TEXT or PLI if the characters in these data sets are lower case.

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

When you submit more than one job with a single command and TSO finds an error while processing the first job, the second job is not processed. An error that occurs in the second job does not affect the first. Any jobs processed before the error are submitted for execution; jobs that were not processed because of the error should be resubmitted after the error is corrected.

If you wish to provide a JOB statement but you also want to be prompted for a unique job name character, put your user ID in the job name field and follow it with blanks so that there is room for the system to insert the prompted-for character. This process allows you to change job names without re-editing the JCL data set.

Once SUBMIT has successfully submitted a job for conventional batch processing, it issues a 'job name(job ID) submitted' message. The job ID is a unique job identifier assigned by the job entry subsystem.

If SUBMIT is to generate a JOB statement preceding one or more job entry subsystem control statements, make the first statement of your data set a comment. If this is not done, SUBMIT generates the JOB statement following any job entry subsystem control statements.

Data sets that are dynamically allocated by the SUBMIT command are not automatically freed when the command terminates. You must explicitly free dynamically allocated data sets via the FREE command.

A submitted data set need not contain an entire job. A JCL data set and a source data set could be used if both were the proper type of data set.

Using the SUBMIT command, you can specify:

- The data set or data sets that contain the input stream you want to process as a batch job
- The data set or data sets that contain the input data you want to use when processing a batch job
- That you want the option of continuing or cancelling the job after the job stream has been read into the system
- A character string that indicates the end of the job stream
- Whether or not you want the system to hold the job's output for later output processing
- Whether or not you want to predefine characters that the system appends to the job's name
- Whether or not you want the system to prompt you for a password when it generates a job statement
- Whether or not you want to define a user ID that the system uses when it generates a JOB statement
- Whether or not you want the system to let you know when your job has terminated.

For more information on the SUBMIT command, see *TSO/E Command Language Reference.*

Submitting a Batch Job

To submit only one member of a data set as a batch job, specify the name of the data set followed by the name of the member enclosed in parentheses.

EXAMPLE

To submit the job in the member JOB1 in data set JCL.CNTL, enter:

```
SUBMIT JCL(JOB1)
```

To submit a number of jobs at once, specify the list of data set names in parentheses, with each data set name separated from the others by a comma.

EXAMPLE

To submit the jobs in data sets JOB5.CNTL, D00ABC1.SIX.JOB, and member JCL.CNTL(JOB1), enter:

```
SUBMIT (JOB5,'D00ABC1.SIX.JOB',JCL(JOB1))
```

You can specify the data sets a job uses for input when you issue the SUBMIT command. To identify the input data set, specify the input data set's name

following the data set name that contains the job. For this form of the command to be valid, the input data set must have a descriptive qualifier of DATA.

EXAMPLE

```
To run the job in data set JOB5.CNTL, and to use the input data
in data set MYDATA.DATA, enter:

SUBMIT (JOB5,MYDATA)
```

To submit a number of jobs at once, and to specify the data sets from which the jobs will get their input, issue the SUBMIT command and enclose in parentheses:

- The data set that holds the job, followed by
- The name of the data set that contains the input.

EXAMPLE

```
To submit the jobs in data sets JOB5.CNTL and JOB7.CNTL,
where data sets FIVEIN.DATA and SEVNIN.DATA contain input
for each job, enter:

SUBMIT (JOB5,FIVEIN,JOB7,SERVININ)
        |       |    |        |
      job     input  |        |
                     |        |
                   job      input
```

Holding a Batch Job's Output - the HOLD/NOHOLD Operands

To have the system hold a job's output for later processing, use the HOLD operand. NOHOLD, specifying that you do not want the output held for later processing, is a default.

EXAMPLE

```
To have the system retain the output from the job in data
set JOB5.CNTL, enter:

SUBMIT JOB5 HOLD
```

Appending Characters to a Batch Job's Job Name - the JOBCHAR/NOJOBCHAR Operands

To have the system append characters to the job name on each job statement in a data set, use the JOBCHAR operand. This operand is useful if you want to submit the same job several times but each time use a different job name. Use only one character if you plan to use the STATUS command and the job name matches your user ID. Issue the SUBMIT command and specify:

- The name of the data set containing the job(s)
- The JOBCHAR operand with the characters to be appended to each job's name enclosed in parentheses.

EXAMPLE

```
To define the characters XL as the characters that the
system appends to each job's name in data set JOB5.CNTL,
enter:

SUBMIT JOB5 JOBCHAR(XL)
```

The NOJOBCHAR operand, indicating that you want the system to prompt you for job name characters whenever the job name is the user ID, is a default with the SUBMIT command.

Password Prompting When Submitting a Batch Job - the PASSWORD/NOPASSWORD Operand

To request that the system prompt you for a password when generating a job statement, use the PASSWORD operand. This operand is valid only if you have the RACF program product installed at your location, and you request the system to generate a JOB statement for you.

EXAMPLE

```
To request that the system prompt you for a password when
it generates a job statement for the job in data set
JOB5.CNTL, enter:

SUBMIT JOB5 PASSWORD
```

If you do not have the RACF program product installed at your installation, the NOPASSWORD operand, specifying that the system not prompt you for a password, is a default.

Specifying a User ID When Submitting a Batch Job - the USER/NOUSER Operand

To use a specific user ID when the system generates a JOB statement, use the USER operand with the user ID enclosed in parentheses.

EXAMPLE

```
To define user ID D00ABC1 as the user ID the system uses
when it generates a JOB statement for the job in data
set JOB5.CNTL, enter:

SUBMIT JOB5 USER(D00ABC1)
```

This operand is valid only if you have the RACF program product installed at your location, and you request the system to generate a JOB statement for you. The user ID you specify with this operand is also used as the job name for the job you submit. If you have the RACF program product on your system, this operand is the default.

To not define either a PASSWORD or a user ID the system uses when it generates a JOB statement, use the NOUSER password.

EXAMPLE

```
If you do not want to have the system build either a PASSWORD
or user ID when you run the job in data set JOB5.CNTL, enter:

SUBMIT JOB5 NOUSER
```

This operand is the default if you specify neither USER nor NOUSER and if you do not have the RACF program product on your system.

Receiving Notice When a Batch Job is Done - the NOTIFY/NONOTIFY Operand

The system notifies you when a batch job has ended, as the NOTIFY operand is a default with the SUBMIT command. The system ignores the NOTIFY operand if the data set you submit already contains a JOB statement.

To prevent the system from notifying you when a batch job has ended, use the NONOTIFY operand.

EXAMPLE

```
If you do not want the system to notify you when the job in
member JOB1 in data set JCL.CNTL has terminated, enter:

SUBMIT JCL.CNTL(JOB1) NONOTIFY
```

The SUBMIT * Function

The SUBMIT command supports an asterisk (*) for the positional operand value and two keyword operands, END and PAUSE. The keyword operands, END and PAUSE, are valid only when (*) is specified and when the issuer is not in EDIT mode.

SUBMIT * allows the job stream source to reside in other than a permanent data set, such as a terminal, in-storage lists, temporary data sets, and the CLIST-type in-storage lists. The job stream may be entered directly without creating and editing a data set. Figure 15-1 illustrates how the SUBMIT * function is used to submit background jobs. Note that the existing SUBMIT * function of EDIT continues to select the current data set as the input job stream. Therefore, this SUBMIT * function is not available in EDIT mode.

```
/*Example 1: Submitting a job with input from a terminal*/
READY
submit *
ENTER INPUT JOB STREAM:
//step       exec  pgm=somepgm
( null line )
ENTER JOBNAME CHARACTER(s)
a
JOB USERIDA (JOB00007) SUBMITTED
READY
/*Example 2: Submitting jobs with input from a CLIST*/
/* The following is a listing of the CLIST about to be
   submitted*/

PROC 0 STEP(2)
CONTROL PROMPT
SUBMIT * PAUSE END(GO)
//USERIDA     JOB   MSGLEVEL=1
//STEPA1      EXEC  PGM=YOURPGM
//SYSPRINT    DD    SYSOUT=A
IF &STEP=2 THEN DO
//STEPA2      EXEC  PGM=PROGRAM2
//SYSPRINT    DD    SYSOUT=A
END
ELSE DO
//STEPA3      EXEC  PGM=PROGRAM3
//SYSPRINT    DD    SYSOUT=A
END
//USERIDB     JOB   MSGLEVEL=1
//STEPB       EXEC  PGM=SOMEPGM
//SYSPRINT    DD    SYSOUT=A
GO

/* The following shows the CLIST being executed.

exec myclist list
ENTER INPUT JOB STREAM:
SUBMIT * PAUSE END(GO)
//USERIDA     JOB   MSGLEVEL=1
//STEPA1      EXEC  PGM=YOURPGM
//SYSPRINT    DD    SYSOUT=A
//STEPA2      EXEC  PGM=PROGRAM2
//SYSPRINT    DD    SYSOUT=A
//USERIDB     JOB   MSGLEVEL=1
//STEPB       EXEC  PGM=SOMEPGM
//SYSPRINT    DD    SYSOUT=A
go
SHOULD INPUT JOB STREAM BE SUBMITTED? ENTER YES OR NO: +
yes
JOB USERIDA(JOB00008) SUBMITTED
JOB USERIDB(JOB00009) SUBMITTED
READY
```

Figure 15-1. The SUBMIT * Function

Ending a Batch Job

To request the option of continuing or cancelling a job after it has been read into the system, use the PAUSE operand.

EXAMPLE

```
To request the option of continuing or cancelling a
submitted job, enter:

SUBMIT * PAUSE
```

The PAUSE operand supports the SUBMIT * function **only**. If you do not specify this operand, the system processes the job stream when it detects the end of the job stream.

To specify a character that indicates the end of a job stream, use the END operand with the character string enclosed in quotation marks.

EXAMPLE

```
If you want the system to end the job when it encounters the
character string #$, enter:

SUBMIT * END(#$)
```

The END operand supports the SUBMIT * function **only**. You can only specify alphabetic, numeric, or national characters when using this operand. If you don't specify this operand, a null or blank line indicates the end of the job stream. Specifying this operand allows you to enter blank lines as part of your job stream. The END characters must begin in column 1 and be the only data on the line.

Displaying the Status of Jobs

Any time after you submit a background job you can use the STATUS command to display its status. The display tells you whether the job is awaiting execution, is currently executing, or has executed but is still on the output queue. The display also indicates whether a job is in hold status.

STATUS Command

Use the STATUS command to display the status of batch jobs at your terminal. You can obtain the status of all your batch jobs, of several specific batch jobs, or of a single batch job.

Using the STATUS command, you can specify which of your job's or jobs' status you want to display.

You can use this command only if the installation management has given you the authority to do so.

For more information on the STATUS command, see *TSO/E Command Language Reference.*

Displaying the Status of All Your Jobs

To display all the jobs whose job names consist of your user ID plus one character, issue the STATUS command with no operands.

EXAMPLE

```
If you submitted ten jobs and you wanted to know the status of
each, enter:

STATUS
```

Displaying the Status of Specific Jobs

To display all the jobs associated with a specific job name, specify the job name following the STATUS command.

EXAMPLE

```
To display all the jobs associated with job name D00ABC1A,
enter:

STATUS D00ABC1A
```

To display all the jobs associated with a number of job names, specify the list of job names, separated by commas, and enclosed in parentheses.

EXAMPLE

To display the status of all the jobs whose job names are D00ABC1A, D00ABC1B, and D00ABC1C, enter:

```
STATUS (D00ABC1A,D00ABC1B,D00ABC1C)
```

To display a specific job associated with a job name, enter the job name with the system-supplied job identifier enclosed in parentheses.

EXAMPLE

To display the status of the job identified by number JOB01098 with job name D00ABC1A, enter:

```
STATUS D00ABC1A(JOB01098)
```

Cancelling a Batch Job's Execution

Use the CANCEL command to halt a batch job's execution and its output.

CANCEL Command

Use the CANCEL command to halt processing of batch jobs that you submitted from your terminal. If you cancel the job successfully, the system displays a READY message at your terminal. The system also notifies the system operator when you cancel a job.

Using the CANCEL command, you can specify:

- Which job or jobs you want to cancel
- Whether or not you want to purge a job's output.

You can use this command only if the installation management has given you the authority to do so.

For more information on the CANCEL command, see *TSO/E Command Language Reference.*

Cancelling Specific Jobs

To cancel the job associated with a specific job name, specify the job name following the CANCEL command.

EXAMPLE

```
To cancel the job associated with job name D00ABC1A, enter:

CANCEL D00ABC1A
```

If there is more than one job associated with job name D00ABC1A, the system prompts you for a specific job identifier.

To cancel each job associated with a number of job names, specify the list of job names enclosed in parentheses and each separated from the others by a comma or blank.

EXAMPLE

```
To cancel the jobs whose names are D00ABC1A, D00ABC1B, and
D00ABC1C, enter:

CANCEL (D00ABC1A,D00ABC1B,D00ABC1C)
```

To cancel a specific job associated with a job name, specify the job name with the system-supplied job identifier enclosed in parentheses.

