File No. S360-36
Order No. GC28-6764-2

OS

Systems Reference Library

IBM System/360 Operating System:

Time Sharing Option

Guide to Writing a Terminal Monitor Program or a Command Processor

OS Release 21.6

This publication describes features of TSO that can be replaced, modified, or added to by each installation of TSO, to adapt the command system to the installation's particular needs. The manual is intended for programmers whose responsibility is to modify the portions of TSO that communicate directly with the user at the terminal.

The publication discusses the Terminal Monitor Program and the Command Processors from the viewpoint of their replaceability, and describes the programming features provided within TSO for user-written Terminal Monitor Programs, Command Processors, and applications programs. These features include:

Service Routines
Macro Instructions
SVCs
The Dynamic Allocation Interface Routine (DAIR)
The TEST Command Processor

Third Edition (August, 1972)

This is a major revision of, and obsoletes, GC28-6764-1 and Technical Newsletter GN28-2524. Changes are listed in the Summary of Amendments. Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

This edition applies to release 21.6 of the IBM System/360 Operating System, and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Changes are continually made to the information herein. Before using this publication in connection with the operation of IBM systems, consult the latest IBM System/360 and System/370 SRL Newsletter, Order No. GN20-0360, for the editions that are applicable and current.

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

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

Preface

This publication describes features of TSO that can be replaced, modified, or added to by each installation of TSO, to adapt the command system to the installation's particular needs. The manual is intended for programmers whose responsibility is to modify the portions of TSO that communicate directly with the user at the terminal.

The publication discusses the Terminal Monitor Program and the Command Processors from the viewpoint of their replaceability, and describes the programming features provided within TSO for user-written terminal monitor programs, command processors, and applications programs. These features include:

Service Routines
Macro Instructions
SVCs
The Dynamic Allocation Interface Routine (DAIR)
The TEST Command Processor

This publication contains information required by a programmer writing a terminal monitor program or a command processor for the Time Sharing Option. It discusses the functions that a Terminal Monitor Program or a command processor should provide, and it describes the macro instructions and service routines that can be used to provide these functions.

The book is divided into twelve sections:

- Introduction
- Terminal Monitor Program
- Command Processors
- Message Handling
- Attention Interruption Handling -- The STAX Service Routine
- Dynamic Allocation of Data Sets -- The Dynamic Allocation Interface Routine (DAIR)
- Using BSAM or QSAM for Terminal I/O
- Using the TSO I/O Service Routines for Terminal I/O
- Using the TGET/TPUT SVC for Terminal I/O
- Using Terminal Control Macro Instructions
- Command Scan and Parse -- Determining the Validity of Commands
- Testing a Newly Written Program -- The TEST Command

The first four sections describe the functions performed by a terminal monitor program or a command processor, and explain message processing conventions peculiar to the Time Sharing Option.

The next seven sections describe the macro instructions and service routines that a programmer can use to provide the required functions. These macro instructions and service routines can be used to schedule and process attention interruptions, to allocate, free, concatenate, and deconcatenate data sets during program execution, to provide I/O between a program and a terminal, to control terminal functions and attributes, and to determine the validity of commands, subcommands, and operands entering the system.

The last section describes the TEST command and how it can be used to test a newly written program at the terminal.

Prerequisite and Reference Publications

The reader of this publication should have a knowledge of the structure of the Time Sharing Option, as described in IBM System/360 Operating System: Time Sharing Option Guide, GC28-6698.

In addition, the reader should have the following publications available for reference:

IBM System/360 Principles of Operation, GA22-6821.

IBM System/360 Operating System:

Data Management for System Programmers, GC28-6550 (formerly System Programmer's Guide).

Data Management Macro Instructions, GC26-3794.

Data Management Services, GC26-3746.

Job Control Language Reference, GC28-6704.

Supervisor Services and Macro Instructions, GC28-6646.

System Control Blocks, GC28-6628.

Storage Estimates, GC28-6551.

Time Sharing Option:

Command Language Reference, GC28-6732.

Command Processor Program Logic Manual, GY28-6771 through GY28-6776.

Control Program, Program Logic Manual, GY27-7199.

Terminal Monitor Program and Service Routines, Program Logic Manual, GY28-6770.

Terminal User's Guide, GC28-6763.

Contents

Figures

Summary of Amendments for GC28-6764-2 OS Release 21.6

DAIRACB - DAIR Attribute Control Block

A correction is made to Figure 30.2.

USING THE PARSE SERVICE ROUTINE (IKJPARS)

The following three macro instructions are added to the Parse Service Routine.

- IKJTERM
- IKJOPER
- IKJRSVWD

These macro instructions provide syntax checking for the following positional parameter types.

- CONSTANT
- VARIABLE
- STATEMENT NUMBER
- EXPRESSION
- RESERVED WORD

Information is provided in the section entitled "Using the Parse Service Routine" (IKJPARS)

IKJNAME - Listing the Keyword or Reserved Word Parameter Names

The IKJNAME macro instruction can be used with the additional Parse macro instructions.

RETURN CODES FROM THE PARSE SERVICE ROUTINE

Return code 24 (decimal) is added to the Parse routine.

Summary of Amendments for GC28-6764-1 as Updated by GN28-2523 Component Release 360S-OS-586

DYNAMIC SPECIFICATION OF DCB PARAMETERS

New fields are defined for the following DAIR Parameter Blocks (DAPB):

Entry Code X'08'
Entry Code X'1C'
Entry Code X'24'
Entry Code X'30'

Two new parameter blocks are described:

DAPB, Entry Code X'34'
DAIRACB - DAIR Attribute Control Block

Summary of Amendments for GC28-6764-1 OS Release 21

LOGON PROCEDURE (Page 20)
A error is corrected in Figure 1.

INITIALIZATION OF THE TERMINAL MONITOR PROGRAM (Page 21)
The length subfield of the PARM field of the LOGON EXEC statement is described.

INVALID INFORMATION IN A JOB FILE CONTROL BLOCK (Page 32)
A previously used job file control block may contain invalid information from an earlier used DCB. The problem and the procedure to circumvent this problem is clarified.

ADDING COMMANDS TO THE TIME SHARING OPTION (Page 39)
The method of adding a new member to SYS1.CMDLIB or concatenating a new command library to SYS1.CMDLIB is clarified.

FORMATTING THE HELP DATA SET (Pages 40-42)
Method of adding new information to the HELP data set is clarified.

STAX MACRO INSTRUCTION (Pages 45,47)
Clarification and guidance on the use of this macro have been added.

DAIR PARAMETER BLOCKS (Pages 55-73)
Miscellaneous changes, corrections, and clarifications have been added.

DYNAMIC ALLOCATION INTERFACE ROUTINE (Pages 52-54,74-79)
Errors have been corrected, and new return codes have been added and others deleted for DAIR and Dynamic Allocation. Requirements for availability of a direct access device have been stressed. The description of the DAIR parameter list has been improved.

TERM=TS PARAMETER (Page 84)
Typographic error is corrected.

STACK PARAMETER BLOCK (Pages 97-98)
Corrections and clarifications are added.

PUTLINE PARAMETER BLOCK (Page 128)
Additional information on the PTPBOPUT field has been added.

PUTGET PARAMETER BLOCK (Page 155)
Corrections have been added.

PUTGET Return Codes (Page 167)
Clarifications and corrections have been made.

TPUT MACRO INSTRUCTION (Pages 169-172)
Describes the capability of the TJID operand when the macro is issued from a background program.

Describes two new operands, HIGHP and LOWP.

In addition, adds general clarifications to the TPUT description.

TGET MACRO INSTRUCTION (Pages 174-175)
Adds clarifications and corrections.

TERMINAL CONTROL MACRO INSTRUCTIONS (Pages 180-195)
The following macro instructions have been moved from the Supervisor and Data Management Macro Instructions SRL to this book:

GTSIZE, RTAUTOPT, SPAUTOPT, STATTN, STATUS, STAUTOCP, STAUTOLN, STBREAK, STCC, STCLEAR, STCOM, STSIZE, STTIMEOU, TCLEARQ.

Clarifications and corrections have made throughout.

COMMAND SCAN SERVICE ROUTINE (Page 197)
Adds new topic to describe command name syntax for a user-written command.

PARSE MACRO INSTRUCTIONS (Pages 213-215)
Typographic errors are corrected.

QUOTED STRING NOTATION (Pages 215-216)
The quoted string option SQSTRING is added to the IKJPOSIT macro instruction.

TEST COMMAND (Pages 255-257,261)
COPY, a new subcommand, and Assignment (=), an old subcommand previously omitted, have been added to the list of TEST subcommands. The use of symbolic addresses has been clarified.

TSO CONTROL BLOCKS (Page 263)
A legend has been added that describes the "bytes and alignment" column of each control block.

ENVIRONMENT CONTROL TABLE (ECT) (Page 264)
Errors have been corrected, and the tabulation has been clarified.

PROTECTED STEP CONTROL BLOCK (PSCB) (Pages 265-266)
Errors have been corrected, and the tabulation has been clarified. Information on the default unit name (PSCBGPNM) has been added.

TIME SHARING JOB BLOCK (Pages 267-269)
New fields have been added and clarifications have been made.

USER PROFILE TABLE (Page 270)
Descriptions have been improved.

Summary of Amendments for GC28-6764-0 as Updated by GN28-2484 OS Release 20.1

FLUSHING OF TGET AND TPUT BUFFERS

When an attention interruption is received, the TGET and TPUT buffers are flushed. The contents of these buffers (if any) are lost.

NEW RETURN CODES FROM DAIR

The meaning of DAIR return code 32 has been changed. DAIR return code 44 has been added.

NEW OPERAND ADDED TO THE STAX MACRO INSTRUCTION

A new operand, DEFER=YES or NO, has been added to the STAX macro instruction to allow the deferring of attention processing.

EDIT AND ASIS OPERANDS HAVE BEEN REDEFINED

The descriptions of the EDIT and ASIS operands have been rewritten. These changes appear in the GETLINE, PUTLINE, and PUTGET macro descriptions as well as in the TGET and TPUT macro descriptions.

TSEVENT MACRO INSTRUCTION, PPMODE, HAS BEEN DESCRIBED

The TSEVENT macro instruction should be issued by a newly written Terminal Monitor Program, to update SMF records and the TSO Trace Writer entries.

REVERSE MERGE INTO THE JOB FILE CONTROL BLOCK HAS BEEN DESCRIBED

A previously used JFCB may contain invalid information obtained from an earlier used Data Control Block.

NEW OPERANDS ON THE PUTGET MACRO INSTRUCTION

The TERM and ATTN operands have been added to the PUTGET macro instruction. These operands affect especially the processing of I/O from an Attention Exit.

Introduction

TSO, the Time Sharing Option of the IBM System/360 Operating System, consists of many, relatively small, functionally distinct modules of code. One major benefit of this modular construction is that the Time Sharing Option may be added to or modified to better suit the needs of the installation and each user. You can add to TSO, replace TSO-supplied code with your own, and delete those functions of TSO which you do not require.

TSO is composed of modules that perform timing, control, and accounting functions, and other modules that communicate with the user at the terminal and perform the work requested by him.

Modifications to the control program portions of TSO should be made only by system programmers responsible for the proper functioning of the Time Sharing Option within the System/360 MVT configuration of the operating system. These modifications are discussed in the Time Sharing Option Guide.

Each installation of the Time Sharing Option can replace those portions of TSO that communicate directly with the user at the terminal. The portions of TSO that communicate with the user are the Terminal Monitor Program (TMP) and the command processors.

If you choose to write your own Terminal Monitor Program or command processors, you can use service routines, interface routines, and macro instructions, supplied with TSO or modified to support TSO, to provide many of the functions required by a TMP or a command processor.

THE TERMINAL MONITOR PROGRAM (TMP) AND COMMAND PROCESSORS

The Terminal Monitor Program is a reenterable problem program that accepts and interprets commands, and causes the appropriate command processors to be scheduled and executed.

When a user logs on to TSO, he must specify, via the LOGON command, the name of a LOGON procedure. The program named in the EXEC statement in the LOGON procedure is attached during the log on as the Terminal Monitor Program. The program named in the EXEC statement can be either the TMP supplied with TSO, one provided by the installation, or one you have written yourself.

Any Terminal Monitor Program must be able to communicate with the user at the terminal, fetch and pass control to command processors, respond to abnormal terminations at its own task level or at lower levels, and respond to and process attention interruptions.

Once the log on has completed, the Terminal Monitor Program requests the user at the terminal to enter a command name. The TSO-supplied TMP writes a READY message to the terminal to request that a command be entered. The TMP determines if the response entered is a command, attaches the requested command processor, and the command processor performs the computing functions requested by the user at the terminal.

You can write your own command processors and add them to the TSO-supplied command library; you can concatenate your own command library to the one supplied with TSO, or you can replace the entire TSO command library with your own.

Command processors must be able to communicate with the user at the terminal, respond to abnormal terminations, process attention interruptions, and if required, fetch, pass control to, and respond to abnormal terminations of subcommand processors.

BASIC FUNCTIONS OF TERMINAL MONITOR PROGRAMS AND COMMAND PROCESSORS

You can see from the preceding discussion, that any Terminal Monitor Program and any command processor must provide four basic functions:

1. Both the TMP and command processors must be able to communicate with the user at the terminal.

2. The TMP must be able to fetch and pass control to a command processor. A command processor must be able to fetch and pass control to its subcommand processors if it has any.

3. Both the TMP and command processors must be able to intercept and investigate abnormal terminations.

4. Both the TMP and command processors must be able to respond to and process attention interruptions entered from the terminal.

You can provide each of these functions to a Terminal Monitor Program or a command processor by using a service routine or a macro instruction provided with or modified to support TSO.

Communicating with the User at the Terminal

With TSO there are three ways a program can communicate with a user at a terminal:

1. The BSAM or QSAM access methods. The major benefit of using BSAM or QSAM to process terminal I/O is that programs using these access methods do not become TSO dependent or device dependent and can execute either under TSO or in the batch environment.

2. The STACK, GETLINE, PUTLINE, and PUTGET I/O service routines. Reached through the STACK, GETLINE, PUTLINE, and PUTGET macro instructions, the I/O Service routines provide the following functions:

   STACK - The STACK service routine establishes and changes the source of input by adding elements to or deleting elements from, an internally maintained input stack. The top element on the input stack determines the current source of input.

   GETLINE - The GETLINE service routine obtains all input lines other than commands or subcommands, and responses to prompting messages (a prompting message asks the user at the terminal to supply required information). The GETLINE service routine returns these lines of input from the input source designated by the top element of the input stack.

   PUTLINE - The PUTLINE service routine formats output lines, writes them to the terminal, and chains second level messages to be written out in response to a question mark from the terminal.

   PUTGET - The PUTGET service routine writes a message to the terminal and obtains a response from the terminal. A message written to the user at the terminal which requires a response is called a conversational message.

3. The TGET and TPUT supervisor call. A supervisor call routine, SVC 93, is reached through the TGET and TPUT macro instructions. TGET and TPUT provide a route for I/O to a terminal. The functions are not as extensive, however, as those provided by the I/O service routines.

Each of these methods performs different functions and is thus suited for particular I/O situations. The programmer designing his own TMP or command processor must understand which of the I/O methods best provides the I/O support required in different programming situations.

Passing Control to Commands and Subcommands

A Terminal Monitor Program must be able to recognize a command name entered into the system, fetch the requested command processor, and pass control to it. A command processor must be able to perform the same functions when a subcommand name is entered.

You can use the Command Scan service routine to scan the input line for a syntactically valid command name or subcommand name, issue the BLDL macro instruction to search command libraries for the requested command processor or subcommand processor, and issue the ATTACH macro instruction to pass control to the requested routines.

When you write a command processor or subcommand processor, you can use the Parse macro instructions to describe to the Parse service routine the operands that may be entered with the command name. You can then use the Parse service routine to determine which operands are present in the input buffer. The Parse service routine compares the information you supplied in the Parse macro instructions with the contents of the input buffer. This comparison indicates which operands are present in the input line. The Parse service routine returns a list to the calling routine, indicating which operands were found in the buffer. These operands indicate to the processing routines which functions the user at the terminal is requesting.

Responding to Abnormal Terminations

One of the responsibilities of a programmer coding a routine to run within TSO is to do all possible to keep that routine from causing the abnormal termination of TSO. If you write your own Terminal Monitor Program or command processors, you should use the STAE macro instruction and the STAI operand on the ATTACH macro instruction to provide error handling exits.

Use the STAE macro instruction to provide the address of an error handling routine to be given control if any routine at the same task level as the error handling routine begins to terminate abnormally.

Use the STAI operand on the ATTACH macro instruction to provide the address of an error handling routine to be given control if a routine at a lower task level begins to terminate abnormally.

Responding to Attention Interruptions

The Terminal Monitor Program and any command processor that accepts subcommands must be able to respond to an attention interruption entered from the terminal. An attention interruption is interpreted within TSO as a signal that the user may want to request a new command or subcommand. You must provide attention exits that can obtain a line of input from the terminal and respond to that input.

Use the STAX service routine, reached through the STAX macro instruction, to build the control blocks and queues necessary for the system to recognize and schedule your attention handling routines.

OTHER FUNCTIONS PROVIDED WITH TSO

Aside from the four basic functions provided by a Terminal Monitor Program or a command processor, other functions, peculiar to time sharing, can be obtained using routines provided with TSO.

Two of these functions are:

1. The dynamic allocation of data sets.
2. The immediate, on-line testing of a newly written Terminal Monitor Program or command processor.

These two functions are provided through the Dynamic Allocation Interface Routine (DAIR), and the TEST command processor.

The Dynamic Allocation of Data Sets

The LOGON procedure named in the LOGON command contains DD statements that define the data sets to be used during a TSO session, and other DD statements, called DD DYNAMS. These DD DYNAMS do not define data sets; they are used by Dynamic Allocation routines to provide data sets requested during program execution by a Terminal Monitor Program or a command processor.

If you write your own Terminal Monitor Program or command processor, you can use the Dynamic Allocation Interface Routine (DAIR) to invoke Dynamic Allocation routines. Using DAIR, you can request Dynamic Allocation to:

- Obtain the current status of a data set.
- Allocate a data set.
- Free a data set.
- Concatenate data sets.
- Deconcatenate data sets.

Testing a Terminal Monitor Program or a Command Processor

After you have coded a new Terminal Monitor Program or command processor, you will want to test it before you enter it into the Time Sharing Option. You can use the TEST command to do this.

The TEST command permits a user at a terminal to test an assembly language program. You test a program by issuing the TEST command and the various TEST subcommands that perform the following basic functions:

- Execute the program under test from its starting address or from any address within the program.
- Display selected areas of the program as it appears in main storage, or display the contents of any of the registers.
- Interrupt the program under test at a specified location or locations.
- Change the contents of specified program locations in main storage or the contents of specific registers.

In addition to these basic debugging functions, you can use the TEST command processor to display various control blocks, program status words, or a main storage map of the program being tested.

SUMMARY

Most of the functions of a terminal monitor program or a command processor can be provided with macro instructions, service routines, or supervisor call routines supplied with the Time Sharing Option.

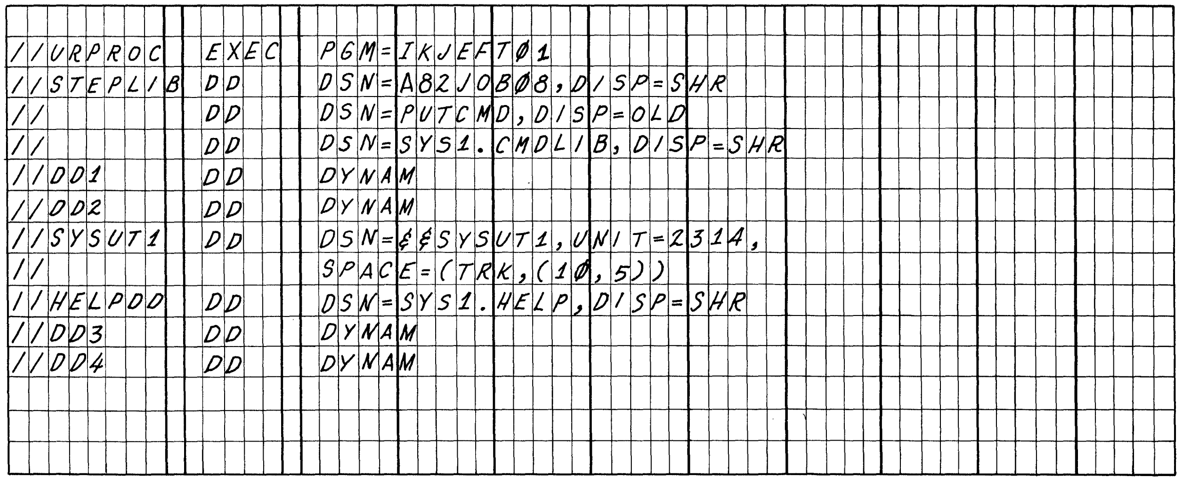
The following sections describe when and how to use these various macro instructions and routines.

The Terminal Monitor Program

The Terminal Monitor Program (TMP) is a reenterable problem program that provides an interface between the terminal user, command processors, and the Time Sharing Control Program. The TSO LOGON/LOGOFF Scheduler attaches the TMP. The TMP is the program you name on the EXEC statement of your LOGON cataloged procedure.

Specifying Data Sets at LOGON

The volumes that contain your data sets cannot be mounted during a terminal session. The volumes must be mounted before the terminal user logs onto the system. The LOGON procedure indicated on the LOGON command contains DD statements that define the data sets to be used during the TSO session, and other DD statements, called DD DYNAM statements, that do not define data sets. These DD DYNAM statements provide blank entries in the Task Input Output Table and the Data Set Extension. These entries are available for the dynamic allocation of previously unallocated data sets. Figure 1 shows an example of a user LOGON procedure containing four DD DYNAM entries. For a complete discussion of a LOGON procedure, see Time Sharing Option Guide.



```
//URPROC   EXEC PGM=IKJEFTØ1
//STEPLIB  DD   DSN=A82JOBØ8,DISP=SHR
//         DD   DSN=PUTCMD,DISP=OLD
//         DD   DSN=SYS1.CMDLIB,DISP=SHR
//DD1      DD   DYNAM
//DD2      DD   DYNAM
//SYSUT1   DD   DSN=¢¢SYSUT1,UNIT=2314,
//              SPACE=(TRK,(1Ø,5))
//HELPDD   DD   DSN=SYS1.HELP,DISP=SHR
//DD3      DD   DYNAM
//DD4      DD   DYNAM
```

Figure 1. A LOGON Procedure Containing Four DD DYNAM Entries

The Terminal Monitor Program you use can be the TMP supplied with TSO, or one provided by the installation, or one you have supplied yourself. If you choose to write your own Terminal Monitor Program, use the TSO service routines and macro instructions described in this book to help you code the TMP and fit it into the Time Sharing Option.

The TMP must be able to respond to the following four conditions:

1. Normal completion of a command processor or user program, and the requesting of another command.

2. An error causing termination of the TMP, a command processor, or a user program.

3. An attention request from the terminal, causing an interruption of the current program.

4. A STOP operator command, forcing a LOGOFF for the user.

This section explains how to respond to these conditions. It describes in general terms how the TSO-supplied TMP functions, and how it fits together with the rest of the Time Sharing Option. For a more specific description of the TSO-supplied TMP, see the TSO Terminal Monitor Program and Service Routines PLM.

Terminal Monitor Program Initialization

When the TMP is attached by the LOGON/LOGOFF scheduler:

- Register 1 contains the address of the value found in the PARM field of the EXEC statement in the LOGON cataloged procedure. The TSO-supplied TMP uses this PARM value as the first command requested. The first two bytes of the PARM value are on a halfword boundary and contain the length of the PARM value. (The length value does not include the two length bytes.)

- Register 13 contains the address of the register save area.

- Register 14 contains the return address of the LOGON/LOGOFF scheduler.

- Register 15 contains the entry point address of the TMP.

The TMP sets up the tables and control blocks it requires, loads the TIME command processor, sets up the STAE and STAI exits to respond to abnormal terminations, sets up the attention exits, builds the command buffer, and initializes the input stack to point to the terminal. The TMP then uses the EXTRACT macro instruction to obtain the addresses of the STOP/MODIFY ECB and the Protected Step Control Block (PSCB) built by the LOGON/LOGOFF scheduler.

The TSO-supplied Terminal Monitor Program attaches the command processor named in the EXEC statement PARM field. If no command was named as a PARM operand, the TMP issues a PUTGET macro instruction to obtain the first command. The TMP shares subpool 78 with the attached command processor but does not share subpool 0. The command processor, in turn, must share subpool 78 with any lower level tasks.

Requesting a Command

Figure 2 summarizes the steps taken by a Terminal Monitor Program to obtain a command, to pass control to that command, and to detach that command when it has finished processing.