EXAMPLE

To cancel the job identified by number JOB01098 with job name D00ABC1A, enter:

```
CANCEL D00ABC1A(JOB01098)
```

Cancelling Jobs and Purging Their Output - the PURGE/NOPURGE Operand

When you cancel a job, its output is not purged from the system. The NOPURGE operand is a default with the CANCEL command. You can purge a job's output from the system when you cancel that job's execution by specifying the PURGE operand. As with cancelling a job, you can purge the output for all jobs associated with your user ID. To purge a job's output, use the PURGE operand following the job name operand.

EXAMPLE

To cancel the job with a job name of D00ABC1A and purge its associated output, enter:

```
CANCEL D00ABC1A PURGE
```

Chapter 16. Directing the Output of a Background Job

Use the OUTPUT command to display or direct any output that is held by the system. Held output is output that the system does not immediately place on a print queue. Held output can consist of the job's job control language (JCL) statements, system messages, or system output data sets.

You can simplify the use of the OUTPUT command by including the NOTIFY operand either on the JOB statement or on the SUBMIT command when you submit a job for batch processing. The system will notify you when the job terminates, giving you an opportunity to use the OUTPUT command. MSGCLASS and SYSOUT data sets should be assigned to reserved classes or explicitly held in order to be available at the terminal.

Once you enter the OUTPUT command and the operands you want, the system places your terminal session in OUTPUT mode. At this point, you can enter any of the four subcommands associated with the OUTPUT command to modify a job's held output. The four subcommands are CONTINUE, END, HELP, and SAVE. For more information on the OUTPUT command's subcommands, see "OUTPUT Subcommands" later in this chapter.

Note that you can also use the OUTPUT command and its subcommands to modify foreground-created output.

OUTPUT Command

Using the OUTPUT command, you can:

- Direct the output from a job to your terminal. The output includes the job's job control language (JCL), system messages (MSGCLASS), and system output (SYSOUT) data sets.
- Direct the output from a job to a specific data set.
- Delete the output for jobs.
- Change the output class(es) for a job.
- Route the output for a job to a remote work station.
- Release the output for a job for printing.

The OUTPUT command applies to all jobs whose job names begin with your user identification. Access to jobs whose job names do not begin with a valid user identification must be provided by an installation-written exit routine.

Displaying Held Output for Specific Jobs

You can display held output for:

- A specific job identified by job name
- A number of jobs
- A specific job identified by a system-supplied job identifier
- A job with a particular output class.

To display the held output from the job associated with a specific job name, specify the job name following the OUTPUT command.

EXAMPLE

```
To display the output from the job with job name D00ABC1A,
enter:

OUTPUT D00ABC1A
```

If there is more than one job associated with a job name, the system prompts you for a specific job identifier.

To display the held output for each job associated with a number of job names, enclose in parentheses the list of job names separated by a comma or blank.

EXAMPLE

```
To display the output for the jobs whose job names are
D00ABC1A, D00ABC1B, D00ABC1C, enter:

OUTPUT (D00ABC1A,D00ABC1B,D00ABC1C)
```

To display the held output for a specific job associated with a job name, specify the job name with the system-supplied job identifier enclosed in parentheses.

EXAMPLE

```
To display the job identified by number JOB02249 with job
name D00ABC1A, enter:

OUTPUT D00ABC1A(JOB02249)
```

To display the held output in a particular output class for the jobs associated with a specific job name, issue the OUTPUT command and specify:

- The job name.
- The CLASS operand with the job class or classes to be searched enclosed in parentheses. A class name is a single letter or digit (A-Z or 0-9). If you do not specify the name of a class, all held output for the jobs are available.

EXAMPLE

```
To display the held output in output class A, B, or S
associated with job name D00ABC1P, enter:

OUTPUT D00ABC1P CLASS(A,B,S)
```

Redirecting Held Output for Specific Jobs - the PRINT Operand

You can redirect output for a specific job to:

- A terminal
- A data set

To redirect the held output from the jobs associated with a specific job name to your terminal, issue the OUTPUT command and specify:

- The job name
- The PRINT operand with an asterisk (*) enclosed in parentheses.

If you do not specify either a data set name or an asterisk, an asterisk is the default. PRINT is the default value if you omit the PRINT, DELETE, NEWCLASS, DEST, HOLD and NOHOLD operands, discussed later in this chapter.

EXAMPLE

```
To redirect the held output for the jobs associated with job
name D00ABC1P to your terminal, enter:

OUTPUT D00ABC1P PRINT(*)
```

To redirect the held output from the jobs associated with a specific job name to a data set, issue the OUTPUT command and specify:

- The job name.
- The PRINT operand.
- Immediately enclose in parentheses the second level, user-supplied name of the data set to which the output is to go.

EXAMPLE

```
To redirect the held output for the jobs associated with
job name D00ABC1P to data set D00ABC1.HLDPRINT.OUTLIST,
enter:

OUTPUT D00ABC1P PRINT(HLDPRINT)
```

In the previous example, the system appends your user ID (D00ABC1) as the first-level qualifier. It uses the name you specify as the user-specified qualifier, and OUTLIST as the default descriptive qualifier.

Directing Held Output Based on Checkpointing - the BEGIN, HERE, and NEXT Operands

A checkpoint is a point at which information about the status of a job and the system can be recorded so that the job step can be restarted later. A data set is checkpointed if it was interrupted during printing and never processed to the end of the data during a terminal session. Interruptions which cause a data set to be checkpointed occur when:

- Processing terminates in the middle of printing a data set because of an error or ABEND condition.
- The attention key is pressed during the printing of a data set and the CONTINUE NEXT subcommand is entered. The KEEP operand must be present or the data set is deleted.
- The attention key is pressed during the printing of a data set and the END subcommand is entered.

You can start output processing of held output data sets at the following points:

- At the beginning of each data set, regardless of checkpoints.
- At the beginning of each data set, if the data set has not been checkpointed.
- At the approximate point of interruption in a checkpointed data set.
- After skipping a checkpointed data set.

For information on checkpointing a data set, see *TSO/E Command Language Reference* or *MVS JCL*.

To start output processing for the held output data sets associated with a job name at the beginning of each data set, regardless of whether the data set has been checkpointed or not, use the BEGIN operand.

EXAMPLE

```
To ensure that the system starts its output processing at
the beginning of each data set that contains the held
output for each job associated with job name D00ABC1P,
enter:

OUTPUT D00ABC1P BEGIN
```

To start output processing for the held output data sets associated with a job name at the beginning of each data set, as long as the data set has not been checkpointed, use the HERE operand. You can also use the HERE operand to start output processing for the held data sets associated with a job name at the approximate point of interruption, as long as the data set has been checkpointed.

EXAMPLE

```
To ensure that the system starts its output processing at the
beginning of each data set that is not checkpointed, and that
contains the held output for each job associated with job
name D00ABC1P, enter:

OUTPUT D00ABC1P HERE
```

To skip output processing for the held output data sets associated with a job name that are checkpointed, use the NEXT operand. Output processing resumes at the beginning of the next uncheckpointed data set. Note that the system deletes the checkpointed data sets it skips, unless you specify the KEEP operand.

EXAMPLE

```
To request that the system skip output processing for each
checkpointed data set that contains the held output for
each job associated with job name D00ABC1P, enter:

OUTPUT D00ABC1P NEXT
```

Pausing to Process Held Output - the PAUSE/NOPAUSE Operand

You can have the system pause after it lists a SYSOUT data set. Do this to allow yourself the opportunity to enter a SAVE or CONTINUE subcommand. To have the system pause after it lists a SYSOUT data set, use the PAUSE operand. The NOPAUSE operand, a default with the OUTPUT command, requests that the system does not pause after listing each SYSOUT data set.

EXAMPLE

```
To have the system pause after it lists each SYSOUT
data set associated with the jobs with job name
D00ABC1P, enter:

OUTPUT D00ABC1P PAUSE
```

Pressing the ENTER key causes normal processing to continue. You can override the PAUSE operand by using the CONTINUE subcommand.

Specifying a Disposition for Held Output - the KEEP/NOKEEP, HOLD/NOHOLD, and DELETE Operands

To request that the system keep the held output data sets associated with a specific job name on the output queue after printing them, use the KEEP operand following the job name. After a job is printed, the system releases and deletes the held output data sets. The NOKEEP and NOHOLD operands specifying the releasing and deleting of held output are defaults.

EXAMPLE

```
To keep the held output data sets associated with the jobs
with job name D00ABC1P on the output queue after printing
them, enter:

OUTPUT D00ABC1P KEEP
```

When you submit a job to the system and the message class or output class is not a held class, the output data sets are not held by the system. To request that the system hold the output data sets associated with a specific job name so you can display them later, use the HOLD operand following the job name operand.

EXAMPLE

```
To hold the output data sets associated with the jobs with
job name D00ABC1P, enter:

OUTPUT D00ABC1P HOLD
```

To request that the system delete the output data sets associated with a specific job name, use the DELETE operand following the job name operand.

EXAMPLE

```
To delete the output data sets associated with the jobs
with job name D00ABC1P, enter:

OUTPUT D00ABC1P DELETE
```

Specifying a New Output Class for Held Output - the NEWCLASS Operand

To change the output class for the held output for the jobs associated with a particular job name, use the NEWCLASS operand with the new output class enclosed in parentheses.

EXAMPLE

```
To change the output class to R for the held output for
the jobs associated with job name D00ABC1P, enter:

OUTPUT D00ABC1P NEWCLASS(R)
```

Routing the Held Output to a Remote Work Station - the DEST Operand

To route the output for the jobs associated with a particular job name to a remote work station, use the DEST parameter with the remote work station ID enclosed in parentheses.

EXAMPLE

```
To route the output from the jobs with job name D00ABC1P
to remote work station PAPUA#NG, enter:

OUTPUT D00ABC1P DEST(PAPUA#NG)
```

OUTPUT Subcommands

There are four subcommands associated with the OUTPUT commands. They are CONTINUE, END, HELP, and SAVE. Use these subcommands to interactively display or direct output data sets. Use OUTPUT subcommands as follows:

- The CONTINUE subcommand to resume interrupted output operations
- The END subcommand to terminate OUTPUT
- The HELP subcommand to obtain further information about the OUTPUT command and its associated subcommands
- The SAVE subcommand to copy a SYSOUT data set to another data set for retrieval by a different method.

CONTINUE Subcommand

Use the CONTINUE subcommand to resume interrupted output operations. Interruptions causing subcommand mode occur when:

- Processing of a SYSOUT data set completes and the PAUSE operand was specified with the OUTPUT command
- You press the attention key.

Pressing the attention key purges the input/output buffers for the terminal. Data and system messages in the buffers at this time may be lost. Although the OUTPUT command attempts to back up records to recover the lost information, results are unpredictable due to record length and buffer size. You may see records repeated or you may notice records missing if you attempt to resume processing of a data set at the point of interruption using the HERE operand with the CONTINUE subcommand, or in the next session using HERE on the OUTPUT command.

Using the CONTINUE subcommand, you can specify:

- Where in the output data set you want the system to begin processing
- Whether or not you want to skip processing for a checkpointed output data set
- Whether or not you want the system to pause after each SYSOUT data set is listed.

Specifying Where to Continue Output Processing - the BEGIN, HERE, and NEXT Operands

To have the system start output processing for the output data set at the beginning of the data set, regardless of whether it has been checkpointed or not, use the BEGIN operand.

EXAMPLE

```
To ensure that the system starts processing the output data
set you are currently displaying at the beginning, enter:

CONTINUE BEGIN
```

For information on checkpointing a data set, see *TSO/E Command Language Reference* and "Directing Held Output Based on Checkpointing" earlier in this chapter.

To have the system start output processing for the output data set at the beginning of the data set, as long as the data set has not been checkpointed, use the HERE operand. You can also use the HERE operand to have the system start output processing for the data set at the approximate point of interruption, as long as the data set is checkpointed.

EXAMPLE

```
As long as the data set you are displaying is not
checkpointed, to ensure that the system starts processing
the output data set at the beginning, enter:

CONTINUE HERE
```

To have the system skip output processing for a checkpointed output data set, use the NEXT operand. Output processing resumes at the beginning of the next uncheckpointed data set. Note that unless you specify the KEEP operand, the checkpointed data sets the system skips are deleted from the system.

EXAMPLE

```
As long as the current data set is checkpointed, to ensure
that the system skip output processing for this data set,
enter:

CONTINUE NEXT
```

Pausing During Output Processing for a Data Set - the PAUSE/NOPAUSE Operand

To have the system pause after it lists each SYSOUT data set, use the PAUSE operand.

EXAMPLE

```
CONTINUE PAUSE
```

Pressing the ENTER key after the pause causes normal processing to continue. You can use this operand to override a NOPAUSE specification on the OUTPUT command.

NOPAUSE, a default on the CONTINUE subcommand, specifies that you do not want the system to pause. You can use this operand to override a PAUSE specification on the OUTPUT command.

END Subcommand

Use the END subcommand to terminate the operation of the OUTPUT command.

Terminating the OUTPUT Command

To terminate the operation of the OUTPUT command, issue the END subcommand, as follows:

```
END
```

HELP Subcommand

Use the HELP subcommand to obtain the syntax and function of the OUTPUT subcommands. Refer to the HELP command for a description of the syntax and function of the HELP subcommand.