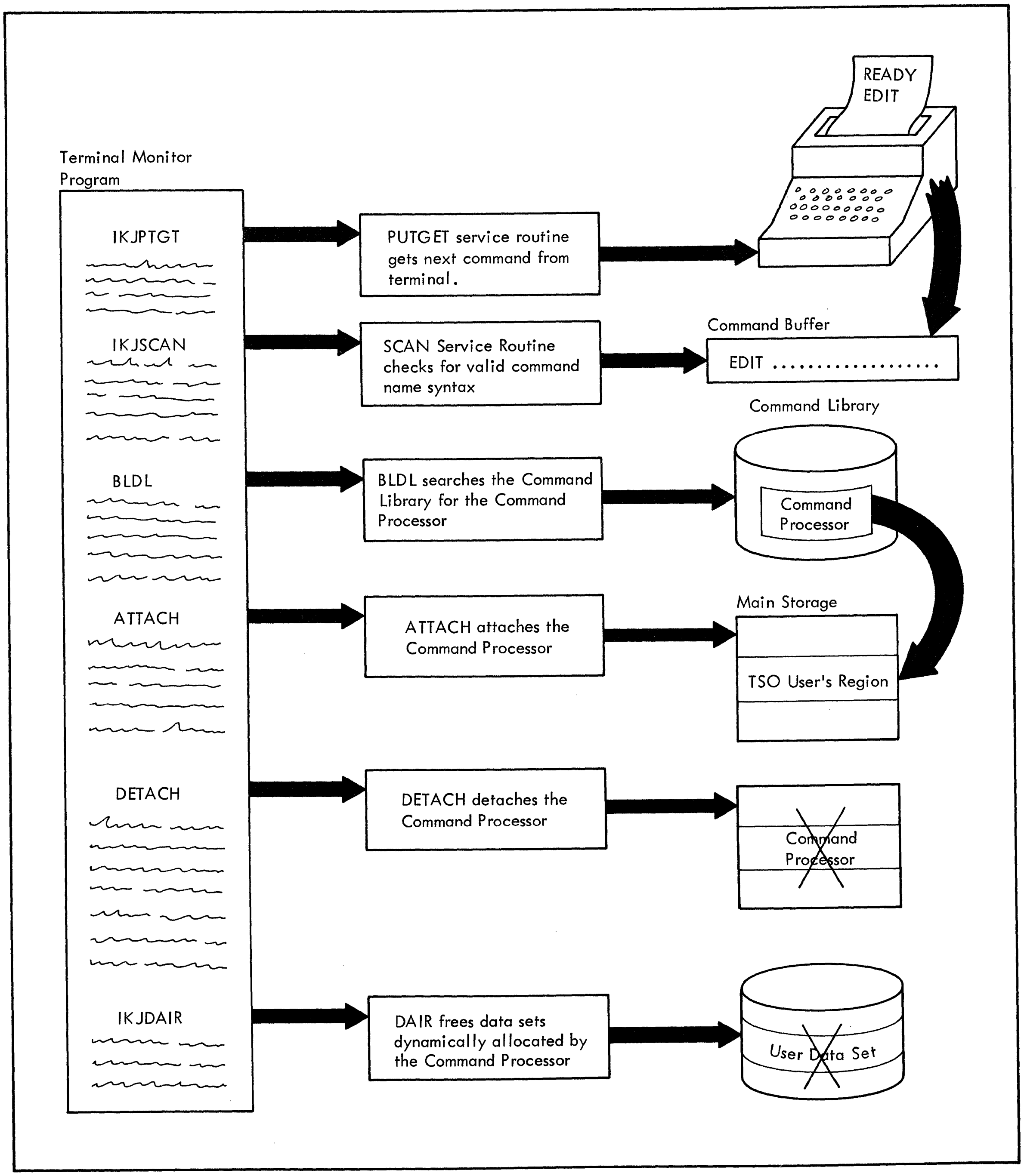

READY
EDIT
Terminal Monitor Program
IKJPTGT
IKJSCAN
BLDL
ATTACH
DETACH
IKJDAIR
PUTGET service routine gets next command from terminal.
SCAN Service Routine checks for valid command name syntax
BLDL searches the Command Library for the Command Processor
ATTACH attaches the Command Processor
DETACH detaches the Command Processor
DAIR frees data sets dynamically allocated by the Command Processor
Command Buffer
EDIT ....
Command Library
Command Processor
Main Storage
TSO User's Region
Command Processor
User Data Set

Figure 2. Requesting a Command

To request a command from the terminal, use the PUTGET service routine. The PUTGET service routine first writes a line to the terminal to inform the user that another command is expected, then returns a line entered in response to the request, and places that line into a command buffer.

Use the Command Scan service routine to determine that the line of input is a syntactically valid command name.

Use the BLDL macro instruction to search the command library or libraries for the command processor load module indicated by the command name, and use the ATTACH macro instruction (specifying a STAI exit routine) to pass control to the requested command processor.

Your TMP must create any parameters expected by the command processor and pass them to the newly attached command processor. The TSO-supplied TMP passes the address of a Command Processor Parameter List in register one. See the section headed "Interface with the I/O Service Routines".

When the command processor completes, the TMP issues a DETACH macro instruction for it, uses the DAIR service routine to mark dynamically allocated data sets available to be freed, and uses the PUTGET service routine to obtain another command.

Please note that the use of an installation-supplied program in place of the Terminal Monitor Program can result in invalid values for the core occupancy time field in SMF record 34, and may cause invalid TSO Trace Writer entries. This situation occurs only when a single user is assigned to a foreground region and the installation-supplied program runs to completion without being swapped out of main storage.

To avoid this problem, your user-written Terminal Monitor Program should issue the TSEVENT macro instruction, specifying the PPMODE operand, before attaching each command processor and after each command processor returns. This issuance of the TSEVENT macro instruction causes SMF record 34 and the TSO Trace Writer entries to be updated.

Issue the TSEVENT macro instruction as follows:

1. Set register one to point to the first character of the command name being attached or released.

2. Set the high order bit in register one to:

   1 if the command processor is beginning execution.

   0 if the command processor is ending.

3. Code the TSEVENT macro instruction as shown in Figure 3.

[label]	TSEVENT	PPMODE

Figure 3. The TSEVENT Macro Instruction Specifying PPMODE

Intercepting an ABEND

The Terminal Monitor Program must be able to recognize and respond to two basic types of ABEND situations:

1. An attached subtask, for example a command processor, is terminating abnormally.

2. The TMP itself or a program linked to by the TMP, for example TEST or Command Scan, is terminating abnormally.

INTERCEPTING A SUBTASK ABEND

When a subtask of the Terminal Monitor Program begins to terminate abnormally, the TMP STAI exit, specified by the TMP when it attached the subtask, receives control. The TMP STAI exit receives control under the TCB of the abending subtask. The subtask will already have performed its own STAE processing, if any was specified. Figure 4 shows the ABEND, STAE, STAI relationship.

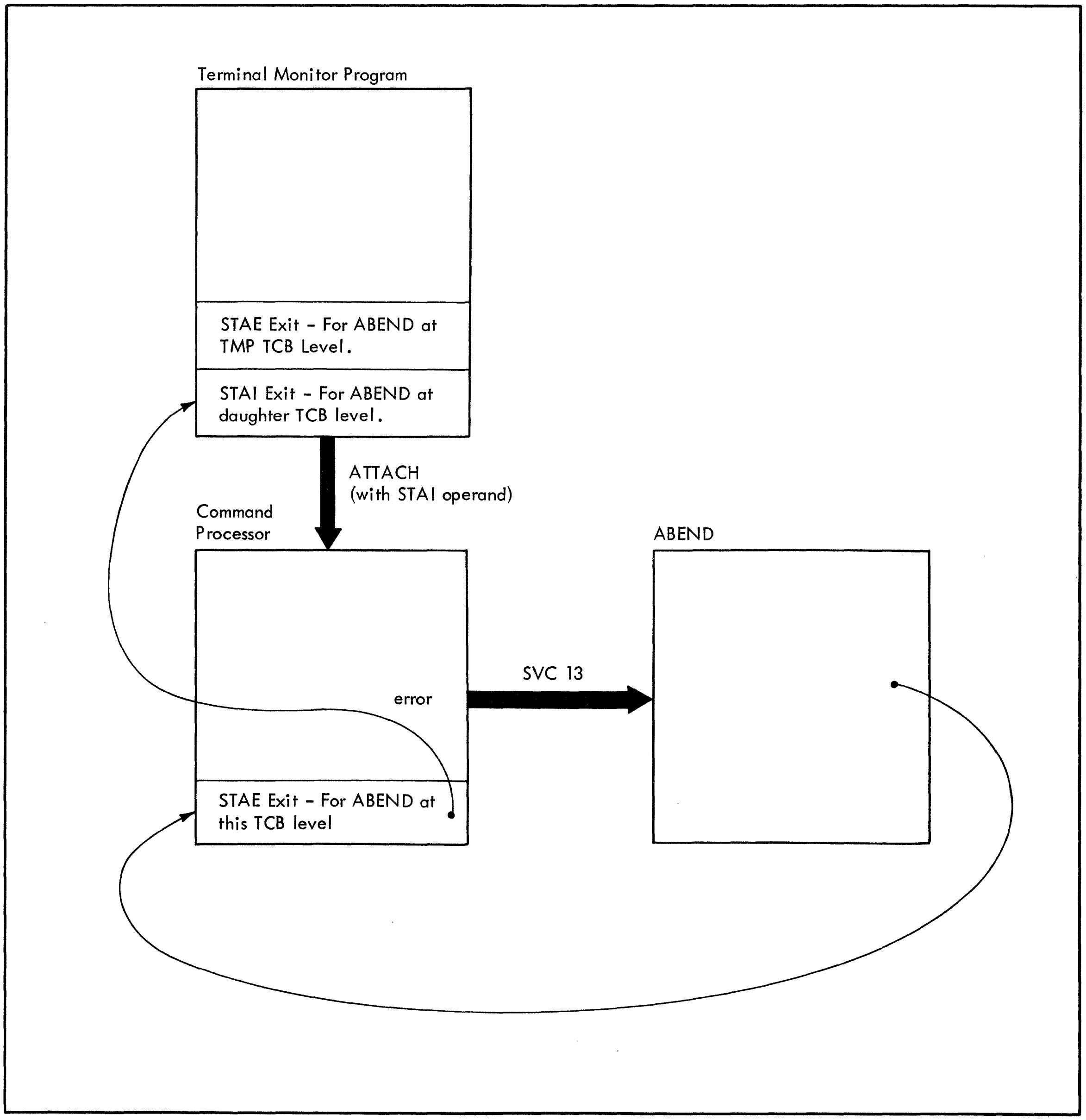

Terminal Monitor Program
STAE Exit - For ABEND at TMP TCB Level.
STAI Exit - For ABEND at daughter TCB level.
ATTACH (with STAI operand)
Command Processor
ABEND
SVC 13
error
STAE Exit - For ABEND at this TCB level

Figure 4. ABEND, STAE, STAI Relationship

The TMP must inform the user at the terminal of the ABEND situation, and allow the user to enter another command at this time. Use the PUTGET service routine, specifying the TERM operand, to inform the user of the ABEND and to return a line of input from the user.

The terminal user has four options:

1. He can allow the ABEND to continue by entering a null line (carriage return).

2. He can stop processing of the ABEND by entering a command name other than TEST or TIME.

3. He can request any secondary messages concerning the terminating program by entering a question mark.

4. He can place the terminating program under the control of the TEST command processor by entering the command name TEST.

Use the Command Scan service routine to determine what the user has entered at the terminal.

If he enters a null line, the TMP returns control to the ABEND routine, and the task is allowed to terminate abnormally. If he enters a command name, other then TEST and TIME, the TMP processes the new command name after detaching the incomplete subtask.

If the user enters a question mark, the PUTGET service routine causes the secondary level informational message chain (if one exists) to be written to the terminal, again puts out the message, and returns the response from the terminal.

If the user enters the command name TEST, the TMP passes control to the TEST command processor via a LINK macro instruction. If any operands were entered on the TEST command, the TMP detaches all subtasks before linking to the TEST command processor. If no operands were entered, the TMP does not detach any currently active subtasks. The user is requesting that the abnormally terminating task be run under the control of TEST.

When the TMP links to the TSO-supplied TEST command processor, register one must contain a pointer to a Test Parameter List (TPL). Figure 5 shows the format of the Test Parameter List you must build and pass to the TEST command processor.

Number of Bytes	Field	Contents or Meaning
4	TPLCBUF	The address of the Command buffer used by the last attached command processor.
4	TPLUPT	The address of the User Profile Table (UPT). The UPT is built by the LOGON/LOGOFF scheduler from information stored in the User Attribute Data Set (UADS) and from information contained in the LOGON command. The address of the UPT is found in the PSCBUPT field of the Protected Step Control Block (PSCB). See Appendix A for the format of the UPT.
4	TPLPSCB	The address of the Protected Step Control Block (PSCB). The PSCB is built by the LOGON/LOGOFF scheduler from information stored in the UADS. The TMP can obtain the address of the PSCB with the EXTRACT macro instruction. See Appendix A for the format of the PSCB.
4	TPLECT	The address of the Environment Control Table (ECT). The ECT must be built by the TMP during its initialization process and is used by the TSO service routines. See Appendix a for the format of the ECT.

Figure 5. The Test Parameter List (Part 1 of 3)

Number of Bytes	Field	Contents or Meaning
4	TPLTBUF	The address of the TEST command buffer. The TEST command buffer contains the TEST command and all operands entered by the terminal user. The variable length command buffer is located in subpool 1. It is preceded by a four-byte header consisting of a two byte length field and a two byte offset field. The length field contains the total length of the buffer including the four bytes of header information.
4	TPLCTCB	The address of the Task Control Block (TCB) of any attached command processor. A value of zero is placed in this field when the command processor is detached. Both the TMP and the TEST command processor are responsible for maintaining this field.
4	TPLSTAI	The address of the TMP STAI exit routine specified as an operand of the ATTACH macro instruction issued by the TMP to attach the current command processor. This exit routine gains control when the attached command processor begins to terminate abnormally.
4	TPLSPLS	The address of the STAI exit parameter list specified on the ATTACH macro instruction issued by the TMP to ATTACH the current command processor.
4	TPLNECB	This four-byte field contains an Event Control Block (ECB) belonging to the TMP STAI exit routine which gets control when a command processor terminates abnormally. This ECB must be posted by either the TMP or the TEST program before the abnormally terminating command processor can resume processing. A post code of X'7F' indicates that a recovery is being attempted. Any other post code causes the ABEND to continue.
4	TPLNTCB	The address of the Task Control Block (TCB) in control when a command processor started to terminate abnormally. The TMP should set this field to zero if the TEST program is invoked by the Attention exit routine.
4	TPLMECB	This four-byte field contains an Event Control Block (ECB) used by TSO to STOP a terminal user's session. When this ECB is posted, the TEST program should return to the TMP as soon as possible. The TMP then must take the appropriate action to DETACH any subtasks before returning to the LOGON/LOGOFF Scheduler for a terminal disconnect.

Figure 5. The Test Parameter List (Part 2 of 3)

Number of Bytes	Field	Contents or Meaning
4	TPLCECB	The address of an Event Control Block (ECB) used by the MVT control program to indicate the termination of an attached task. This ECB address is the one you specify as the ECB operand on the ATTACH macro instruction issued to attach the command processor.
4	TPLIECB	The address of an Event Control Block (ECB) used by the TMP STAI exit routine to indicate that the attached command processor is terminating abnormally.
4	TPLAECB	The address of an Event Control Block (ECB) used by the TMP Attention exit routine to indicate that an attention interruption has occurred.
4	RESV	Reserved.

Figure 5. The Test Parameter List (Part 3 of 3)

When the TEST Command processor returns control to the TMP, use the PUTGET service routine to obtain a new command.

INTERCEPTING A TMP TASK ABEND

When the TMP (or any program linked to by the TMP except TEST) causes an ABEND, the TMP STAE exit gains control. The TMP specifies its own STAE exit routine by issuing the STAE macro instruction. (See Supervisor Services and Macro Instructions for a discussion of the STAE macro instruction.)

Your TMP STAE exit routine can use the contents of the STAE work area created by the STAE macro instruction to determine the type of error, the cause of the error, the PSW at the time of the ABEND, the last PSW before the program ABEND, and the contents of the program registers.

If your TMP STAE exit routine cannot correct the problem, it should use the PUTLINE macro instruction to inform the user at the terminal that a task running under the TMP TCB is terminating abnormally, take a dump of the user's region if a SYSABEND or a SYSUDUMP data set was specified in the user's LOGON cataloged procedure, clear the user's region, then load a fresh copy of the TMP, and begin processing as if the TMP had been invoked by the LOGON/LOGOFF Scheduler.

If the error persists; that is, the TMP fails again, control should pass to the PUTLINE service routine to notify the user. A log off should be forced by returning to the LOGON/LOGOFF Scheduler.

Processing an Attention Interruption

After having been attached by the LOGON/LOGOFF Scheduler, the TMP must set up its attention handling facilities during its initialization process. You can use the STAX macro instruction to pass the address of your attention handling routine to the system.

Several attention handling routines may be enqueued at any one time; that is, both the TMP and the currently active command processor may have issued STAX macro instructions. The attention exit routine specified by the last attached task is the one given control if one attention interruption occurs.

The attention handling routine you specify for the Terminal Monitor Program is given control under any of the following conditions:

1. An attention interruption is entered from the terminal while the Terminal Monitor Program is in control.
2. An attention interruption is received from the terminal while a program other than the Terminal Monitor Program is in control, but that program has not provided an attention handling routine.
3. A program other than the Terminal Monitor Program is in control. The program has provided an attention exit, but the user at the terminal has issued sufficient attention interruptions to reach the Terminal Monitor Program's attention handling routine. As an example, if a command processor that has provided an attention handling routine is in control, and a user enters two successive attention interruptions from the terminal, the Terminal Monitor Program's attention exit receives control.

You can defer attention interruption processing with the DEFER operand of the STAX macro instruction. If you do use the DEFER option, attention interruptions are queued as they are received, and are not processed until you request that the DEFER option be removed.

PARAMETERS RECEIVED BY ATTENTION HANDLING ROUTINES

When your attention exit routine is entered, the registers contain the following information:

Register	Contents
0,2-12	Irrelevant
1	The address of the Attention Exit Parameter List.
13	Save area address.
14	Return address.
15	Entry point address of the attention handling routine.

The Attention Exit Parameter List pointed to by register one, contains the address of a Terminal Attention Interruption Element (TAIE).

The parameter structure received by your attention exit routine is shown in Figure 6.

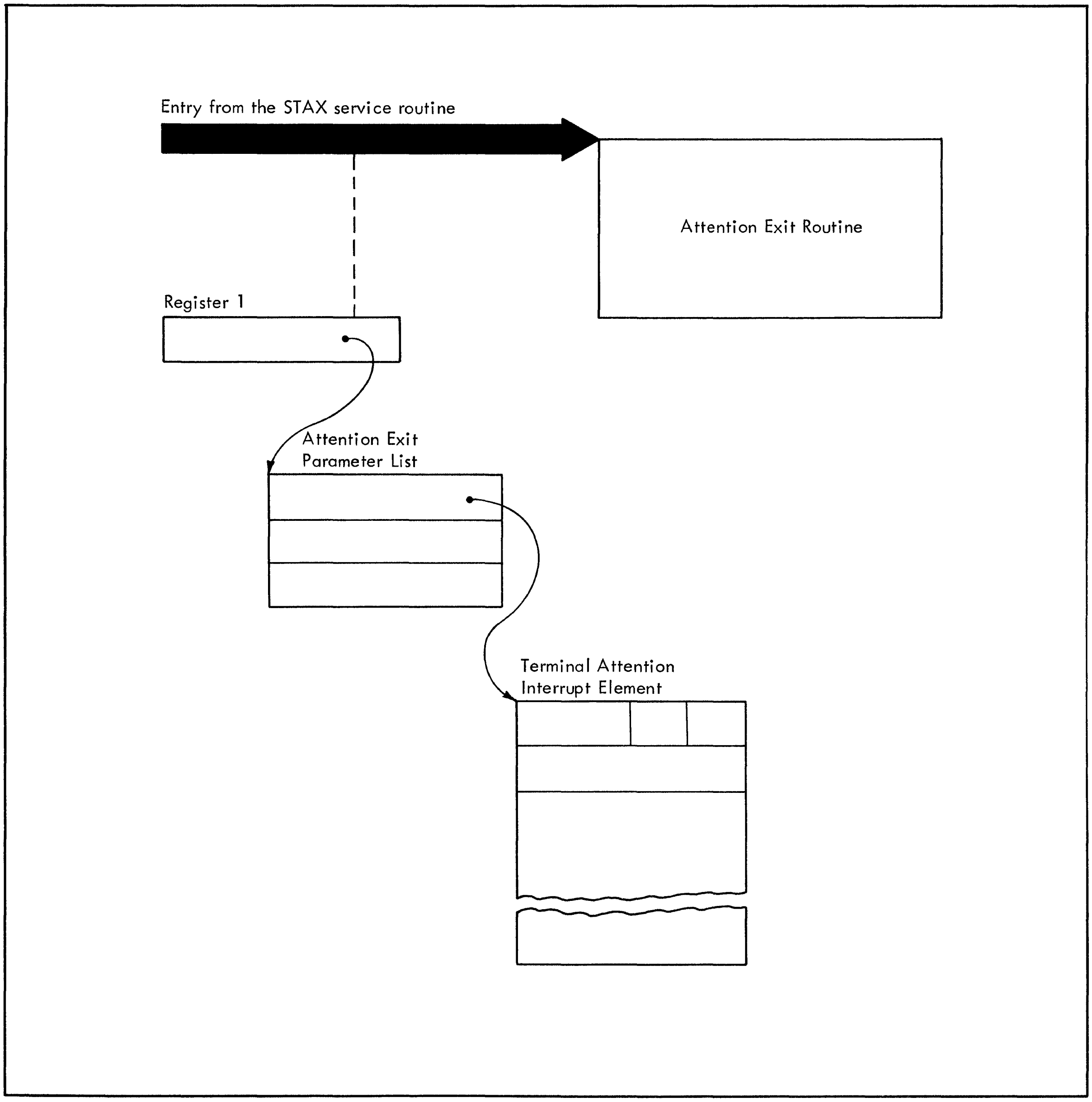


Figure 6. Parameters Passed to the Attention Exit Routine

The Attention Exit Parameter List

Figure 7 shows the format of the Attention Exit Parameter List pointed to by register one when an attention exit routine receives control.

Number of Bytes	Field	Contents or Meaning
4		The address of the Terminal Attention Interrupt Element (TAIE).
4		The address of the input buffer you specified as the IBUF operand of the STAX macro instruction. Zero if you did not include the IBUF operand in the STAX macro instruction.
4		The address of the user parameter information you specified as the USADDR operand of the STAX macro instruction. ZERO if you did not exclude the USADDR operand in the STAX macro instruction.

Figure 7. The Attention Exit Parameter List

The Terminal Attention Interrupt Element (TAIE)

The first word of the Attention Exit Parameter List contains the address of an eighteen-word Terminal Attention Interrupt Element (TAIE). Figure 8 shows the format of the TAIE.

Number of Bytes	Field	Contents or Meaning
2	TAIEMSGL	The length in bytes of the message placed into the input buffer you specified as the IBUF operand on the STAX macro instruction. Zero if you did not code the IBUF operand in the STAX macro instruction.
1	TAIETGET	The return code from the TGET macro instruction issued to get the input line from the terminal.
1		Reserved.
4	TAIEIAD	Interruption address. The right half of the interrupted PSW. The address at which the program (or a previous attention exit) was interrupted.
64	TAIERSAV	The contents of general registers, in the order 0 - 15, of the interrupted program.

Figure 8. The Terminal Attention Interrupt Element

If you did not include the IBUF and the OBUF operands in the STAX macro instruction that set up the attention handling exit, use the PUTGET macro instruction, specifying the TERM operand, to send a mode message to the terminal identifying the program that was interrupted, and to obtain a line of input from the terminal.

If you specify the OBUF operand on the STAX macro instruction without an IBUF operand, or with an IBUF length of 0, you can then use the PUTGET macro instruction, specifying the ATTN operand. This causes the PUTGET service routine to inhibit the writing of the mode message, since a message was already written to the terminal from the output buffer specified in the STAX macro instruction. The PUTGET service routine merely returns a logical line of input from the terminal.

In either of the above cases, if the user enters a question mark, the PUTGET service routine automatically causes the secondary level informational message chain (if one exists) to be written to the terminal, again puts out the mode message, and returns a line from the terminal.