Obtaining Information About OUTPUT Subcommands

To obtain syntax and function information about all the subcommands of the OUTPUT command, issue the HELP command, as follows.

```
HELP
```

To obtain syntax and function information about a specific subcommand of the OUTPUT command, use the HELP subcommand, followed by the name of the subcommand in question.

EXAMPLE

```
To obtain information about the CONTINUE subcommand, enter:

HELP CONTINUE
```

SAVE Subcommand

Use the SAVE subcommand to copy the SYSOUT data set from the spool data set to the named data set. The named data set can be any data set that would be valid if used with the PRINT operand. There is no restriction against saving JCL. Using the SAVE command you can specify the name of the data set into which you want to save the output currently being processed.

To use SAVE, you must specify the PAUSE operand on the OUTPUT command. SAVE does not save the entire SYSOUT output of the job; it saves only the data set currently being processed.

To have the system save the output currently being processed into a data set, use the SAVE subcommand followed by the user-specified qualifier of the destination data set's name.

EXAMPLE

```
To save the data from the output data set you are currently
processing into data set OUTSAVE.OUTLIST, enter:

SAVE OUTSAVE
```

Chapter 17. Executing Foreground Commands from a Background Job

There are times when it is not practical to execute a series of commands or a CLIST from your terminal. If a job is going to take an extended period of time to execute or if a large amount of output is to be printed, it is more convenient to execute in the background, that is, independent of the terminal.

TSO allows the user to execute a CLIST or a series of commands in the background. To use this function, you *must be* authorized by your installation to use the SUBMIT command or to process JCL.

Note: There are a number of restrictions that apply to using commands in the background. See Figure 17-9, "Processing Considerations (RACF and non-RACF Systems)" for a complete list of the restrictions.

Concurrent Execution

When executing commands in the background and foreground concurrently, you should be aware that allocation of a data set both in the foreground and the background might not be successful. If a data set has been allocated (foreground or background mode) with a disposition of OLD, MOD, or NEW, it cannot be allocated in the opposite mode by the ALLOCATE command or any other command processor. For example, the ALLOCATE command issued in the background for a data set in use in the foreground, will issue an error message that the data set is already in use. Any command remaining in the job stream is processed.

If a user's LOGON procedure allocates a data set with a disposition of NEW, MOD, or OLD, a background job specifying the same LOGON procedure will not execute until the user logs off or frees the data set in the foreground. Similarly, if a background job is executing and the user attempts to log on a terminal with the same LOGON procedure, the logon attempt fails.

Output Handling

Output produced by a background job differs from output in the foreground in the following ways:

- Messages producing multiple levels are printed in their entirety. (It is the same as entering a (?) and recovering all levels.)

- The allocation of an output data set to the terminal causes that output to be printed after all other output.

If it is necessary to see the output at your terminal rather than waiting to have it printed on the system printer, one of two things can be done; place the output in a data set or have the output held and use the OUTPUT command to look at it.

For the appropriate JCL, see "Writing JCL for Command Execution" in this book.

Submitting Commands Using the SUBMIT Command

The SUBMIT command is used to submit batch jobs. Refer to the "SUBMIT Command" in Chapter 15, "Submitting and Monitoring a Background Job" for more information about the SUBMIT command. There are two techniques for submitting batch jobs using the SUBMIT command.

1. Use the SUBMIT command or subcommand of EDIT. When submitting a batch job in a data set, the data set must be in the same format as a CNTL-type data set. Using the EDIT command to create the data set assures you of the correct data set format. Figure 17-1 illustrates how the EDIT command can be used to create a data set containing commands and how the SUBMIT command is used to submit this same data set as a background TSO session.

2. The SUBMIT command supports an asterisk (*) for the positional operand value and two keyword operands, END and PAUSE. The keyword operands, END and PAUSE, are valid only when (*) is specified and when the issuer is not in EDIT mode.

```
edit examp2.cntl new
INPUT
00010 logon
00020 profile prefix(user2)
00030 edit a.data new emode
00040 5 this is first line
00050 10 this is second line
00060 save b.data reuse
00070 end save
00080 ( null line )
EDIT
submit * jobc(a)
JOB USER2A(JOB000347) SUBMITTED
EDIT
end save
READY
```

Figure 17-1. Creating and Submitting Data Sets Containing Commands

If you want the SUBMIT command to generate the JCL statements to execute commands in the background, specify LOGON as the first command in the data set. No operands are required on the LOGON command. However, you must specify your user ID if you specified any other operands on the LOGON

command. If you want to charge the job to an account number other than the one you are currently using, you must specify the ACCT operand. MAIL, NOMAIL, NONOTICE and RECONNECT operands are ignored when you specify them on a LOGON command executed in the background. Figure 17-2 illustrates how the system integrates your data with the JCL generated by the system from the example in Figure 17-1

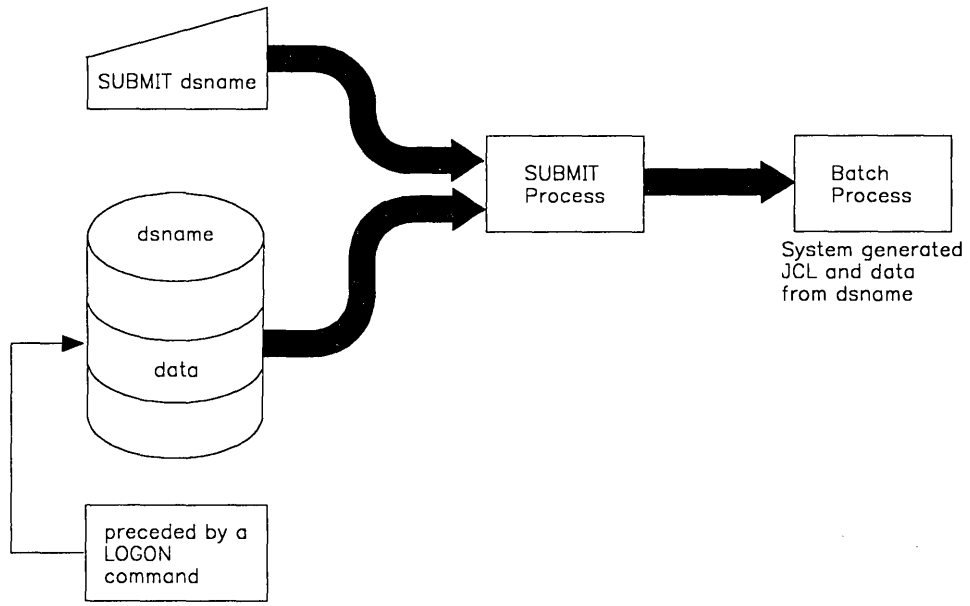


Figure 17-2. The SUBMIT Process Using System-Generated JCL

Other conditions that apply when submitting jobs in the background are:

- You can precede the LOGON command with JES2 or JES3 JCL statements. If this is done, however, TSO does not process any other JCL statements in the job stream, and the system does not generate any JCL statements. If a JCL statement follows the LOGON command, it is executed in the background as an input line.

- You must write the LOGON command on one line in columns 1 thru 72 only.

- You can have more than one LOGON command within a single data set. SUBMIT creates the JCL statements for each of them and executes them accordingly.

- You cannot use the characters */ in columns one and two in a data set where SUBMIT-generated JCL is to be used. SUBMIT designates these characters as a delimiter. If you must use these characters in columns one and two, you have to provide your own JCL.

- Use only the PROC operand on a submitted LOGON command if the procedure is available in the SYS1.PROCLIB.

- The PROC specified on a submitted LOGON command must not contain the ddnames SYSTSIN or SYSTSPRT. SUBMIT will always generate these ddnames.

For example, if you submit a data set containing the commands LOGON and LISTCAT, SUBMIT enters the following background job:

```
//YOURID     JOB    ACCT.INFO.,
//                  YOURID
//                  NOTIFY=YOURID
//                  MSGLEVEL=(1,1)
//*************************************************************
//*  THE FOLLOWING LOGON COMMAND WAS FOUND IN SUBMIT'S        *
//*  INPUT DATA SET(S) AND WAS USED TO GENERATE THE JCL       *
//*  TO EXECUTE TSO COMMANDS IN THE BACKGROUND:               *
//*                                                           *
//*  LOGON                                                    *
//*                                                           *
//*************************************************************
//CBSTEP     EXEC PGM=IKJEFT01,DYNAMNBR=30
//SYSTSPRT   DD   SYSOUT=A  **OUTPUT FROM COMMANDS IN
                            BACKGROUND**
//SYSTSIN    DD   DATA,DLM='*/'  **INPUT COMMANDS**
  LISTCAT
*/
```

If the above SUBMIT-generated JCL statements are not sufficient, you can insert your own JCL. Do not include a LOGON command if this is done. Figure 17-3 illustrates the SUBMIT process if you create your own JCL.

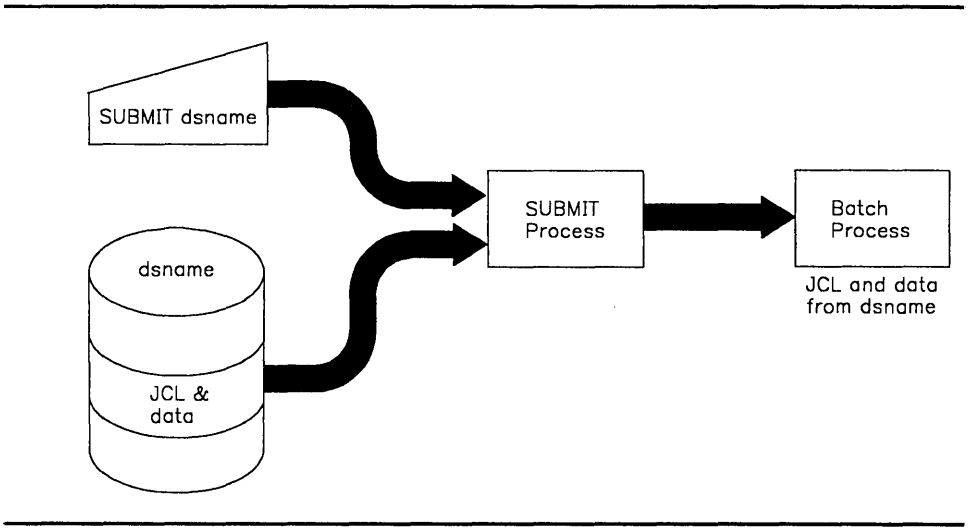


Figure 17-3. The SUBMIT Process With User-Created JCL Statements

Submitting Commands Using JCL

When submitting a background job that contains foreground commands, you must use at least one of each type of the JCL cards shown in the following figure.

Figure 17-4 illustrates the JCL statements required for input and the appropriate placement of the data statements.

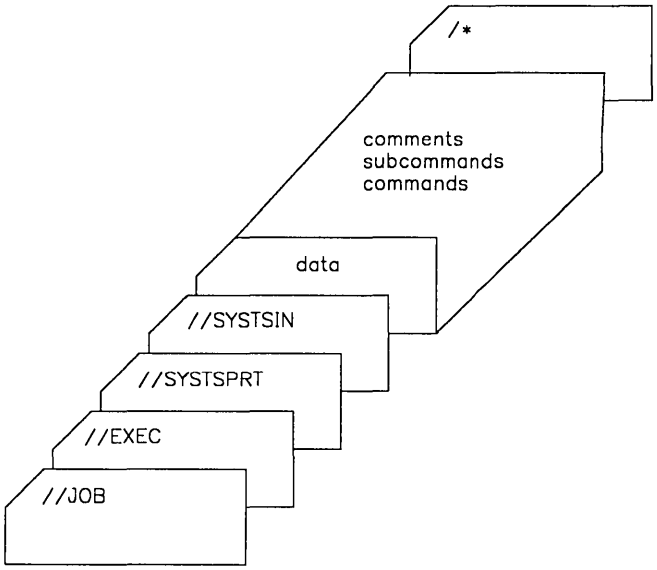


Figure 17-4. JCL Setup for Processing Commands in the Background

Writing JCL for Command Execution

The following JCL statements are required for executing commands in the background. Other statements can be used, but are not required. (See *MVS JCL* for a complete description of JCL statements.)

JOB Statement

The JOB statement is the first JCL statement in a batch job. It marks the beginning of a job and, when jobs are stacked in the input stream, marks the end of the control statements for the preceding job. This statement consists of four fields. The format is:

```
//jobname JOB operands comments
```

jobname
is required and is used to identify the job to the system.

JOB
the JCL term that identifies the type of statement.

operands
specifies the specific processing options the system uses when processing this job.

comments
optional user-supplied information pertaining to the function of this statement.

EXEC Statement

The program needed to execute TSO commands from the background is the terminal monitor program, known as TMP, or by its program name, IKJEFT01. The EXEC (execute) statement is used to execute program IKJEFT01 (TSO terminal monitor program (TMP)). The format is:

```
//stepname EXEC PGM=IKJEFT01,DYNAMNBR=nn,PARM='command'
```

stepname
is optional and can be used as a step identifier in the programs consisting of more than one step.

EXEC
the JCL term that identifies this type of statement.

PGM=
specifies the module being executed.

DYNAMNBR=
the number of allocations of data sets, ddnames, or attribute lists that can be used at one time per job step. This number can be from 1 to 1635.