If you used the IBUF operand on the STAX macro instruction, note that no logical line processing or question mark processing is performed. If the user returns a question mark, you will have to use the PUTLINE macro instruction to write the secondary level informational message chain to the terminal. Then issue a PUTGET macro instruction, specifying the TERM operand, to write a mode message to the terminal and to return a line of input from the terminal.

Use the Command Scan service routine to determine that the line of input is syntactically correct in the input buffer returned by the PUTGET service routine, or in the attention input buffer (pointed to by the second word of the Attention Exit Parameter List).

Special functions such as the TIME function should be performed immediately by the attention handling routine, and a new READY message should then be put out to the terminal, so that the terminal user may enter another command.

Any other command should be passed to the TMP mainline routine for processing as if it were a newly entered command.

Note that the TGET and TPUT buffers are flushed when an attention interruption is entered. If the user enters an attention interruption from the terminal and then enters a null line to continue processing, the contents, if any, of the TGET and TPUT buffers are lost.

Processing a STOP Command

A STOP/MODIFY ECB is created by the time sharing system and can be obtained by your TMP by use of the EXTRACT macro instruction. During TMP processing, if a STOP command is indicated by a post to the STOP ECB, return to the LOGON/LOGOFF Scheduler so that the user may be logged off the system.

Command Processors

A command processor is a problem program invoked by the TMP when a user at a terminal enters a command name.

The internal logic of the TSO-supplied command processors is described in the TSO Command Processor PLM. The command language used to request each of these command processors is described in the TSO Command Language Reference.

If you choose to write your own command processors, you should be familiar with the Service Routines described in this book.

This section discusses the relationships between the command processors and the rest of the Time Sharing Option, and provides guidelines for coding your own command processors.

The section is divided into the following topics:

- Response Time - Discusses the steps you should take to insure that your command processor does not adversely affect system response time.
- Command Processor Use of the TSO Service Routines - Briefly discusses each of the TSO Service Routines and the situations in which they should be used.
- The STAE and STAI Exit Routines - Discusses the functions your error routines should provide.
- Attention Exit Routines - Discusses the need for attention handling exits and the functions those exits should perform.
- Adding Commands to the Time Sharing Option - Discusses the methods you can use to place a newly written command processor into the Time Sharing Option.
- The HELP Data Set - Discusses the HELP data set, private HELP data sets, and the means of entering information into a HELP data set.

Programming Note: In TSO, assembly language programs may fail or cause a performance impact when they use the same job file control block (JFCB) more than once for the same data set. When the data set is opened, the Open routine fills any unspecified fields in the data control block from information in the data set control block (DSCB) and the job file control block. The Open routine then does a "reverse merge" from the data control block back into the job file control block, filling zeroed or unspecified fields in the job file control block. If the same data set is reopened by a later program by use of a new OPEN macro instruction, the Open routines will retrieve old information from the job file control block for fields not specified in the data set control block. The retrieved information could be unwanted for the new use of the data set and therefore could cause program failure or performance impact. Examples of such unwanted information include key length for BSAM and QSAM, and buffer size or channel program parameters for QSAM.

If any of your command processors specify DCB information which could cause a failure on a subsequent use of a JFCB, you can follow the procedure outlined below to inhibit the reverse merge from the DCB back into the JFCB.

1. Issue a RDJFCB macro instruction to read the JFCB into your own main storage.

2. Set the JFCBTSDM field (offset 52 decimal, 34 hex in the Job File Control Block) to X'0A' to inhibit the DCB to JFCB merge.

3. Issue an OPEN macro instruction specifying TYPE=J.

For a discussion of the RDJFCB macro instruction and the OPEN macro instruction type J, see Data Management for System Programmers.

Response Time

A Time Sharing system depends upon fast response. If you write your own command processors to run under the IBM Time Sharing Option, your command processors will directly affect system response time. The following recommendations are included to help you keep system response time to a minimum.

PROGRAM DESIGN

Any command processors you write should not modify themselves in any way during their execution. They should obtain all work areas with a GETMAIN macro instruction so that the in-line code remains unchanged. This allows the command processor to be executed from the Time Sharing Link Pack Area, and used by several tasks concurrently.

TSO provides, along with the system Link Pack Area, a Time Sharing Link Pack Area. Figure 9, a storage map of MVT with the Time Sharing Option, shows the Time Sharing Link Pack Area within the Time Sharing Control Region.

Frequently used Command Processors can be placed in the Time Sharing Link Pack Area. Placing programs in the Time Sharing Link Pack Area reduces the amount of time required to access them since they are resident in the system and need not be brought in from an external data set.

Besides reducing access time, placing command processors in the Time Sharing Link Pack Area provides two additional benefits:

1. Swap time is reduced. Swap time is the time required to move one user's programs and data from a foreground region to a swap data set and to move the next user's programs and data from a swap data set back into the foreground region.

   One of the factors that affects swap time is the amount of data that must be swapped. If the currently active command processor is executing from the Time Sharing Link Pack Area, it is not swapped when the foreground region is swapped. You therefore swap less data if your command processors are resident in the Time Sharing Link Pack Area than if they execute from the foreground region. See Time Sharing Option Guide for a discussion of the swapping algorithms used in TSO.

2. If you are running several foreground regions, your total storage requirement is less if frequently used command processors are resident in the Time Sharing Link Pack Area. Command processors resident in the Time Sharing Link Pack Area can be executed for any foreground region and need not be loaded into those regions. Your foreground regions may therefore be smaller if some of the larger command processors can be executed in the Link Pack Area.

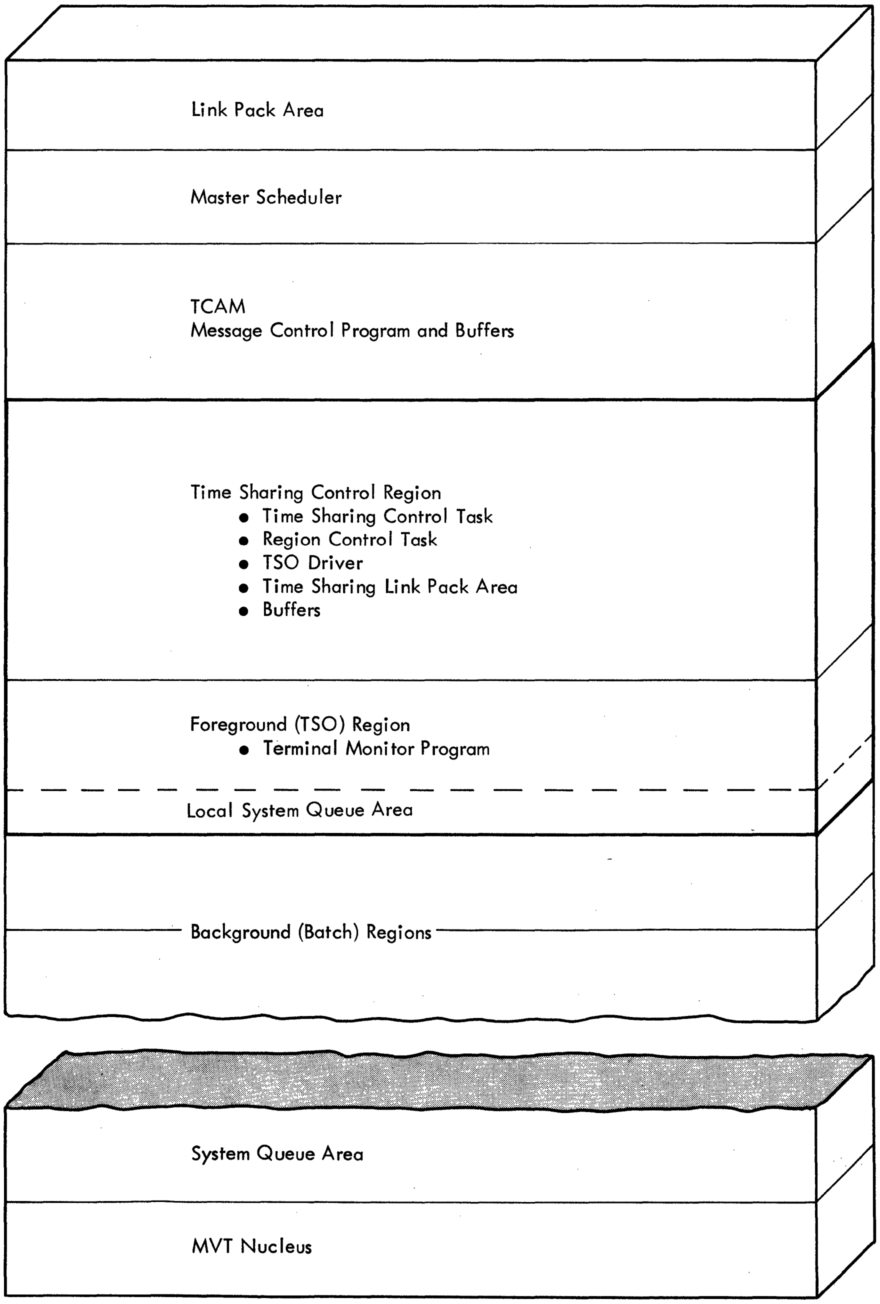

Link Pack Area
Master Scheduler
TCAM
Message Control Program and Buffers
Time Sharing Control Region
• Time Sharing Control Task
• Region Control Task
• TSO Driver
• Time Sharing Link Pack Area
• Buffers
Foreground (TSO) Region
• Terminal Monitor Program
Local System Queue Area
Background (Batch) Regions
System Queue Area
MVT Nucleus

Figure 9. Storage Map - MVT with Time Sharing Option

MODULE SIZE AND STORAGE REQUIREMENTS

Command processors that do not execute in the Time Sharing Link Pack Area should be designed to minimize the average amount of data swapped.

The more a command processor interacts with a user, the more often it must wait for input from the terminal. Since programs waiting for input from the terminal are eligible to be swapped, the probability is great that the program will be swapped. If a command processor is large and is likely to be swapped several times before it can complete its function, consider dividing it into several load modules to reduce the amount of data swapped. Keep in mind however, that additional time is required to perform a BLDL and a fetch for each of the additional load modules.

Keep in mind also that the device type used to contain the swap data sets affects the amount of time for each swap. See Storage Estimates for block sizes swapped to various device types.

Command Processor Use of the TSO Service Routines

Use the TSO-provided service routines described in this manual when coding your own command processors. Read the sections on the various service routines and macro instructions for an understanding of what services they perform and how to use them. The following topics provide information on when to use each of the service routines.

STACK SERVICE ROUTINE

Use the STACK service routine to change the source of input by adding an element to the input stack, and to reset the input stack to the terminal element originally specified by the Terminal Monitor Program.

A command processor should issue the STACK macro instruction in the following circumstances:

1. Your command processor has created a series of commands to be executed after the command processor terminates. The command processor builds an in-storage list containing the commands to be executed and uses the STACK macro instruction to place a pointer to the list on the input stack.

2. You may want to pass data from one of your command processors to another command processor. This data may be passed in storage via the input stack. Issue the STACK macro instruction to place a pointer to the in-storage data on the input stack.

3. If you write a command processor to perform functions similar to those performed by the TSO-supplied EXEC command, (that is, to execute a command procedure), issue the STACK macro instruction to place a pointer on the input stack to the command procedure to be executed.

4. Whenever one of your command processors terminates with an error condition, its error handling routine should issue the STACK macro instruction to reset the input stack.

GETLINE SERVICE ROUTINE

Your command processors should use the GETLINE service routine to obtain data. The buffer returned by GETLINE is in subpool 1 and is owned by your command processor. If your command processor issues multiple GETLINE macro instructions, it should free the buffers either with the DETACH or the FREEMAIN macro instructions.

PUTLINE SERVICE ROUTINE

Your command processors should use the PUTLINE service routine to write informational messages or data to the terminal and to chain second level informational messages. PUTLINE writes the output lines to the terminal regardless of the source of input.

PUTGET SERVICE ROUTINE

Your command processors should use the PUTGET service routine for prompting and for subcommand requests. Use the operands on the PUTGET macro instruction to specify logical line processing with editing and the WAIT option.

If the user at the terminal enters a question mark in response to a message issued with a PUTGET macro instruction, the PUTGET service routine prints the second level messages chained by previous PUTLINE macro instructions. If the user responds with a subcommand name, the second level messages are deleted and the storage they occupied is freed. See the topic headed "PUTGET Processing" for exceptions to this usual method of processing.

As with the GETLINE service routine, the buffers returned by the PUTGET service routine belong to, and should be freed by, the command processor.

DAIR SERVICE ROUTINE

Your command processors should use the DAIR service routine to allocate and free data sets and to obtain information concerning data sets. Command processors should allocate data sets by DSNAME and use the DDNAMES returned by DAIR -- if necessary passing them on to any subcommands or problem programs running under the command processor.