PARM=
is optional and can be used to supply the first (or only) command to be executed. This is used most often when you only execute one command in the step.

SYSTSPRT DD Statement

The SYSTSPRT DD statement is used to control the output from your background job. By specifying different operands on this statement, you can have the output listed on a system printer, placed in a specified data set for later use, or held in a work data set so you can look at it using the OUTPUT command.

If you want to see your output as soon as the job has executed, hold the output and specify the NOTIFY=user ID operand on the JOB statement. NOTIFY causes a message to be printed at your terminal when the job completes. For

example, assume that you specified NOTIFY=your user ID on the JOB statement. To hold the output, enter:

```
//SYSTSPRT  DD  SYSOUT=H
```

where H is a held class and MSGCLASS=H is also specified on the JOB statement.

You can also hold output by specifying:

```
//SYSTSPRT  DD  SYSOUT=N,HOLD=YES
```

where N is not a held class.

You might not always want to hold a job's output. To not hold a job's output, specify the SYSOUT operand with the non-held output class value. For example,

```
//SYSTSPRT  DD  SYSOUT=N
```

where N is the installation-defined class for output that is not held.

You can write a job's output to a specified data set. To write a job's output to new data set D00ABC1.JOBOUT.DATA, where your user ID is assumed to be D00ABC1, enter the following:

```
//SYSTSPRT  DD  DSN=D00ABC1.JOBOUT.DATA,DISP=NEW,SPACE=(TRK,5,5),
          UNIT=SYSDA
```

where D00ABC1.JOBOUT.DATA is a new data set allocated with 5 tracks of primary space and secondary extents of 5 tracks each.

To write a job's output to old data set D00ABC1.OLDOUT.DATA, where your user ID is assumed to be D00ABC1, specify the following:

```
//SYSTSPRT  DD  DSN=D00ABC1.OLDOUT.DATA,DISP=OLD
```

where the data set already existed before the step executes.

SYSTSIN DD Statement

The SYSTSIN DD statement is used to specify that the data to follow consists of executable commands and/or subcommands. For example, to indicate to the system that all data following this statement is to be used as input, until the system encounters an input delimiter, such as the characters /* or the DLM operand, specify:

```
//SYSTSIN  DD  *
```

If any of the input statements start with the characters //, use the DD DATA statement instead.

To indicate to the system that all data following this statement is to be used as input, including statements that start with the characters //, until an input delimiter (/* or DLM) is found, specify:

```
//SYSTSIN  DD  DATA
```

To indicate to the system that all the input data can be found in data set D00ABC1.INPUT.DATA, specify:

```
//SYSTSIN  DD  DSNAME=D00ABC1.INPUT.DATA
```

The SYSTSIN and SYSTSPRT DD statements can refer to a sequential data set or a member of a partitioned data set.

Command Processing Differences in the Background

When executed in the background, some TSO commands process differently from when they are executed in the foreground. The commands listed here show only the differences between how they process in the background as opposed to how they process in the foreground. The command syntax is shown only where operands have been added. For a complete description of each of these commands, see *TSO/E Command Language Reference*. Restrictions which apply to commands processed in the background are listed in Figure 17-9.

CALL Command

Service aids, utilities, and other programs obtaining their input from an allocated file such as SYSIN, must have the input in a created data set or a job stream data set that is created with a DD * or a DD DATA statement. Once the data set has been created and allocated, the CALL command can be used to execute the program that accesses the SYSIN data. Figure 17-5 illustrates the allocation and creation of input data sets:

```
//examp1   exec    pgm=ikjeft01,dynamnbr=20
//systsprt dd      sysout=a
//systsin  dd      *
  profile prefix(user1)
  allocate file (sysprint) dataset(*)
  allocate file(sysin) altfile(inputdd)
  call prog1
  allocate file(sysin) altfile(inputdd2) reuse
  call prog2
  free all
//inputdd  dd     *
  **input to prog1**
//inputdd2 dd     *
  **input to prog2**
/*
```

Figure 17-5. Allocating and Creating Input Data Sets

In the preceding example, the second allocate statement allocates the SYSIN data for PROG1, and specifies that the data is defined by the DD statement named INPUTDD. The third allocate statement allocates the SYSIN data for PROG2, and specifies that the data is defined by the DD statement named INPUTDD2. DD statements INPUTDD and INPUTDD2 use in-stream data as input.

Note: Allocating the input file to a terminal results in an I/O error message. Termination occurs when the program tries to get input from the terminal.

EDIT Command

Although EDIT performs the same in the foreground and background, you must be careful when inserting blank lines into a data set because EDIT interprets a blank line as a null line. This causes the system to switch from input mode to edit mode. To insert blank lines into a data set, you can:

- Insert a character string that you do not use anywhere else in the data set. Then, before ending the EDIT session, issue a global change, changing every occurrence of the character string to blanks.
- Specify the EMODE operand on the EDIT command. Each new line is preceded by a line number, wherever line numbering is allowed.

Figure 17-6 illustrates how to insert blank lines using the unused character string method.

```
edit   examp4.cntl  new
INPUT
00010 logon user4 proc(proca)
00020 profile prefix(userid)
00030 edit p data new
00040 line one
00050 @@@@@
00060 line two
00070 @@@@@
00080 line three
00090 @@@@@
00100 line four
00110 ( null line )
00120 c 10 999 /@@@@@// all
00130 list
00140 end save
00150 ( null line )
end save
READY
submit examp4.cntl notify jobchar(a)
JOB USER4A(JOB00001) SUBMITTED
READY
```

Figure 17-6. Entering Blank Lines Into Your Data Set

LOGOFF Command

When the LOGOFF command is executed in the background, your TSO session is terminated normally. Any remaining commands in the input stream are ignored.

PROFILE Command

The following differences should be noted for foreground/background processing:

- Changes made while processing in the foreground are saved from session to session.
- Changes made while processing in the background are not saved.

You can use the RECEIVE command in a background job to receive a message or data set. To use RECEIVE in the background, you must specify the RECEIVE command and the responses to any prompts that you would anticipate when using RECEIVE in the foreground. Supply the responses in the same order as they would appear in the foreground. The commands are executed as if PROFILE NOPROMPT were entered.

The data set or message that you receive will be prefixed by the userid that you supply on the JOB statement. To use an alternate prefix as the data set name, include the following command at the beginning of the SYSTSIN data stream:

```
PROFILE PREFIX(prefix)
```

Be sure that any alternate prefix you use is a valid catalog alias.

Messages are written to the data set defined in the SYSTSPRT DD statement.

Figure 17-7 shows a batch job for receiving a data set in the background. Figure 17-8 shows a batch job for receiving a message in the background.

```
//JOBNAME  JOB     USER=USERID,PASSWORD=PASSWD
//STEP1    EXEC    PGM=IKJEFT01,REGION=512K
//SYSTSPRT DD      SYSOUT=A
//SYSTSIN  DD      *
  RECEIVE
  RESTORE
  **responses to anticipated prompts**
  r
/*
```

Figure 17-7. Receiving a Data Set in the Background

```
//JOBNAME  JOB     USER=USERID,PASSWORD=PASSWD
//STEP1    EXEC    PGM=IKJEFT01,REGION=512K
//SYSTSPRT DD      SYSOUT=A
//SYSTSIN  DD      *
  RECEIVE
/*
```

Figure 17-8. Receiving a Message in the Background

Error Condition Handling

The return code from the job step that executes commands in the background is that of the last command executed.

Specific ABEND codes are documented in *Message Library: System Codes.* ABENDs occurring in the background will be handled in the following manner:

XXX ABEND code '*xxx*' will occur if a command or the TMP has abended. A dump will be taken if a SYSUDUMP or SYSABEND has been allocated, either in the JCL or with the ALLOCATE command, and the remainder of the commands in SYSTSIN will be ignored.

These restrictions always apply to the execution of commands processed in the background:

- The OPERATOR and TERMINAL commands are not supported. You get an error if you try to issue them.
- You cannot specify USER(*) on the SEND command in the background. A user ID is required. USER(*) causes an error message to be issued.
- Specifying RECOVER with the EDIT command has no affect. The EDIT CLIST workfile recovery function is not supported.
- The CLIST statements READ and TERMIN are not supported.
- The symbolic variable &SYSPROC has a null value.

The following restrictions apply only to the execution of commands processed in the background when the RACF program product is not in the system.

- The IBM-supplied installation exit rejects the CANCEL, RECEIVE, and OUTPUT commands because no user ID is available in the background.
- You must specify operands on the STATUS command. The system issues an error message if none are supplied.
- A PROFILE command with a PREFIX (userid) operand is required if you want a user ID to be prefixed to all data-set-names, or if something other than a null value is required for the CLIST symbolic variable, &SYSPREF.
- The CLIST symbolic variable &SYSUID has a null value.
- When a data set containing a job to be submitted in the background does not contain a JOB statement, SUBMIT generates one. It attempts to construct the job name using the value on the USER operand. If there is no value with the USER operand, the system issues a warning message and uses default job name SUBMIT JB.
- When generating a JOB statement, SUBMIT does not insert a NOTIFY=user ID operand.

Note: In a system with RACF installed, there are fewer restrictions for users that are both RACF and TSO defined. When RACF is installed, TSO can establish profile attributes for a background job, provided the JOB statement contains a value for the USER operand. TSO accesses the UADS entry associated with the user identified on the USER operand on the JOB statement.

Figure 17-9. Processing Considerations (RACF and Non-RACF Systems)

Chapter 18. Invoking Programs or Commands from a Program with the TSO Service Facility

Introduction

To enhance system security, some installations give special authority to commands and programs that execute in restricted areas of system storage. System programmers are responsible for authorizing such commands and programs.

Authorized commands and programs can usually be invoked only from authorized environments. However, the TSO service facility allows a TSO user to invoke some authorized programs and commands (such as RECEIVE) from an unauthorized environment like VSAPL.

Note: ISPF is also an unauthorized environment, but in Release 2 and later releases it automatically invokes the TSO service facility to access authorized commands or programs.

When you create a program that uses authorized commands or programs, it must invoke the TSO service facility. Unauthorized commands or programs can use the TSO service facility to access any commands or programs. However, authorized commands or programs can use the TSO service facility to invoke only other authorized commands or programs.

Note that the EXEC, LOGON, and LOGOFF commands do not execute through the TSO service facility. For more information on these commands, see the discussion of the TSO service facility's parm2 later in this chapter.

The TSO service facility can be invoked in both foreground and background TSO sessions by either a command or a program. An applications program written in a high level language such as PLI, COBOL,FORTRAN, PASCAL, or assembler can use the TSO service facility.

TSO Service Facility Parameters

The TSO service facility is invoked via module IKJEFTSR or alias name TSOLNK. If you want to use a command or program, you must specify six parameters. If you want to pass parameters to a program, you can specify an optional seventh parameter when you invoke the TSO service facility. The seventh parameter is intended for use with assembler programs. The seven parameters are:

PARM1

This parameter contains a fullword of flags.

1. Bytes one and two are zeros.
2. Byte three is the error processing flag byte.
   - It contains either a binary zero (0) to show that no dump should be taken if the invoked function abends, or
   - It contains a binary one (1) to show that a dump should be taken if the invoked function abends.
3. Byte four is the function flag byte.
   - It contains either a binary one (1) to show that a TSO command is being invoked, or
   - It contains a binary two (2) to show a program is being invoked.

For non-assembler language programmers, this parameter can be thought of as an integer which contains the value of byte 3 multiplied by 256 plus the value of byte 4.

PARM2

The second parameter contains a character string, which contains the name of the command or program being invoked. If a command is invoked, the character string must also contain all the parameters for the command. If you are invoking an authorized program using assembler language, the invoked program must be in a member of a partitioned data set allocated to STEPLIB or LINKLIB. All the data sets in the concatenated STEPLIB must be authorized.

Note that the execution of the EXEC, LOGON, and LOGOFF commands remains pending until the program environment terminates. That is, if invoked from within a program, these commands would take effect after the program finishes, where you would ordinarily see a READY prompt. These commands provide a return code of 0 in parm4 if the syntax is accurate.

PARM3

The third parameter contains the length of the invoked command or program (parameter 2). This is automatically provided in some Fortran compilers.

PARM4

The fourth parameter will contain the return code of the invoked program or command specified in parm2.

PARM5

The meaning of the fifth parameter is dependent on the service facility return code found in register 15. If the service facility return code is 12, parm5 contains the abend reason code of the invoked function. If the service facility return code is 20, parm5 contains the service facility reason code. See Figure 18-2 for the meaning of the service facility reason codes.

PARM6

The sixth parameter will contain the ABEND code, if the requested program or command ends unsuccessfully.

PARM7 - optional

The seventh parameter contains a parameter list used to pass parameters to the invoked program. Parm7 is optional, and intended for use with assembler language programs. It can only be coded when a program (not a TSO command) is being invoked. The parameter list pointed to by this parameter is a variable length list of one to four parameters and the high order bit of the last address must be on to indicate the end of the list.

The parameters of the parm7 parameter list are:

- parameters 1-3
  - a halfword contains the parameter length in binary bytes, immediately followed by
  - a variable length data string
- parameter 4
  - a fullword containing 2 bytes of binary zeros, immediately followed by
  - 2 bytes containing the binary number of full words of data, immediately followed by
  - a variable length data string

The exact format of this parameter list will vary depending on the program being invoked.

The following pages contain return and reason codes from IKJEFTSR and examples of programs that invoke the TSO Service Facility.

Return Codes from IKJEFTSR

This table contains the return codes from the TSO service facility.

Return Codes	Meaning
0	IKJEFTSR and the requested program or command completed successfully.
4	The invoked function (program or command) had a non-zero return code
8	The invoked function was terminated because of an attention. If the application programmer wishes to notify the end user his application program should issue a message.
12	The invoked function abended. The 6th parameter contains the abend code. The reason code in the 5th parameter contains the reason code associated with the abend.
16	One of the first 6 parameters in the parameter list contains addresses of storage not accessible to the calling program.
20	The IKJEFTSR parameter list contains an error or the TSO service facility was invoked from a non-TSO environment. The 5th parameter contains the reason code associated with the error.
24	The TSO routines associated with IKJEFTSR encountered an unexpected failure.
28	The invoker of IKJEFTSR has AMODE 24 and the parameter list contains 31-bit addresses.

Figure 18-1. IKJEFTSR Return Codes

Reason Codes from IKJEFTSR

This reason code is found in parm5 if IKJEFTSR completes with a return code of 20.

Reason Codes	Meaning
4	The length of the parameter list was invalid. One of the following is true „ The invoker of the TSO service facility did not turn on the first bit of the last parameter to indicate the end of the list. „ The high order bit is on in any of the first five parameters. „ More than seven parameters are coded
8	The reserved flags (bytes 1 and 2) of the function(program or command) flag field pointed to by the first parameter are non-zero.
12	The function(program or command) flag byte (byte 4) of the flag field pointed to by the first parameter was invalid. It should contain a decimal one for a command or a decimal two for a program.
16	The function(program or command) flag byte (byte 4) of the flag field pointed to by the first parameter specified a command (contained a decimal one). However, a seventh parameter (program parameter list) was also coded. The seventh parameter can only be coded for the program function.
20	The abend processing flag byte is invalid. This byte (byte 3) of the flag field pointed to by the first parameter should contain either a decimal zero to request a dump, or a decimal one to indicate no dump is to be taken.
24	IKJEFTSR was invoked from a non-TSO environment. This service can only be used in a TSO (foreground or background) environment.
28	The function buffer length is invalid. The function buffer pointed to by the 2nd parameter must be greater than zero and less than 32K-5.
32	The program parameter list (pointed to by the seventh parameter of the TSO service routine parameter list) contains addresses of storage not accessible to the calling program.
36	The program parameter list pointed to by the 7th parameter is invalid.
40	The requested function (program or command) was not found.
44	IKJSCAN detected a syntax error in the function (program or command) name.
48	A command began with "%". CLISTs are not supported.
52	Unsupported background function (program or command).
56	The function (program or command) is authorized, but the a copy of the function could not be found in an authorized library.

Figure 18-2. IKJEFTSR Reason Codes

Examples of Invoking the TSO Service Facility

IKJEFTSR is invoked according to the rules of the applications programming language in use. Since certain programming languages can accept only up to six characters in a name, the alias TSOLNK may be used for IKJEFTSR. For link editing the user program, the SYSLIB concatenation must contain SYS1.LPALIB, in which TSOLNK and IKJEFTSR reside.

The following sample programs demonstrate the use of the TSO Service Facility to invoke commands and programs in:

- assembler
- FORTRAN
- PLI
- PASCAL
- COBOL.

In these examples, the term 'function' means the program or command IKJEFTSR invokes.

For further information about the TSO service facility, see *TSO Terminal Monitor Program and Service Routines Logic.*

```
*********************************************************
* THIS ASSEMBLER PROGRAM CALLS THE TSO SERVICE ROUTINE  *
* TO EXECUTE THE LISTBC COMMAND.                        *
*********************************************************
TSFCMD   CSECT
         STM   14,12,12(13)        ENTRY LINKAGE
         BALR  12,0
         USING *,12                USE R12 AS BASE REG
         ST    13,SAVE+4           SAVE CALLERS SAVE AREA
         LA    13,SAVE             HAVE POINTER TO THIS SAVE AREA
         L     15,=V(IKJEFTSR)     GET ADDRESS OF IKJEFTSR
   CALL  (15),(FLAGS,CMD,CMDLEN,RETCODE,REASONC,ABENDCD),VL
         LTR   15,15               WAS RETURN CODE FROM IKJEFTSR = 0?
         BZ    NOERROR             NO ERROR ---- PROCEED ON
*
*
* CHECK RETCODE, REASONC, AND ABENDCD AT THIS POINT
*
*
NOERROR  EQU  *                   CONTINUE ON WITH PROGRAM
*
*
*
         L     13,SAVE+4           GET CALLERS SAVE AREA
         LM    14,12,12(13)        EXIT LINKAGE
         BR    14                  RETURN TO SUPERVISOR
* DECLARES
SAVE     DS    18F
FLAGS    DS    0F
         DC    XL2'0000'           INITIALIZE TO ZERO
         DC    XL1'01'             SPECIFY DUMP TO BE TAKEN
         DC    XL1'01'             COMMAND TO BE EXECUTED
CMD      DC    C'LISTBC'           COMMAND TEXT
CMDLEN   DC    F'6'                LENGTH OF COMMAND NAME
RETCODE  DS    F                   FUNCTION RETURN CODE
REASONC  DS    F                   TSF REASON CODE
ABENDCD  DS    F                   ABEND CODE
         END
```

Figure 18-3. Assembler Program Demonstrating the Use of IKJEFTSR to Invoke a Command

```
C     THIS FORTRAN PROGRAM WILL INTERFACE WITH THE TSO SERVICE ROUTINE.
C     THE PROGRAM ISSUES THE LISTD COMMAND IN TSO AND THEN PRINTS OUT
C     THE COMMAND BUFFER  AND THE RETURN, REASON AND ABEND CODES
C     RESULTING FROM THE EXECUTION OF THE TSO SERVICE ROUTINE.
C     SINCE THIS PROGRAM DOES ITS OWN I/O AFTER IT CALLS THE
C     TSO SERVICE ROUTINE, THE USER MIGHT WANT TO ALLOCATE THE FILE
C     NAME FT06F001 TO THE TERMINAL WITH THE TSO COMMAND:
C          ALLOC F(FT06F001) DSN(*)
C     THIS PROGRAM WAS COMPILED ON THE FORTRAN G1 COMPILER
C
      EXTERNAL TSOLNK
      INTEGER PARM11,PARM12,PARM13,PARM14
      INTEGER PARM1,PARM3,PARM4,PARM5
      INTEGER PARM2(20),FILL
C     PLACE COMMAND NAME IN PARM2
      DATA PARM2  /'LIST','D ''S','YS1.','LINK','LIB''','    ','    ',
     +'    ','    ','    ','    ','    ','    ','    ','    ','    ',
     +'    ','    ','    ','/
      DATA FILL /'    '/
      PARM11 = 0
      PARM12 = 0
      PARM13 = 0
C     SPECIFY THAT A COMMAND IS TO BE EXECUTED
      PARM14 = 1
C     FILL IN THE CONTROL BITS OF THE FIRST PARAMETER
C       TO REQUEST DUMP OFF
      PARM1 = (PARM11*16**6)+(PARM12*16**4)+(PARM13*16**2)+PARM14
C     PUT THE COMMAND LENGTH INTO THE THIRD PARAMETER
      PARM3 = 80
C     ZERO OUT THE RETURNED VALUES BEFORE THE CALL
      PARM4 = 0
      PARM5 = 0
      PARM6 = 0
C     EXECUTE THE TSO SERVICE ROUTINE
      I = TSOLNK(PARM1,PARM2,PARM3,PARM4,PARM5,PARM6)
      WRITE (6,104)
C     PRINT OUT THE COMMAND EXECUTED
104   FORMAT (' ','COMMAND EXECUTED:  ')
C     PRINT OUT THE RETURNED VALUES
      WRITE (6,103) I
103   FORMAT (' ','THE TSOLNK RETURN CODE IS ',I6)
      WRITE (6,105) FILL,(PARM2(I),I=1,20)
105   FORMAT (21A4)
      WRITE (6,100) PARM4
100   FORMAT (' ','THE FUNCTION RETURN CODE IS ',I6)
      WRITE (6,101) PARM5
101   FORMAT (' ','     THE TSF REASON CODE IS ',I6)
      WRITE (6,102) PARM6
102   FORMAT (' ','THE ABEND  CODE IS ',I6)
      STOP
      END
```

Figure 18-4. Fortran Program Demonstrating the Use of TSOLNK to Invoke a Command

```
/*********************************************************************/
/*  THIS PLI PROGRAM WILL INTERFACE WITH THE TSO SERVICE ROUTINE.    */
/*  THE PROGRAM WILL ISSUE THE LISTA COMMAND IN TSO AND THEN PRINT   */
/*  OUT THE RETURN, REASON AND ABEND CODES AS A RESULT OF USING THE  */
/*  SERVICE.  SINCE THIS PROGRAM DOES ITS OWN I/O AFTER IT CALLS     */
/*  THE TSO SERVICE ROUTINE, THE USER MUST ALLOCATE THE FILE NAME    */
/*  FILEOUT, PREFERABLY TO THE TERMINAL WITH THE TSO COMMAND:        */
/*        ALLOC F(FILEOUT) DSN(*)                                    */
/*                                                                   */
/*  THIS PROGRAM WILL RUN ON THE OS/VS PLI COMPILER RELEASE 2 OR     */
/*  HIGHER.                                                          */
/*********************************************************************/
TSOCALL:
  PROCEDURE OPTIONS(MAIN);
/*********************************************************************/
/*  DECLARE PARAMETERS                                               */
/*********************************************************************/
DECLARE 1 PARM1,
          2 PARM11 FIXED BINARY (15,0), /* RESERVED                 */
          2 PARM13 BIT(8),              /* ABEND FLAG               */
                                        /* 0 -ABEND WITHOUT DUMPT   */
                                        /* 1 -ABEND WITH DUMP       */
          2 PARM14 BIT(8);              /* FUNCTION CODE            */
                                        /* 1 - COMMAND              */
                                        /* 2 - PROGRAM              */
DECLARE PARM2 CHARACTER(8);             /* COMMAND OR PROGRAM       */
DECLARE PARM3 FIXED BINARY(31,0);       /* LENGTH OF CMD OR PROG    */
DECLARE PARM4 FIXED BINARY(31,0);       /* FUNCTION RETURN CODE     */
DECLARE PARM5 FIXED BINARY(31,0);       /* TSF REASON CODE          */
DECLARE PARM6 FIXED BINARY(31,0);       /* FUNCTION ABEND CODE      */
/*********************************************************************/
/*  DECLARE OUTPUT FILE                                              */
/*********************************************************************/
DECLARE (FILEOUT) FILE;
/*********************************************************************/
/*  DECLARE TSOLNK ROUTINE PARAMETER LIST                            */
/*********************************************************************/
DECLARE TSOLNK ENTRY(                  /*                          */
  1,                                    /* STRUCTURE OF 4 BYTES     */
    2 FIXED BINARY(15,0),               /* BYTE 1 RESERVED          */
    2 BIT(8),                           /* BYTE 3 ABEND FLAG        */
    2 BIT(8),                           /* BYTE 4 FUNCTION FLAG     */
  CHARACTER (*),                        /* NAME OF PROGRAM OR CMD   */
  FIXED BINARY(31,0),                   /* LENGTH OF CMD OR PROG    */
  FIXED BINARY(31,0),                   /* FUNCTION RETURN CODE     */
  FIXED BINARY(31,0),                   /* REASON CODE              */
  FIXED BINARY(31,0)                    /* FUNCTION ABEND CODE      */
  )
EXTERNAL OPTIONS(ASSEMBLER RETCODE INTER);
DECLARE PLIRETV BUILTIN;
```

Figure 18-5 (Part 1 of 2). PLI Program Demonstrating the Use of TSOLNK to Invoke a Command

```
/*********************************************************************/
/*  START OF EXECUTABLE CODE                                         */
/*********************************************************************/
PARM13='00000001'B;                    /* DUMP YES OR NO             */
PARM14='00000001'B;                    /* COMMAND OR PROGRAM         */
PARM2 ='LISTA ST';                     /* FUNCTION NAME              */
PARM3 = 8;                             /* COMMAND LENGTH             */
/*********************************************************************/
/* CALL TSO SERVICE ROUTINE                                          */
/*********************************************************************/
CALL TSOLNK(PARM1,PARM2,PARM3,PARM4,PARM5,PARM6);
/*********************************************************************/

/* PRINT RESULTS OF CALL                                             */
/* PRINT OUT RETURN CODE OF TSO SERVICE ROUTINE                      */
/*********************************************************************/
PUT FILE (FILEOUT) EDIT (' THE TSOLNK RETURN CODE IS ',PLIRETV)
                        (A,F(3));
/*********************************************************************/
/* PRINT OUT RETURN CODE OF LISTA COMMAND PROCESSOR                  */
/*********************************************************************/
PUT FILE (FILEOUT) EDIT (' THE FUNCTION RETURN CODE IS ',PARM4)
                        (SKIP, A, F(3));
/*********************************************************************/
/* PRINT OUT REASON CODE                                             */
/*********************************************************************/
PUT FILE (FILEOUT) EDIT ('      THE TSF REASON CODE IS ',PARM5)
                        (SKIP, A, F(3));
/*********************************************************************/
/* PRINT OUT ABEND CODE OF LISTA COMMAND PROCESSOR                   */
/*********************************************************************/
PUT FILE (FILEOUT) EDIT (' THE FUNCTION ABEND CODE IS ',PARM6)
                        (SKIP, A, F(3));
 END TSOCALL;
```