Whenever the user specifies a password for a data set, the password should be passed by the command processor to DAIR when allocation is requested.

Command processors that accept subcommands should use the DAIR service routine to mark any data sets allocated by the subcommands as allocatable before detaching the terminated subcommand.

COMMAND SCAN SERVICE ROUTINE

Your command processors should use the Command Scan service routine to scan for valid subcommand names. The option of checking the remainder of the input line for non-separator characters should be requested. If no additional significant characters are found in the line, the command processor subroutine need not invoke the PARSE service routine to scan the command operands (none will be present).

PARSE SERVICE ROUTINE

Your command processors and subcommand processors should use the PARSE service routine to scan the operands entered with the command or subcommand name. The PARSE service routine returns a Parameter Descriptor List to the calling routine. The Parameter Descriptor List describes the operands found in the command buffer.

Command processors and subcommand processors can specify to PARSE that validity checking exits be taken on certain types of operands. Since the PARSE Service routine checks the operands only for syntax errors, you should specify that validity checking routines be entered whenever a logical, rather than a syntactical, error might occur.

STAE/STAI Exit Routines - Intercepting an ABEND

Use the STAE and STAI exits in your command processors to keep the system operable if abnormal termination occurs. STAE/STAI exits should be used in such a way that the command processor gets control if a subcommand abnormally terminates. STAE provides the command processor with the ability to intercept an ABEND so that cleanup, bypass, and if possible, execution retry can be accomplished. (See Data Management for System Programmers, for a discussion of the STAE macro instruction. See Supervisor Services and Macro Instructions for a a discussion of the STAI operand of the ATTACH macro instruction.)

The following types of command processors should use STAE exit routines:

- All command processors that process subcommands.
- All command processors that request system resources that are not freed by ABEND or DETACH.
- Command processors that process lists, to allow processing of other elements in the list if a failure occurs while processing one element in the list.

Command processors that attach subcommands should also provide a STAI exit to intercept abnormally terminating subcommand processors.

STAE and STAI exit routines should observe the following guidelines:

1. The error handling exit routine should issue a diagnostic error message of the form:

   ```
   1st level   command name     ENDED DUE TO ERROR
               subcommand name

   2nd level   COMPLETION CODE IS xxxx
   ```

   where the name supplied in the first level message is obtained from the Environment Control Table, and the code supplied in the second level message is the completion code passed to the STAE or STAI exit from ABEND.

   The routine should issue these messages so that the original cause of abnormal termination is recorded should the error handling exit itself terminate abnormally before diagnosing the error.

When an ABEND is intercepted, the command processor STAE exit routine should determine whether retry is to be attempted. If so, the exit routine should issue the diagnostic message and return, indicating via return code that a STAE retry routine is available. If a retry is not to be attempted, the exit routine should return, indicating via return code that no retry is to be attempted. The TMP STAI exit routine will issue the diagnostic message. (For a description of the return codes and their meanings, see Supervisor Services and Macro Instructions.)

2. The STAE or STAI routine that receives control from ABEND should perform all necessary steps to provide system cleanup. This cleanup should be performed in the STAE exit routine rather than in the STAE retry routine because DETACH with the STAE=YES operand does not allow the subtask to retry from a STAE/STAI exit.

3. The error handling exit routine should attempt to retry program execution when possible. If the command processor can circumvent or correct the condition that caused the error, the error handling routine should attempt to do so. In other cases, however, RETRY has no function and the command processor STAE exit should not specify the RETRY option.

Attention Exit Routines

An attention exit routine should be provided by any command processor that accepts subcommands. Use the STAX macro instruction to specify the address of your attention handling routine. See the section headed "ATTENTION INTERRUPTION HANDLING - THE STAX SERVICE ROUTINE", for a complete discussion of the STAX macro instruction.

If you did not include the IBUF and the OBUF operands in the STAX macro instruction that set up the attention handling exit, use the PUTGET macro instruction, specifying the TERM operand, to send a mode message to the terminal identifying the program that was interrupted, and to obtain a line of input from the terminal.

If you specify the OBUF operand on the STAX macro instruction without an IBUF operand, or with an IBUF length of 0, you can then use the PUTGET macro instruction, specifying the ATTN operand. This causes the PUTGET service routine to inhibit the writing of the mode message, since a message was already written to the terminal from the output buffer specified in the STAX macro instruction. The PUTGET service routine merely returns a logical line of input from the terminal.

In either of the above cases, if the user enters a question mark, the PUTGET service routine automatically causes the secondary level informational message chain (if one exists) to be written to the terminal, again puts out the mode message, and returns a line from the terminal.

If you used the IBUF operand on the STAX macro instruction note that no logical line processing or question mark processing is performed. If the user returns a question mark, you will have to use the PUTLINE macro instruction to write the secondary level informational message chain to the terminal. Then issue a PUTGET macro instruction, specifying the TERM operand, to write a mode message to the terminal and to return a line of input from the terminal.

Whether you use the IBUF operand on the STAX macro instruction or the PUTGET macro instruction to return a line from the terminal, you can use the Command Scan service routine to determine what the user has entered.

If the user enters a null line, the attention handling routine should return to the point of interruption. Note however, that the TGET and TPUT buffers are flushed during attention interruption processing. If any data was present in these buffers, it is lost.

If a new command or subcommand is entered, the attention handling routine should:

- Reset the input stack.
- Post the command processor's Event Control Block to cause active service routines to return to the command processor.
- Exit.

Adding Commands to the Time Sharing Option

There are two methods you can use to place a new command processor into the Time Sharing Option. You can enter your newly written command processor as a member of the partitioned data set SYS1.CMDLIB, via the Linkage Editor, or you can create your own command library and concatenate it to the SYS1.CMDLIB data set. In the latter case, use the utility IEBUPDTE to create new statements in the link list (LNKLST00) in SYS1.PARMLIB. If you choose to concatenate your library to SYS1.CMDLIB, note that you cannot do it during a terminal session. You must concatenate the two libraries with data definition statements within your LOGON procedure. The DDNAME must be STEPLIB.

See Data Management Services for information on creating data sets, entering members into data sets, and concatenating data sets.

The HELP Data Set

A terminal user can enter the HELP command to retrieve information about commands and subcommands. This information is stored in a data set labeled SYS1.HELP (the HELP data set). If you add command processors to the Time Sharing Option, you should either add HELP information to the existing SYS1.HELP data set, or create your own private HELP data set.

SYS1.HELP is a cataloged, partitioned data set consisting of one member, named "COMMANDS", and individual members for each command in the system. The 'COMMANDS' member contains a list of the commands available to the user, and a brief description of each. The individual members for each command are named with the command name, and contain more specific information about the command and its subcommands. The HELP information contained within any member of the HELP data set consists of card images. The logical record length is therefore 80 characters.

Each of the SYS1.HELP members, other than the "COMMANDS" member, is divided into the following subgroups, each of which can be displayed at the terminal:

- A subcommand list - This information appears only if the command has subcommands.
- Functional description - This subgroup provides a brief description of the function of the command or subcommand.
- Syntax - This information describes the syntax of the command or subcommand.

- Operand description - This subgroup provides information on the command positional operands, followed by individual sections containing brief descriptions of each keyword and its parameters.

PRIVATE HELP DATA SETS

Private HELP data sets must be structured exactly like the SYS1.HELP data set, since both data sets are processed alike.

You may concatenate your data set to the SYS1.HELP data set (or vice versa) but the data sets must have the same attributes. Concatenated data sets are searched in the order of concatenation. If SYS1.HELP and a private HELP data set have been concatenated, the first 'COMMANDS' member encountered by the HELP processor is used as the list of available commands. Thus, if you concatenate your own HELP data set to SYS1.HELP, you should make additions to the "COMMANDS" member of SYS1.HELP.

FORMATTING THE HELP DATA SET

Use the IEBUPDTE utility program to update SYS1.HELP. Use the information described in Figure 10 to format the data set when you add to SYS1.HELP or set up your own HELP data set. The control characters, beginning in card column 1, divide the data set into the subgroups previously described, and thereby permit the HELP command processor to select message text according to the operands supplied on the terminal user's HELP command. (See TSO Command Language Reference for a discussion of the HELP command.)

Control Character	Purpose of Data Card
)S	This card indicates that a list of commands or subcommands follows.
)F	This card indicates that the functional discussion of the command or subcommand follows.
)X	This card indicates that the syntax description of the command or subcommand follows.
)O	This card indicates that the command operands and their descriptions follow. Positional operands must follow immediately after the ")O" control card and before the ")) keyword" control cards.
))keyword	This card indicates that information follows describing the named keyword. One of these control cards must be present for each KEYWORD operand within the command. Each card contains the name of the keyword it describes.
=subcommandname	This card indicates that information follows concerning the subcommand named after the equal sign. One of these cards is required for each subcommand accepted by the command being described. Note that this card merely names the subcommand; it does not describe it. Describe the subcommand in the same manner you would describe a command. If the subcommand has an alias name, you may include the alias name on the control card, i.e. =subcommandname=subcommandalias. Note that no blanks may appear between the subcommand name and the alias.

Figure 10. Cards Used to Format a HELP Data Set

All data cards, except the =subcommandname card, can contain additional information. If you include additional information on the cards, the control characters)S,)F,)X, and)O must be followed by at least one blank, and the control character))keyword by at least one blank or a left parenthesis. Use the left parenthesis when the keyword you are describing is followed by operands enclosed in parentheses. See Figure 9 for an example of this.

The only restrictions on data cards are that columns 72-80 are reserved for sequence numbers, and column one must contain either a right parenthesis or an equal sign.

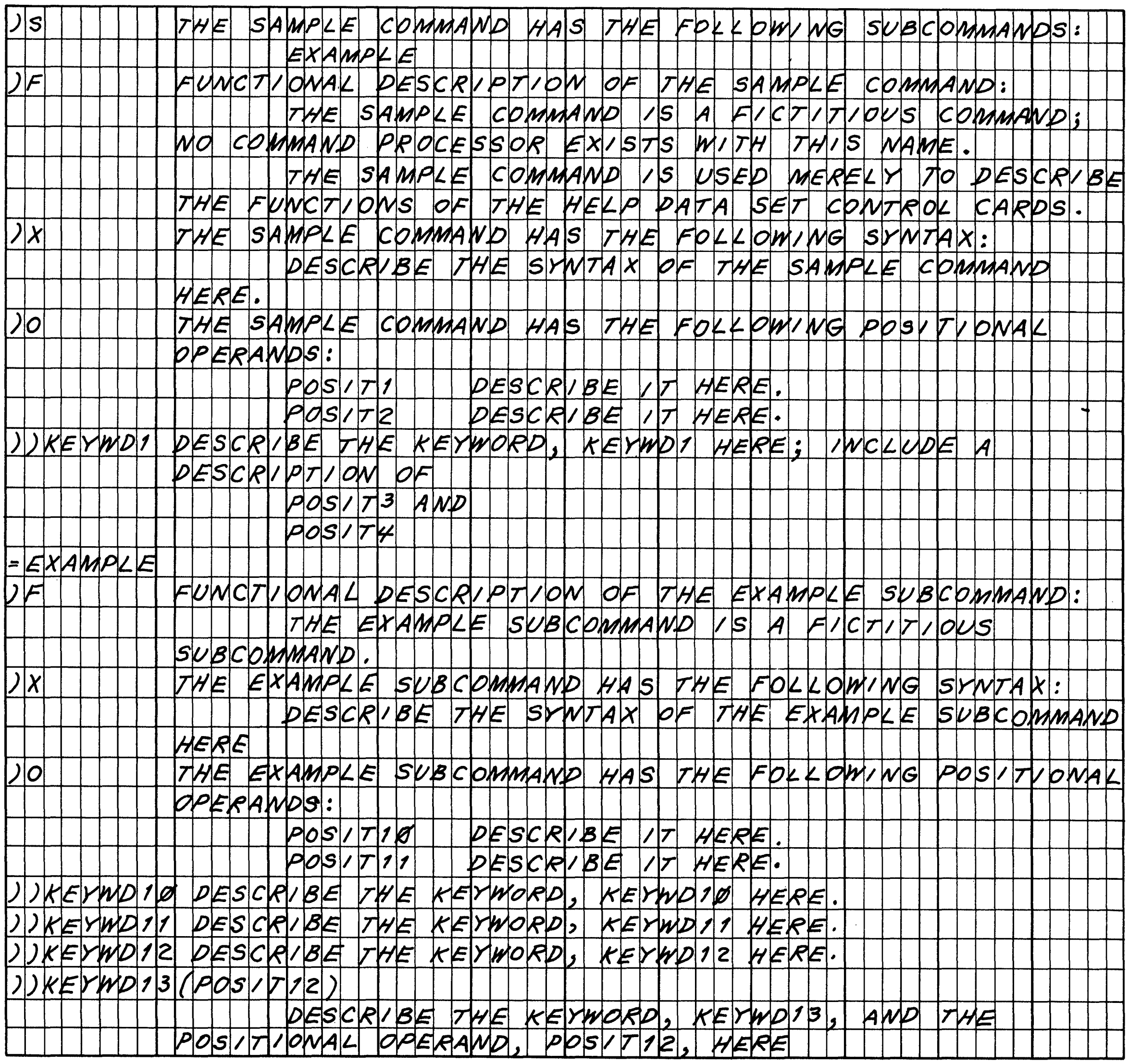
For example, information concerning the sample command shown below could be formatted for entry into the HELP data set (or your own private help data set) using the cards shown in Figure 11. The fictitious SAMPLE command could have the following format:

SAMPLE	posit1 [,(posit2)][KEYWD1 [(posit3,posit4)]]

The SAMPLE command has one subcommand, the EXAMPLE subcommand. The fictitious EXAMPLE subcommand has the following format:

```
┌──────────────┬────────────────────────────────────────────────────────┐
│              │                      ┌KEYWD10┐                          │
│   EXAMPLE    │    posit10,posit11   │KEYWD11│ [KEYWD13(posit12)]       │
│              │                      └KEYWD12┘                          │
└──────────────┴────────────────────────────────────────────────────────┘
```

Figure 11 shows data cards that would present and format information about the SAMPLE command for inclusion in the HELP data set.



```
)S       THE SAMPLE COMMAND HAS THE FOLLOWING SUBCOMMANDS:
               EXAMPLE
)F       FUNCTIONAL DESCRIPTION OF THE SAMPLE COMMAND:
               THE SAMPLE COMMAND IS A FICTITIOUS COMMAND;
         NO COMMAND PROCESSOR EXISTS WITH THIS NAME.
               THE SAMPLE COMMAND IS USED MERELY TO DESCRIBE
         THE FUNCTIONS OF THE HELP DATA SET CONTROL CARDS.
)X       THE SAMPLE COMMAND HAS THE FOLLOWING SYNTAX:
               DESCRIBE THE SYNTAX OF THE SAMPLE COMMAND
         HERE.
)O       THE SAMPLE COMMAND HAS THE FOLLOWING POSITIONAL
         OPERANDS:
               POSIT1    DESCRIBE IT HERE.
               POSIT2    DESCRIBE IT HERE.
))KEYWD1 DESCRIBE THE KEYWORD, KEYWD1 HERE; INCLUDE A
         DESCRIPTION OF
               POSIT3 AND
               POSIT4
=EXAMPLE
)F       FUNCTIONAL DESCRIPTION OF THE EXAMPLE SUBCOMMAND:
               THE EXAMPLE SUBCOMMAND IS A FICTITIOUS
         SUBCOMMAND.
)X       THE EXAMPLE SUBCOMMAND HAS THE FOLLOWING SYNTAX:
               DESCRIBE THE SYNTAX OF THE EXAMPLE SUBCOMMAND
         HERE
)O       THE EXAMPLE SUBCOMMAND HAS THE FOLLOWING POSITIONAL
         OPERANDS:
               POSIT10   DESCRIBE IT HERE.
               POSIT11   DESCRIBE IT HERE.
))KEYWD10 DESCRIBE THE KEYWORD, KEYWD10 HERE.
))KEYWD11 DESCRIBE THE KEYWORD, KEYWD11 HERE.
))KEYWD12 DESCRIBE THE KEYWORD, KEYWD12 HERE.
))KEYWD13(POSIT12)
               DESCRIBE THE KEYWORD, KEYWD13, AND THE
         POSITIONAL OPERAND, POSIT12, HERE
```

Figure 11. Coding Example -- Including the SAMPLE Command in the HELP Data Set

Message Handling

TSO messages are divided into three classes:

- Prompting messages
- Mode messages
- Informational messages

Prompting messages are of the form ENTER... or REENTER..., and require a response from the user. Prompting messages should be initiated by the PARSE service routine, using the text supplied by the command processor as the PROMPT operand of the IKJPOSIT, IKJTERM, IKJOPER, IKJRSVWD or IKJIDENT parse macro instructions. See the section headed "Using the PARSE Service routine (IKJPARS)" for a discussion of the PROMPT operand on the these macro instructions.

Mode messages are the READY message sent by the Terminal Monitor Program, and any other similar messages sent by command processors, such as the EDIT mode message sent by the EDIT command processor. They inform the user which command component is in control and let him know that the system is waiting for him to enter a new command or subcommand.

Informational messages include all others; that is, any message which does not require an immediate response from the user.

Prompting and Mode messages should be displayed using the PUTGET service routine. Informational messages should be displayed using the PUTLINE service routine.

Message Levels

Messages usually should have associated with them other messages that more fully explain the initial message. These messages, called second level messages, third level messages, and so forth, are displayed only if the user specifically requests them by entering a question mark "?".

Prompting messages may have any number of additional levels. The second level is displayed if the user enters a question mark in response to the initial message. The last level is displayed if the user enters a question mark in response to the next to the last message. If the user at the terminal enters a question mark after all levels have been displayed, PUTGET displays the message "NO INFORMATION AVAILABLE". Pass your second level prompting messages to the PARSE service routine by coding them as the HELP operand in the IKJPOSIT, IKJTERM, IKJOPER, IKJRSVWD and IKJIDENT parse macro instructions.

An informational message can have only one second level message associated with it. Since many informational messages might be displayed at the terminal before a question mark is returned from the terminal, PUTLINE moves all second level informational messages to subpool 78 and chains them off the Environment Control Table. This chain exists from one PUTGET for a mode message to the next. In other words, whenever the user can enter a new command or subcommand, he can enter a question mark instead, requesting all the second level messages for informational messages issued during execution of the previous command or subcommand. If he does not enter a question mark, PUTGET deletes the second level messages and frees the main storage they occupy.

Mode messages cannot have second level messages, since a question mark entered in response to a mode message is defined as a request for the second levels of previous informational messages. Your program should request all commands or subcommands by issuing a mode message with the PUTGET service routine so that second level informational messages may be properly handled.

Effects of the Input Source on Message Processing

Message handling is considerably affected if the input source designated by the input stack is an in-storage list rather than a terminal. See the explanation of the STACK macro instruction for a discussion of in-storage lists. In-storage lists may be either procedures or source lists.

If a procedure is being executed, the PUTGET Service Routine does not display prompting messages, but returns an error code (12) in register 15. If the PARSE Service Routine issued the PUTGET macro instruction, PARSE issues an informational message to the terminal, and returns an error code to its caller, (code 4). The command processor should reset the input stack and terminate. If a command processor issued the PUTGET macro instruction, the command processor should use the PUTLINE service routine to write an appropriate informational message to the terminal prior to terminating.

If a source in-storage list is being processed, prompt messages are displayed to, and responses read from, the terminal by the PUTGET Service Routine.

If the user at the terminal has specified the PAUSE operand on the PROFILE command, PUTGET issues a special message, "PAUSE", if all of these three conditions exist:

(1) A mode message is to be written out.
(2) Second level messages exist.
(3) An in-storage list is being processed.

The user may enter either a question mark or a null line. If he enters a question mark, the chain of second level messages is written to the terminal. If he enters a null line, control returns to the executing command processor. In either case, the next line from the in-storage list is returned to the command processor.

A special situation arises if: an in-storage list is being processed, second level messages are chained, and the user has specified NOPAUSE as an operand of the PROFILE command. Normally, if a subcommand encounters an error situation, it issues an information message and returns. The command processor then uses the PUTGET service routine to issue a mode message on the assumption that the user can take corrective action with other subcommands. When processing from an in-storage list, this is not true. If NOPAUSE was specified, PUTGET merely returns an error code (12) to the calling routine. In most cases, the command processor should reset the input stack and terminate. If the message producing the second level message was purely informational and does not require corrective action, the command processor may set the ECTMSGF flag in the Environment Control Table to delete the second level message, and reissue the PUTGET macro instruction to continue.

Attention Interruption Handling -- The STAX Service Routine

The STAX service routine creates the control blocks and queues necessary for the system to recognize and schedule user exits due to attention interruptions. Your Terminal Monitor Program, your command processors, or the problem program provide the address of an attention exit to the STAX service routine by issuing the STAX macro instruction. You should provide attention exit routines within the Terminal Monitor Program and any command processors that accept subcommands.

When the attention exit routine is entered, all the subtasks of the interrupted task are stopped. If the subtasks must be dispatchable during attention exit processing, it is the user's responsibility to start the subtasks again by issuing the STATUS macro instruction.

Note that when an attention interruption is entered from the terminal, the TGET and TPUT buffers are flushed. Any data contained in these buffers is lost. If the user then attempts to continue processing from the point of interruption, he may have lost an input or an output record, or an output message from the system.

Specifying a Terminal Attention Exit - The STAX Macro Instruction

The STAX macro instruction is used to specify the address of an attention exit routine that is to be given control asynchronously when the attention key is struck or when a simulated attention is specified. (See the STATTN macro instruction for a description of the simulated attention function.)

The STAX macro instruction can also be used to cancel the last attention exit routine established by the task. To do this, specify the STAX macro instruction without the exit address and DEFER operands.

The STAX macro instruction is used only in a time sharing environment. It is ignored in a system that includes the time sharing option (TSO) if TSO is not active when the macro instruction is issued. In addition, attention exits can be established only for time sharing tasks operating in the foreground.

Issue the STAX macro instruction to provide the information required by the STAX service routine. The STAX macro instruction has a list and an execute form.

The List form of the STAX macro instruction (MF=L) generates a STAX Parameter List. The EXECUTE form of the STAX macro instruction (MF=E,(address)) completes or modifies that list and passes its address to the STAX service routine only if you specify either or both on exit address or deferral action.

Figure 12 shows the format of the list and the execute forms of the STAX macro instruction; each of the operands is explained following the figure. Appendix B describes the notation used to define macro instructions.

```
[symbol]   STAX   ┌exit address[,OBUF=(output buffer address,size)]┐
                  │             [,IBUF=(input buffer address,size)] │
                  │             [,USADDR=user address]              │
                  │             [,REPLACE={YES}]                    │
                  └                        {NO }                    ┘
                                [,DEFER={YES}]
                                        {NO }

                                {,MF=L             }
                                {,MF=(E,(address)) }
```

Figure 12. The STAX Macro Instruction -- List and Execute Forms

exit address

Specify the entry point of the routine to be given control when an attention interruption is received. You must specify the exit address in both the list and the execute forms of the STAX macro instruction when you are establishing an attention interruption handling exit.

You need not specify an exit address if you are using the DEFER operand as long as you code no other operands (except the MF operand). If you exclude the exit address and the DEFER operand and code other operands, the STAX service routine merely cancels the previous attention exit established by the task issuing this STAX macro instruction. If you exclude the exit address and code the DEFER operand, with or without other operands, only the deferral status is changed.

OBUF=(output buffer address,output buffer size)

Output buffer address - Supply the address of a buffer you have obtained and initiated with the message to be put out to the terminal user who entered the attention interruption. This message may identify the exit routine and request information from the terminal user. It is sent to the terminal before the attention exit routine is given control.

Output buffer size - Indicate the number of characters in the output buffer. The size may range from 0 to 32,767 (2^{15}-1 inclusive).

IBUF=(input buffer address,input buffer size)

Input buffer address - Supply the address of a buffer you have obtained to receive responses from the terminal user. The attention exit routine is not given control until the STAX service routine has placed the terminal user's reply into this buffer.

Input buffer size - Indicate the number of bytes you have provided as an input buffer. The size may range from 0 to 32,767 (2^{15}-1 inclusive).

USADDR=(user address)

The user address is a pointer to any information you want passed to your attention handling exit routine when it is given control.

REPLACE=YES or NO

YES indicates that the attention exit specified by this STAX macro instruction replaces any attention exit specified by a STAX macro instruction previously issued by this task. YES is the default value. REPLACE implies add, if no previous attention exit has been established.

NO indicates that this attention exit is an additional exit to any that have been previously established for this task.