Figure 18-5 (Part 2 of 2). PLI Program Demonstrating the Use of TSOLNK to Invoke a Command

```
(*****************************************************************)
(*                                                               *)
(*  THIS PASCAL PROGRAM WILL INTERFACE WITH THE TSO SERVICE ROUTINE*)
(*  THIS PROGRAM WILL ISSUE THE LISTC TSO COMMAND.  SINCE THIS   *)
(*  PROGRAM DOES ITS OWN I/O AFTER THE TSO COMMAND IS EXECUTED TO *)
(*  DISPLAY RETURN CODES, THE USER SHOULD ALLOCATE FILE "FILEOUT" *)
(*  TO THE TERMINAL.                                             *)
(*                                                               *)
(*                                                               *)
(*****************************************************************)
PROGRAM TSOINTER;
 TYPE
    PA4=PACKED ARRAY (.1..4.) OF CHAR;
    PA80=PACKED ARRAY (.1..80.) OF CHAR;
(*****************************************************************)
(*                                                               *)
(*  SET UP CALL TO TSOLNK - THE TSO SERVICE ROUTINE.             *)
(*  WITH PARAMETER LIST.                                         *)
(*                                                               *)
(*****************************************************************)
 PROCEDURE TSOLNK( VAR PARM1:PA4;
                   VAR PARM2:PA80;
                   VAR PARM3:INTEGER;
                   VAR PARM4:INTEGER;
                   VAR PARM5:INTEGER;
                   VAR PARM6:INTEGER);
 FORTRAN; (* THIS KEYWORD IS REQUIRED TO ESTABLISH LINKAGE TO TSF *)
 VAR
  PARM1:PA4;           (* WORD OF CONTROL BITS *)
  PARM2:PA80;          (* COMMAND BUFFER       *)
  PARM3:INTEGER;       (* LENGTH OF COMMAND    *)
  PARM4:INTEGER;       (* FUNCTION RETURN CODE                    *)
  PARM5:INTEGER;       (* TSO SERVICE ROUTINE REASON CODE         *)
  PARM6:INTEGER;       (* FUNCTION ABEND CODE                     *)
  FILEOUT:TEXT;        (* DECLARE OUTPUT FILE NAME                *)
   BEGIN
    PARM1(.1.):=CHR(0);      (* ZERO OUT                        *)
    PARM1(.2.):=CHR(0);      (* ZERO OUT BYTE                   *)
    PARM1(.3.):=CHR(1);      (* SPECIFY DUMP      *)
    PARM1(.4.):=CHR(1);      (* SPECIFY COMMAND TO BE EXECUTED *)
    PARM2:='LISTC';          (* FILL IN COMMAND BUFFER          *)
    PARM3:=5;                (* SPECIFY COMMAND LENGTH          *)
    PARM4:=0;        (* ZERO OUT TSO FUNCTION RETURN CODE        *)
    PARM5:=0;        (* ZERO OUT TSO SERVICE ROUTINE REASON CODE *)
    PARM6:=0;        (* ZERO OUT TSO FUNCTION ABEND CODE         *)
    TSOLNK(PARM1,
           PARM2,
           PARM3,
           PARM4,
           PARM5,
           PARM6);  (* INTERFACE WITH TSO SERVICE ROUTINE    *)
    WRITELN(FILEOUT, 'THE FUNCTION RETURN CODE IS ',PARM4);
                     (* PRINT OUT FUNCTION RETURN CODE    *)
    WRITELN(FILEOUT, '     THE TSR REASON CODE IS ',PARM5);
                     (* PRINT OUT TSR REASON CODE    *)
    WRITELN(FILEOUT, 'THE FUNCTION  ABEND CODE IS ',PARM6);
                     (* PRINT OUT FUNCTION ABEND CODE    *)
   END.
```

Figure 18-6. PASCAL Program Demonstrating the Use of TSOLNK to Invoke a Command

```
 ID DIVISION.
 PROGRAM-ID.   TSOSVR.
* THIS COBOL PROGRAM WILL INTERFACE WITH THE TSO SERVICE ROUTINE.
* THIS PROGRAM WILL ISSUE THE LISTBC COMMAND.
* SINCE THIS PROGRAM DOES
* ITS OWN I/O AFTER THE TSO COMMAND IS EXECUTED TO DISPLAY RETURN
* CODES, USER SHOULD ALLOCATE FILE "SYSPRT" TO THE TERMINAL.

* THIS PROGRAM WILL RUN ON THE OS/VS COBOL COMPILER RELEASE 3 OR
* HIGHER

 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.

*    DEFINE OUTPUT DEVICE

     SELECT TRMPRT ASSIGN TO UT-S-SYSPRT.
 DATA DIVISION.
 FILE SECTION.

     DEFINE OUTPUT FILE

 FD  TRMPRT
     LABEL RECORDS ARE OMITTED
     RECORD CONTAINS 133 CHARACTERS.

*    DEFINE OUTPUT RECORD

 01  OUTREC.
     02 OUT-LINE     PICTURE X(133).

 WORKING-STORAGE SECTION.

*    DEFINE OUTPUT RECORD FORM

 01  OUT-RECORD.
     02 CONTROL-CHAR PICTURE X VALUE SPACE.
     02 COMMENT      PICTURE X(25).
     02 OUT-VALUE    PICTURE +++++++++9.
     02 FILLER       PICTURE X(111) VALUE SPACES.

*    DEFINE COMMENT VALUES FOR OUTPUT RECORD FORM

 01  RETURN-COMMENT PICTURE X(25) VALUE 'FUNCTION RETURN CODE IS  '.
 01  REASON-COMMENT PICTURE X(25) VALUE '     TSF REASON CODE IS '.
 01  ABEND-COMMENT  PICTURE X(25) VALUE 'FUNCTION ABEND  CODE IS  '.

*    DEFINE FLAGS TO BE FULL WORDS WITH APPROPRIATE BITS ON

 01  FLAG1-ON  PICTURE S9(9) VALUE +16777216 COMP.
 01  FLAG2-ON  PICTURE S9(9) VALUE +65536 COMP.
 01  FLAG3-ON  PICTURE S9(9) VALUE +256 COMP.
 01  FLAG4-ON  PICTURE S9(9) VALUE +2 COMP.
 01  FLAG1-OFF PICTURE S9(9) VALUE +0 COMP.
 01  FLAG2-OFF PICTURE S9(9) VALUE +0 COMP.
 01  FLAG3-OFF PICTURE S9(9) VALUE +0 COMP.
 01  FLAG4-OFF PICTURE S9(9) VALUE +1 COMP.
```

Figure 18-7 (Part 1 of 3). COBOL Program Demonstrating the Use of TSOLNK to Invoke a Command

```
* DEFINE STORAGE FOR PARMS
*    PARM1 IS DECIMAL VALUE OF FLAGS
*    PARM2 IS COMMAND TEXT
*    PARM3 IS COMMAND LENGTH (SET TO 80)
*    PARM4 IS FUNCTION RETURN CODE VALUE FROM TSOLNK
*    PARM5 IS TSF REASON CODE VALUE FOR ABEND FROM TSOLNK
*    PARM6 IS FUNCTION ABEND CODE VALUE FROM TSOLNK

 01  PARM1 PICTURE S9(9) COMP.
 01  PARM2 PICTURE X(80).
 01  PARM3 PICTURE S9(9) VALUE +80 COMP.
 01  PARM4 PICTURE S9(9) VALUE +0 COMP.
 01  PARM5 PICTURE S9(9) VALUE +0 COMP.
 01  PARM6 PICTURE S9(9) VALUE +0 COMP.

 PROCEDURE DIVISION.

*    MOVE DESIRED COMMAND TO PARM2

 READY-COMMAND.
     MOVE SPACES TO PARM2.
     MOVE 'LISTBC' TO PARM2.

*    SET FLAGS BY ADDING APPROPRIATELY VALUED FLAG VARIABLES

 READY-FLAGS.
     MOVE 0 TO PARM1.

*    RESERVED FLAG

     ADD FLAG1-OFF TO PARM1.

*    RESERVED FLAG

     ADD FLAG2-OFF TO PARM1.

*    FLAG3-ON TO REQUEST ABEND WITH DUMP

     ADD FLAG3-ON  TO PARM1.

*    FLAG4-OFF TO REQUEST A TSO COMMAND (NOT A PROGRAM) BE INVOKED

     ADD FLAG4-OFF TO PARM1.

     CALL TSOLNK
```

Figure 18-7 (Part 2 of 3). COBOL Program Demonstrating the Use of TSOLNK to Invoke a Command

```
 CALL-TSOLNK.
      CALL 'TSOLNK' USING PARM1 PARM2 PARM3 PARM4 PARM5 PARM6.

*     PRINT RESULTS

 PRINT-COMMENTS.
      OPEN OUTPUT TRMPRT.

*     PRINT THE FUNCTION RETURN CODE

      MOVE RETURN-COMMENT TO COMMENT.
      MOVE PARM4 TO OUT-VALUE.
      WRITE OUTREC FROM OUT-RECORD.

*     PRINT THE TSF REASON CODE

      MOVE REASON-COMMENT TO COMMENT.
      MOVE PARM5 TO OUT-VALUE.
      WRITE OUTREC FROM OUT-RECORD.

*     PRINT THE FUNCTION ABEND CODE

      MOVE ABEND-COMMENT TO COMMENT.
      MOVE PARM6 TO OUT-VALUE.
      WRITE OUTREC FROM OUT-RECORD.

      CLOSE TRMPRT.
      STOP RUN.
```

Figure 18-7 (Part 3 of 3). COBOL Program Demonstrating the Use of TSOLNK to Invoke a Command

```
*****************************************************************
*  THIS ASSEMBLER PROGRAM CALLS THE TSO SERVICE FACILITY TO      *
*  EXECUTE THE LINKAGE EDITOR PROGRAM (SYS1.LINKLIB(IEWL)).      *
*  THIS PROGRAM PASSES ONE PARAMETER TO THE LINKAGE EDITOR.      *
*  TO SUCCESSFULLY EXECUTE THE PROGRAM, THE USER SHOULD          *
*  ALLOCATE THE FOLLOWING FILES: SYSUT1, SYSLMOD, SYSLIN AND     *
*  SYSPRINT.                                                     *
*****************************************************************
TSFPROG  CSECT
         STM   14,12,12(13)        ENTRY LINKAGE
         BALR  12,0
         USING *,12                USE R12 AS BASE REG
         ST    13,SAVE+4           SAVE CALLERS SAVE AREA
         LA    13,SAVE             HAVE POINTER TO THIS SAVE AREA
         L     15,=V(IKJEFTSR)     GET ADDRESS OF IKJEFTSR
   CALL  (15),(FLAGS,PGM,PGMLEN,RETCODE,REASONC,ABENDCD,PARMLIST),VL
         LTR   15,15               WAS RETURN CODE FROM IKJEFTSR = 0?
         BZ    NOERROR             NO ERROR ---- PROCEED ON
*
*
* CHECK RETCODE, REASONC, AND ABENDCD AT THIS POINT
*
*
NOERROR  EQU  *                   CONTINUE ON WITH PROGRAM
*
*
*
         L     13,SAVE+4           GET CALLERS SAVE AREA
         LM    14,12,12(13)        EXIT LINKAGE
         BR    14                  RETURN TO SUPERVISOR
* DECLARES
SAVE     DS    18F
FLAGS    DS    0F
         DC    XL2'0000'           INITIALIZE TO ZERO
         DC    XL1'01'             SPECIFY DUMP TO BE TAKEN
         DC    XL1'02'             PROGRAM TO BE EXECUTED
PGM      DC    C'IEWL'             NAME OF PROGRAM / LINKAGE EDITOR
PGMLEN   DC    F'4'                LENGTH OF PROGRAM NAME
RETCODE  DS    F                   FUNCTION RETURN CODE
REASONC  DS    F                   TSF REASON CODE
ABENDCD  DS    F                   ABEND CODE
PGMPARM1 DS    0F                  FIRST AND ONLY PARAMETER TO IEWL
         DC    H'8'                LENGTH OF PARM TO IEWL
         DC    C'MAP,XREF'         THE ACTUAL PARM TO IEWL
PARMLIST CALL  ,(PGMPARM1),VL,MF=L SET UP PARM LIST TO IEWL
         END
```

Figure 18-8. Assembler Program Demonstrating the Use of IKJEFTSR to Invoke a Program

```
C     THIS FORTRAN PROGRAM INVOKES THE IEBCOPY PROGRAM IN TSO AND
C     THEN PRINTS OUT THE COMMAND BUFFER  AND THE RETURN, REASON
C     AND ABEND CODES RESULTING FROM THE EXECUTION OF THE TSO
C     SERVICE ROUTINE.  TO USE THE EXAMPLE THE USER MUST ALLOCATE
C     THE FOLLOWING FILES:
C          ALLOC F(FT06FT001) DSN(*)
C          ALLOC F(SYSIN) DSN(YOUR.SYSIN)
C          ALLOC F(SYSPRINT) DSN(*)
C          ALLOC F(INDD) DSN(YOUR.INPDS)
C          ALLOC F(OUTDD) DSN(YOUR.OUTPDS)
C     THE EXAMPLE REQUIRES THE FOLLOWING CARD IN YOUR.SYSIN FILE:
C          EXAMPLE    COPY  OUTDD=OUTDD,INDD=INDD
C
C      THIS PROGRAM WAS COMPILED ON THE FORTRAN G1 COMPILER
C
       EXTERNAL TSOLNK
       INTEGER PARM11,PARM12,PARM13,PARM14
       INTEGER PARM1,PARM3,PARM4,PARM5
       INTEGER PARM2(20),FILL
C      PLACE PROGRAM NAME IN PARM2
       DATA PARM2   /'IEBC','OPY ',','    ','    ','    ','    ','    ',
      +'    ','    ','    ','    ','    ','    ','    ','    ','    ',
      +'    ','    ','    ','    ','/
       DATA FILL /'    '/
       PARM11 = 0
       PARM12 = 0
       PARM13 = 0
C      SPECIFY THAT A PROGRAM IS TO BE EXECUTED
       PARM14 = 2
C      FILL IN THE CONTROL BITS OF THE FIRST PARAMETER
C        TO REQUEST DUMP OFF
       PARM1 = (PARM11*16**6)+(PARM12*16**4)+(PARM13*16**2)+PARM14
C      PUT THE PROGRAM LENGTH INTO THE THIRD PARAMETER
       PARM3 = 80
C      ZERO OUT THE RETURNED VALUES BEFORE THE CALL
       PARM4 = 0
       PARM5 = 0
       PARM6 = 0
C      EXECUTE THE TSO SERVICE ROUTINE
       CALL TSOLNK(PARM1,PARM2,PARM3,PARM4,PARM5,PARM6)
       WRITE (6,104)
C      PRINT OUT THE FUNCTION EXECUTED
104    FORMAT (' ','FUNCTION EXECUTED:  ')
       WRITE (6,105) FILL,(PARM2(I),I=1,20)
105    FORMAT (21A4)
C      PRINT OUT THE RETURNED VALUES
C      PRINT OUT THE FUNCTION RETURN CODE
       WRITE (6,100) PARM4
100    FORMAT (' ','THE FUNCTION RETURN CODE IS ',I6)
C      PRINT OUT THE TSF REASON CODE
       WRITE (6,101) PARM5
101    FORMAT (' ','     THE TSF REASON CODE IS ',I6)
C      PRINT OUT THE FUNCTION ABEND CODE
       WRITE (6,102) PARM6
102    FORMAT (' ','THE FUNCTION ABEND  CODE IS ',I6)
       STOP
       END
```

Figure 18-9. Fortran Program Demonstrating the Use of TSOLNK to Invoke a Program

```
/*********************************************************************/
/*  THIS PL/I PROGRAM ISSUES THE IEBCOPY PROGRAM IN TSO AND THEN     */
/*  PRINTS OUT THE COMMAND BUFFER  AND THE RETURN, REASON AND        */
/*  ABEND CODES RESULTING FROM THE EXECUTION OF THE TSO SERVICE      */
/*  ROUTINE.  TO USE THE EXAMPLE THE USER MUST ALLOCATE THE          */
/*  FOLLOWING FILES:                                                 */
/*       ALLOC F(FILEOUT) DSN(*)                                     */
/*       ALLOC F(SYSIN) DSN(YOUR.SYSIN)                              */
/*       ALLOC F(SYSPRINT) DSN(*)                                    */
/*       ALLOC F(INDD) DSN(YOUR.INPDS)                               */
/*       ALLOC F(OUTDD) DSN(YOUR.OUTPDS)                             */
/*  THE EXAMPLE REQUIRES THE FOLLOWING CARD IN YOUR.SYSIN FILE:      */
/*       EXAMPLE    COPY  OUTDD=OUTDD,INDD=INDD                      */
/*                                                                   */
/*  THIS PROGRAM WILL RUN ON THE OS/VS PLI COMPILER RELEASE 2 OR     */
/*  HIGHER.                                                          */
/*********************************************************************/
TSOCALL:
  PROCEDURE OPTIONS(MAIN);
DECLARE 1 PARM1,
          2 PARM11 FIXED BINARY (15,0), /* RESERVED                 */
          2 PARM13 BIT(8),              /* ABEND FLAG               */
                                        /* 0 -ABEND WITHOUT DUMPT   */
                                        /* 1 -ABEND WITH DUMP       */
          2 PARM14 BIT(8);              /* FUNCTION CODE            */
                                        /* 1 - COMMAND              */
                                        /* 2 - PROGRAM              */
DECLARE PARM2 CHARACTER(8);             /* COMMAND OR PROGRAM       */
DECLARE PARM3 FIXED BINARY(31,0);       /* LENGTH OF CMD OR PROG    */
DECLARE PARM4 FIXED BINARY(31,0);       /* FUNCTION RETURN CODE     */
DECLARE PARM5 FIXED BINARY(31,0);       /* TSF REASON CODE          */
DECLARE PARM6 FIXED BINARY(31,0);       /* FUNCTION ABEND CODE      */
DECLARE (FILEOUT) FILE;                 /* PLI OUTPUT FILE          */
DECLARE TSOLNK ENTRY(                   /*                          */
  1,                                    /* STRUCTURE OF 4 BYTES     */
    2 FIXED BINARY(15,0),               /* BYTE 1 RESERVED          */
    2 BIT(8),                           /* BYTE 3 ABEND FLAG        */
    2 BIT(8),                           /* BYTE 4 FUNCTION FLAG     */
  CHARACTER (*),                        /* NAME OF PROGRAM OR CMD   */
  FIXED BINARY(31,0),                   /* LENGTH OF CMD OR PROG    */
  FIXED BINARY(31,0),                   /* FUNCTION RETURN CODE     */
  FIXED BINARY(31,0),                   /* TSF REASON CODE          */
  FIXED BINARY(31,0)                    /* FUNCTION ABEND CODE      */
  )
  EXTERNAL OPTIONS(ASSEMBLER RETCODE INTER);
DECLARE PLIRETV BUILTIN;
```

Figure 18-10 (Part 1 of 2). PLI Program Demonstrating the Use of TSOLNK to Invoke a Program

```
/*****************************************************************/
/*  START OF EXECUTABLE CODE                                     */
/*****************************************************************/
PARM13='00000001'B;                    /* DUMP YES OR NO          */
PARM14='00000010'B;                    /* COMMAND OR PROGRAM      */
PARM2 = 'IEBCOPY';                     /* FUNCTION NAME           */
PARM3 = 7;                             /* LENGTH OF PROGRAM       */
CALL TSOLNK(PARM1,PARM2,PARM3,PARM4,PARM5,PARM6);
                                        /* CALL TSO SERVICE ROUTINE */
     PUT FILE (FILEOUT) EDIT (' THE TSOLNK RETURN CODE IS ',PLIRETV)
                            (A,F(3)); /* PRINT OUT RETURN CODE OF
                                         TSO SERVICE ROUTINE  */
     PUT FILE (FILEOUT) EDIT (' THE FUNCTION RETURN CODE IS ',PARM4)
                       (SKIP, A, F(3)); /* PRINT OUT RETURN CODE OF
                                           IEBCOPY PROGRAM         */
     PUT FILE (FILEOUT) EDIT ('      THE TSF REASON CODE IS ',PARM5)
                       (SKIP, A, F(3)); /* PRINT OUT TSF REASON CODE*/
     PUT FILE (FILEOUT) EDIT ('  THE FUNCTION ABEND CODE IS ',PARM6)
                       (SKIP, A, F(3)); /* PRINT OUT ABEND CODE
                                           FOR IEBCOPY */
 END TSOCALL;
```

Figure 18-10 (Part 2 of 2). PLI Program Demonstrating the Use of TSOLNK to Invoke a Program

```
(*********************************************************************)
(*                                                                   *)
(*  THIS PASCAL PROGRAM ISSUES THE IEBCOPY PROGRAM IN TSO AND        *)
(*  THEN PRINTS OUT THE COMMAND BUFFER  AND THE RETURN, REASON       *)
(*  AND ABEND CODES RESULTING FROM THE EXECUTION OF THE TSO          *)
(*  SERVICE ROUTINE.  TO USE THE EXAMPLE THE USER MUST ALLOCATE      *)
(*  THE FOLLOWING FILES:                                             *)
(*       ALLOC F(OUTPUT) DSN(*)                                      *)
(*       ALLOC F(SYSIN) DSN(YOUR.SYSIN)                              *)
(*       ALLOC F(SYSPRINT) DSN(*)                                    *)
(*       ALLOC F(INDD) DSN(YOUR.INPDS)                               *)
(*       ALLOC F(OUTDD) DSN(YOUR.OUTPDS)                             *)
(*  THE EXAMPLE REQUIRES THE FOLLOWING CARD IN YOUR.SYSIN FILE:      *)
(*       EXAMPLE    COPY  OUTDD=OUTDD,INDD=INDD                      *)
(*                                                                   *)
(*                                                                   *)
(*********************************************************************)
PROGRAM TSOINTER;
 TYPE
    PA4=PACKED ARRAY (.1..4.) OF CHAR;
    PA80=PACKED ARRAY (.1..80.) OF CHAR;
(*********************************************************************)
(*                                                                   *)
(*  SET UP CALL TO TSOLNK - THE TSO SERVICE ROUTINE.                 *)
(*  WITH PARAMETER LIST.                                             *)
(*                                                                   *)
(*********************************************************************)
 PROCEDURE TSOLNK( VAR PARM1:PA4;
                   VAR PARM2:PA80;
                   VAR PARM3:INTEGER;
                   VAR PARM4:INTEGER;
                   VAR PARM5:INTEGER;
                   VAR PARM6:INTEGER);
 FORTRAN; (* THIS KEYWORD IS REQUIRED TO ESTABLISH LINKAGE TO TSF *)
 VAR
  PARM1:PA4;           (* WORD OF CONTROL BITS *)
  PARM2:PA80;          (* PROGRAM BUFFER       *)
  PARM3:INTEGER;       (* LENGTH OF PROGRAM    *)
  PARM4:INTEGER;       (* FUNCTION RETURN CODE                    *)
  PARM5:INTEGER;       (* TSO SERVICE ROUTINE REASON CODE         *)
  PARM6:INTEGER;       (* FUNCTION ABEND CODE                     *)
  FILEOUT:TEXT;        (* DECLARE OUTPUT FILE NAME                *)
```

Figure 18-11 (Part 1 of 2). PASCAL Program Demonstrating the Use of TSOLNK to Invoke a Program

```
BEGIN
 PARM1(.1.):=CHR(0);      (* ZERO OUT                          *)
 PARM1(.2.):=CHR(0);      (* ZERO OUT BYTE                     *)
 PARM1(.3.):=CHR(1);      (* SPECIFY DUMP      *)
 PARM1(.4.):=CHR(2);      (* SPECIFY PROGRAM TO BE EXECUTED *)
 PARM2:='IEBCOPY';        (* FILL IN PROGRAM BUFFER            *)
 PARM3:=7;                (* SPECIFY PROGRAM LENGTH            *)
 PARM4:=0;        (* ZERO OUT FUNCTION RETURN CODE *)
 PARM5:=0;        (* ZERO OUT TSF REASON CODE      *)
 PARM6:=0;        (* ZERO OUT FUNCTION ABEND CODE  *)
 TSOLNK(PARM1,
        PARM2,
        PARM3,
        PARM4,
        PARM5,
        PARM6);  (* INTERFACE WITH TSO SERVICE ROUTINE    *)
 WRITELN(FILEOUT, 'THE FUNCTION RETURN CODE IS ',PARM4);
                  (* PRINT OUT FUNCTION RETURN CODE    *)
 WRITELN(FILEOUT, '     THE TSF REASON CODE IS ',PARM5);
                  (* PRINT OUT TSF REASON CODE    *)
 WRITELN(FILEOUT, 'THE FUNCTION  ABEND CODE IS ',PARM6);
                  (* PRINT OUT FUNCTION ABEND CODE    *)
END.
```

Figure 18-11 (Part 2 of 2). PASCAL Program Demonstrating the Use of TSOLNK to Invoke a Program

```
 ID DIVISION.
 PROGRAM-ID.   TSOSVR.
*     THIS COBOL PROGRAM ISSUES THE IEBCOPY PROGRAM IN TSO AND
*     THEN PRINTS OUT THE COMMAND BUFFER  AND THE RETURN, REASON
*     AND ABEND CODES RESULTING FROM THE EXECUTION OF THE TSO
*     SERVICE ROUTINE.  TO USE THE EXAMPLE THE USER MUST ALLOCATE
*     THE FOLLOWING FILES:
*          ALLOC F(SYSPRT) DSN(*)
*          ALLOC F(SYSIN) DSN(YOUR.SYSIN)
*          ALLOC F(SYSPRINT) DSN(*)
*          ALLOC F(INDD) DSN(YOUR.INPDS)
*          ALLOC F(OUTDD) DSN(YOUR.OUTPDS)
*     THE EXAMPLE REQUIRES THE FOLLOWING CARD IN YOUR.SYSIN FILE:
*          EXAMPLE    COPY  OUTDD=OUTDD,INDD=INDD
*
* THIS PROGRAM WILL RUN ON THE OS/VS COBOL COMPILER RELEASE 3 OR
* HIGHER

 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.

*    DEFINE OUTPUT DEVICE

     SELECT TRMPRT ASSIGN TO UT-S-SYSPRT.
 DATA DIVISION.
 FILE SECTION.

     DEFINE OUTPUT FILE

 FD  TRMPRT
     LABEL RECORDS ARE OMITTED
     RECORD CONTAINS 133 CHARACTERS.

*    DEFINE OUTPUT RECORD

 01  OUTREC.
     02 OUT-LINE     PICTURE X(133).

 WORKING-STORAGE SECTION.

*    DEFINE OUTPUT RECORD FORM

 01  OUT-RECORD.
     02 CONTROL-CHAR PICTURE X VALUE SPACE.
     02 COMMENT      PICTURE X(25).
     02 OUT-VALUE    PICTURE +++++++++9.
     02 FILLER       PICTURE X(111) VALUE SPACES.

*    DEFINE COMMENT VALUES FOR OUTPUT RECORD FORM

 01  RETURN-COMMENT PICTURE X(25) VALUE 'FUNCTION RETURN CODE IS  '.
 01  REASON-COMMENT PICTURE X(25) VALUE '     TSF REASON CODE IS '.
 01  ABEND-COMMENT  PICTURE X(25) VALUE 'FUNCTION ABEND  CODE IS  '.
```

Figure 18-12 (Part 1 of 3). COBOL Program Demonstrating the Use of TSOLNK to Invoke a Program

```
*    DEFINE FLAGS TO BE FULL WORDS WITH APPROPRIATE BITS ON

 01  FLAG1-ON  PICTURE S9(9) VALUE +16777216 COMP.
 01  FLAG2-ON  PICTURE S9(9) VALUE +65536 COMP.
 01  FLAG3-ON  PICTURE S9(9) VALUE +256 COMP.
 01  FLAG4-ON  PICTURE S9(9) VALUE +2 COMP.
 01  FLAG1-OFF PICTURE S9(9) VALUE +0 COMP.
 01  FLAG2-OFF PICTURE S9(9) VALUE +0 COMP.
 01  FLAG3-OFF PICTURE S9(9) VALUE +0 COMP.
 01  FLAG4-OFF PICTURE S9(9) VALUE +1 COMP.

* DEFINE STORAGE FOR PARMS
*    PARM1 IS DECIMAL VALUE OF FLAGS
*    PARM2 IS FUNCTION TEXT
*    PARM3 IS FUNCTION LENGTH (SET TO 80)
*    PARM4 IS FUNCTION RETURN CODE VALUE FROM TSOLNK
*    PARM5 IS TSF REASON CODE VALUE FROM TSOLNK
*    PARM6 IS FUNCTION ABEND CODE VALUE FROM TSOLNK

 01  PARM1 PICTURE S9(9) COMP.
 01  PARM2 PICTURE X(80).
 01  PARM3 PICTURE S9(9) VALUE +80 COMP.
 01  PARM4 PICTURE S9(9) VALUE +0 COMP.
 01  PARM5 PICTURE S9(9) VALUE +0 COMP.
 01  PARM6 PICTURE S9(9) VALUE +0 COMP.

 PROCEDURE DIVISION.

*    MOVE DESIRED FUNCTION TO PARM2

 READY-COMMAND.
     MOVE SPACES TO PARM2.
     MOVE 'IEBCOPY' TO PARM2.

*    SET FLAGS BY ADDING APPROPRIATELY VALUED FLAG VARIABLES
```

Figure 18-12 (Part 2 of 3). COBOL Program Demonstrating the Use of TSOLNK to Invoke a Program

```
 READY-FLAGS.
      MOVE 0 TO PARM1.

*     RESERVED FLAG

      ADD FLAG1-OFF TO PARM1.

*     RESERVED FLAG

      ADD FLAG2-OFF TO PARM1.

*     FLAG3-ON TO REQUEST ABEND WITH DUMP

      ADD FLAG3-ON  TO PARM1.

*     FLAG4-OFF TO REQUEST A TSO PROGRAM (NOT A COMMAND) BE INVOKED

      ADD FLAG4-ON TO PARM1.

      CALL TSOLNK

 CALL-TSOLNK.
      CALL 'TSOLNK' USING PARM1 PARM2 PARM3 PARM4 PARM5 PARM6.

*     PRINT RESULTS

 PRINT-COMMENTS.
      OPEN OUTPUT TRMPRT.

*     PRINT THE FUNCTION RETURN CODE

      MOVE RETURN-COMMENT TO COMMENT.
      MOVE PARM4 TO OUT-VALUE.
      WRITE OUTREC FROM OUT-RECORD.

*     PRINT THE TSF REASON CODE

      MOVE REASON-COMMENT TO COMMENT.
      MOVE PARM5 TO OUT-VALUE.
      WRITE OUTREC FROM OUT-RECORD.

*     PRINT THE FUNCTION ABEND CODE

      MOVE ABEND-COMMENT TO COMMENT.
      MOVE PARM6 TO OUT-VALUE.
      WRITE OUTREC FROM OUT-RECORD.

      CLOSE TRMPRT.
      STOP RUN.
```

Figure 18-12 (Part 3 of 3). COBOL Program Demonstrating the Use of TSOLNK to Invoke a Program

Appendix A. Full Screen Logon Processing

Full screen logon menus are available to a 3270 terminal user if the display format is 24 rows by 80 columns or larger. After you enter LOGON *userid,* TSO displays a menu containing the logon operand fields relevant to your session. If you enter more operands than your user ID on the LOGON command, TSO accepts and displays them, except for the old and new password fields for a RACF-defined user, and the current password field for a non-RACF-defined user. If you enter the password operand on the LOGON command, TSO ignores it.

Unless you override default values on the LOGON command, logon operand values default to the values present on the menu the last time you logged on. Excluding user ID and password values, you may override one or more of the default values on the LOGON command or on the menu itself.

If you are a RACF-defined user, you may change your password by entering your password in the PASSWORD field and by entering the value that is to replace the current password in the NEW PASSWORD field.

When you are satisfied with the values on the logon menu, press the appropriate key to transmit the LOGON command to TSO. To process a full screen menu, you must supply TSO with at least a valid user ID and a valid password. Also, TSO will not process a menu until your other operand values are acceptable.

Command Entry Field

You may also enter a single command, up to 80 characters in length, in the command field on the logon menu. TSO executes this command *after* any command entered in the PARM Field on the EXEC statement of the logon procedure.

Full Screen Logon Menus

Note: In the following discussion, *yourid* represents your user ID and the logon menu values are hypothetical.

If you are a non-RACF-defined user:

1. Enter LOGON ***yourid***. TSO displays the following menu with default values:

```
VS2 REL xxx TIME SHARING OPTION



PF1/PF13 ==>       Help PF3/PF15 ==>  Logoff PA1 ==>  Attention  PA2 ==> Reshow
You may request specific HELP information by entering a '?' in any ENTRY field.
ENTER LOGON PARAMETERS BELOW:        RACF LOGON PARAMETERS:

USERID     ===> YOURID

PASSWORD   ===>                      NEW PASSWORD  ===>

PROCEDURE  ===> PROC01               GROUP IDENT   ===>

ACCT NMBR  ===> G4002

SIZE       ===>

PERFORM    ===>

COMMAND    ===> CALL MY.PROG

ENTER AN 'S' BEFORE EACH OPTION DESIRED BELOW:

        -NOMAIL  -NONOTICE   -RECONNECT  -OIDCARD
```

2. Type in your password. For your protection, TSO inhibits the display of the password on the screen. If you want to change any of the other values on the menu, except userid, use the cursor positioning keys to position the cursor to the appropriate fields.

3. Press the appropriate key to transmit the logon menu to TSO. After TSO verifies your logon values, it displays the following preliminary message:

   ```
   YOURID LOGON IN PROGRESS AT 9:00 ON 2/7/80
   ```

4. When you are logged on, TSO displays the following message:

   ```
   READY
   ```

In this case TSO processes your command, CALL MY.PROG. Had you left the command entry field blank, you could begin entering commands immediately after the READY message.

If you are a RACF-defined user, and this is your first logon using the full screen logon menu:

1. Enter LOGON *yourid.* TSO displays the following menu with default values:

```
VS2 REL xxx TIME SHARING OPTION



PF1/PF13 ==> Help       PF3/PF15 ==> Logoff  PA1 ==> Attention  PA2 ==> Reshow
You may request specific HELP information by entering a '?' in any ENTRY field.
ENTER LOGON PARAMETERS BELOW:          RACF LOGON PARAMETERS:

USERID      ===> YOURID

PASSWORD    ===>                       NEW PASSWORD ===>

PROCEDURE   ===> PROC02                GROUP IDENT  ===>

ACCT NMBR   ===> N0341B

SIZE        ===>

PERFORM     ===>

COMMAND     ===>

ENTER AN 'S' BEFORE EACH OPTION DESIRED BELOW:

        -NOMAIL  -NONOTICE   -RECONNECT  -OIDCARD
```

2. Enter your password (commonly a group password) in the PASSWORD field, a new password in the NEW PASSWORD field, and, if you wish, other logon operand values.

3. Press the appropriate key to transmit the logon menu to TSO. After a preliminary message, TSO displays the READY message and, then, processes your command (if you specified one on the menu).

Error Prompting

Full screen logon processing checks the syntax of each entry field. When TSO discovers a syntax violation, it:

- Sounds an alarm (if terminal is so equipped)
- Displays an appropriate syntax message on the third line of the menu
- Places an asterisk (*) before the syntactically incorrect entry(ies)
- Highlights syntactically incorrect entry(ies)
- Positions the cursor in the first syntactically incorrect entry field.

TSO highlights nonsyntactical errors in the same ways. For example, if you enter an incorrect user ID that is syntactically valid on the LOGON command, TSO displays a menu containing any parameter values you entered on the LOGON

command, including the invalid user ID. If RACF is installed and active, TSO also displays RACF entry fields. No default values are available however.

Program Function Key Support for Full Screen Logon

During full screen logon processing, TSO processes interrupts caused only by the following program function keys.

PA1 Attention Key - Full screen logon attention processing erases the present screen. When the word LOGON appears at the top of the blank screen, you may enter LOGOFF or another LOGON command. If you enter anything else, TSO issues an appropriate error message.

PA2 Reshow Key - When you modify entries on a menu and then decide to start over, press PA2 to retrieve and re-display the entries you have overridden. TSO displays the logon information present on the screen when you last caused an interrupt.

PF1/PF13 HELP Key - Whenever you enter an incorrect value, TSO prompts you for a correct one. At this time, you may press either PF1 or PF13 to request syntactical and functional information about the entry field. TSO displays the appropriate text and, except for the password entry fields, also displays the incorrect data.

Note: Whenever you can enter data in the USERID, PASSWORD, or RACF PASSWORD field, pressing PF1 or PF13 requests information on *all* of the logon operands. In this case, when you are finished reading a panel of information, press the ENTER key to notify TSO that you are ready to view the next panel. Any parameter values you have entered are also displayed with the corresponding HELP text.

Question Mark (?) HELP Key - You may enter a question mark (?) in any entry field to obtain syntactical and functional information about an entry. When you enter a question mark in an entry field containing data, TSO erases it. In addition, whenever you can enter data in the USERID, PASSWORD, or RACF PASSWORD field, you may enter a question mark for any entry to obtain information about the entry(ies). If you enter more than one question mark, use the ENTER key to request the next panel of information.

PF3/PF15 End HELP Key - Press PF3 or PF15 to notify TSO that you want to stop viewing the information you requested using PF1 or PF13. If TSO is displaying HELP information for one field only, you may also use the ENTER key or any PF key to end the HELP display and return to the logon menu.

PF3/PF15 LOGOFF Key - Any time TSO is displaying the logon menu, you may press either PF3 or PF15 to log off.

Index

TSO Extensions User's Guide

SC28-1333-1

S370-39

Printed in U.S.A.

TSO Extensions
User's Guide

SC28-1333-1

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. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

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 comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name, company, mailing address, and date:

_____

_____

_____

_____

Please use pressure sensitive or other gummed tape to seal this form.

Cut or Fold Along Line

What is your occupation? _____

How do you use this publication? _____

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 or you may mail directly to the address in the Edition Notice on the back of the title page.)

TSO Extensions User's Guide

SC28-1333-1 S370-39

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 921-2
PO Box 390
Poughkeepsie, New York 12602

Fold and tape Please Do Not Staple Fold and tape

IBM®

Printed in U.S.A.

SC28-1333-01