DEFER=YES or NO

The DEFER operand is optional. If the DEFER operand is coded in the STAX macro instruction, the option you request (YES or NO) applies to all tasks within the task chain in which the macro instruction was issued. Any task may issue the STAX macro instruction to specify DEFER=YES or NO; the issuing task need not itself have provided an attention exit routine. If the DEFER operand is not coded in the macro instruction, no action is taken by the STAX service routine regarding the deferral of attention exits.

YES indicates that any attention interruptions received are to be queued and are not to be processed until another STAX macro instruction is executed specifying DEFER=NO, or until the program that issued the STAX with the DEFER=YES terminates.

NO indicates that the defer option is being cancelled. Any attention interruptions received while the defer option was in effect are to be processed in a first-in, first-out manner. If the DEFER operand is omitted, the control program leaves the deferral status unchanged.

Be aware that if a program issues a STAX macro instruction specifying DEFER=YES, it can get into a situation where an attention interruption cannot be received from the terminal. If your program enters a loop or an unending wait before it has issued a STAX macro instruction specifying DEFER=NO, you cannot regain control at the terminal by entering an attention interruption.

You need not specify an exit address in a STAX macro instruction issued only to change deferral status. Note, however, that a STAX macro instruction entered without an exit address is considered to be a STAX cancel if any operands are included other than DEFER and MF.

When control is passed to another routine with an XCTL macro instruction, the routine receiving control assumes the deferral status of the routine that issued the XCTL macro instruction.

When control is passed to another routine with a LOAD or CALL macro instruction, the routine receiving control also receives the deferral status of the routine that passed control. If the routine receiving control changes deferral status, it remains changed when control is returned.

When control is passed to another routine with a LINK macro instruction, the routine receiving control maintains its own deferral status: It does not receive a deferral status when it receives control nor does it return a deferral status when it returns control.

MF=L

This specifies the list form of the STAX macro instruction. It generates a STAX Parameter List.

MF=(E,(address))

This specifies the execute form of the STAX macro instruction. It completes or modifies the STAX Parameter List and passes the address of the Parameter List to the STAX service routine. Place the address of the STAX Parameter list (the address of the list form of the STAX macro instruction) into a register and specify that register number within parentheses.

You can place each of the required address and size parameters into registers and specify those registers, within parentheses, in the STAX macro instruction. Figure 13 shows how an execute form of the STAX macro instruction may look if you load all the required parameters into registers.

If an attention exit is specified in the list form, but no attention exit is specified in the execute form, then this is considered a cancel operation.

```
     STAX   (2),IBUF=((3),(4)),OBUF=((5),(6)),USADDR=(7),MF=(E,(1))
```

Figure 13. Using Registers in the STAX Macro Instruction

The STAX Parameter List

When the list form of the STAX macro instruction expands, it builds a five word STAX Parameter List. The list form of the macro instruction initializes this STAX Parameter List according to the operands you have coded.

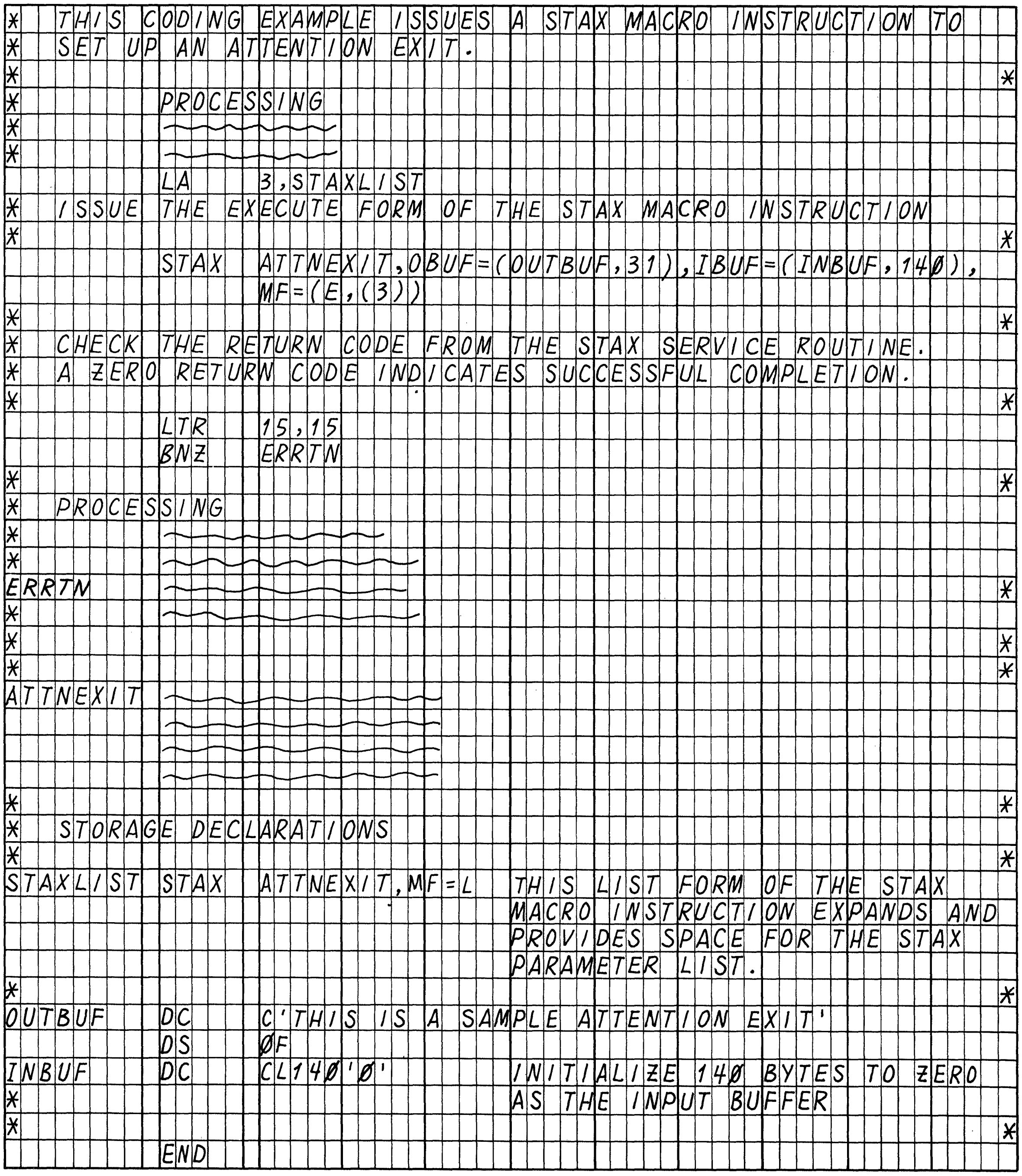
The execute form of the STAX macro instruction modifies the STAX Parameter List and passes its address to the STAX service routine. Figure 14 describes the contents of the STAX Parameter List.

Number of Bytes	Field	Contents or Meaning
4	STXEXIT	The address of the attention exit routine to receive control in response to an attention interruption. This is the address you supplied as the exit address operand on the STAX macro instruction.
2	STXISIZ	Contains a binary number representing the size of the input buffer you provided as the IBUF operand on the STAX macro instruction. The maximum buffer size is 4095 bytes.
2	STXOSIZ	Contains a binary number representing the size of the output buffer you provided as the OBUF operand on the STAX macro instruction. The maximum buffer size is 4095 bytes.
4	STXOBUF	Contains the address of the output buffer you provided as the OBUF operand on the STAX macro instruction.
4	STXIBUF	Contains the address of the input buffer you provided as the IBUF operand on the STAX macro instruction.
1	STXOPTS	STAX option flags.
	.0..	REPLACE=YES
	.1..	REPLACE=NO
	..1.	Defer attention interruption processing, that is DEFER=YES.
	...1	Cancel the deferral of attention interruption processing, that is DEFER=NO.
	x... xxxx	Reserved bits.
3	STXUSER	Contains the address of the parameters you want passed to your attention handling exit routine when it is given control. This is the address you supplied as the USADDR operand on the STAX macro instruction.

Figure 14. The STAX Parameter List

Coding Example of the STAX Macro Instruction

The coding example shown in Figure 15 uses the list and the execute forms of the STAX macro instruction to set up an attention handling exit. The OBUF operand provides a message to be written to the terminal when the attention interruption is received, and the IBUF operand provides space for an input buffer. This example does not code the REPLACE operand in the macro instruction; YES is the default value. The attention handling exit established by this execution of the STAX macro instruction replaces the previous attention handling exit established for this task.



```
*  THIS CODING EXAMPLE ISSUES A STAX MACRO INSTRUCTION TO
*  SET UP AN ATTENTION EXIT.
*                                                                      *
*        PROCESSING
*        ~~~~~~~~~~
*        ~~~~~~~~~~
         LA    3,STAXLIST
*   ISSUE THE EXECUTE FORM OF THE STAX MACRO INSTRUCTION
*                                                                      *
         STAX  ATTNEXIT,OBUF=(OUTBUF,31),IBUF=(INBUF,14Ø),
               MF=(E,(3))
*                                                                      *
*  CHECK THE RETURN CODE FROM THE STAX SERVICE ROUTINE.
*  A ZERO RETURN CODE INDICATES SUCCESSFUL COMPLETION.
*                                                                      *
         LTR   15,15
         BNZ   ERRTN
*                                                                      *
*  PROCESSING
*        ~~~~~~~~~~~~~
*        ~~~~~~~~~~~~~~~
ERRTN    ~~~~~~~~~~~~~~~                                               *
*        ~~~~~~~~~~~~~~~
*                                                                      *
*                                                                      *
ATTNEXIT ~~~~~~~~~~~~~~~~~
         ~~~~~~~~~~~~~~~~~
         ~~~~~~~~~~~~~~~~~
         ~~~~~~~~~~~~~~~~~
*                                                                      *
*  STORAGE DECLARATIONS
*                                                                      *
STAXLIST STAX  ATTNEXIT,MF=L  THIS LIST FORM OF THE STAX
                              MACRO INSTRUCTION EXPANDS AND
                              PROVIDES SPACE FOR THE STAX
                              PARAMETER LIST.
*                                                                      *
OUTBUF   DC    C'THIS IS A SAMPLE ATTENTION EXIT'
         DS    ØF
INBUF    DC    CL14Ø'Ø'       INITIALIZE 14Ø BYTES TO ZERO
*                             AS THE INPUT BUFFER
*                                                                      *
         END
```

Figure 15. Coding Example -- STAX Macro Instruction

Return Codes From the STAX Service Routine

When the STAX service routine returns control to the program that issued the STAX macro instruction, register 15 contains one of the following return codes:

CODE	MEANING
0	The STAX service routine successfully completed the function you requested. That is, it queued the attention exit you passed it, or it cancelled an existing attention exit.
4	Deferral of attention exits has already been requested and is presently in effect. Any other operands you specified in the STAX macro instruction have been processed successfully.
8	Invalid parameter passed to the STAX service routine; your STAX macro instruction was ignored.

Dynamic Allocation of Data Sets -- The Dynamic Allocation Interface Routine (DAIR)

Dynamic Allocation routines allocate, free, concatenate, and deconcatenate data sets dynamically; that is, during problem program execution. With the Time Sharing Option, dynamic allocation permits the Terminal Monitor Program, Command Processors, and other problem programs executing in the foreground region to allocate data sets after LOGON and free them before LOGOFF.

For a complete discussion of Dynamic Allocation, see the TSO Terminal Monitor Program and Service Routines PLM.

The Dynamic Allocation routines may be accessed from a TSO foreground region only through the Dynamic Allocation Interface Routine (DAIR). In general, DAIR obtains information about a data set and, if necessary, invokes Dynamic Allocation routines to perform the requested function.

You can use DAIR to perform the following functions:

- Obtain the current status of a data set.
- Allocate a data set (See note).
- Free a data set.
- Concatenate data sets.
- Deconcatenate data sets.
- Build a list of attributes (DCB parameters) to be assigned to data sets.
- Delete a list of attributes.

Note:
If you wish to allocate a data set to a direct access device, the device must be available. To be available, the device must be:

- On line
- Ready
- Shareable.

Further conditions:

- An offline or unload condition must not be pending.
- There must be no outstanding MOUNT message.
- The volume attributes must have been defined.

Using DAIR

Enter the DAIR service routine with a LINK macro instruction to entry point IKJEFD00 in load module IKJEFD00. The control block structure required by the DAIR service routine is shown in Figure 16. Note that the DAIR Parameter Block (DAPB) is a variable-size block; the block size depends upon the function requested by the calling routine. That function is indicated to the DAIR service routine by the code in the first two bytes of the DAIR Parameter Block.

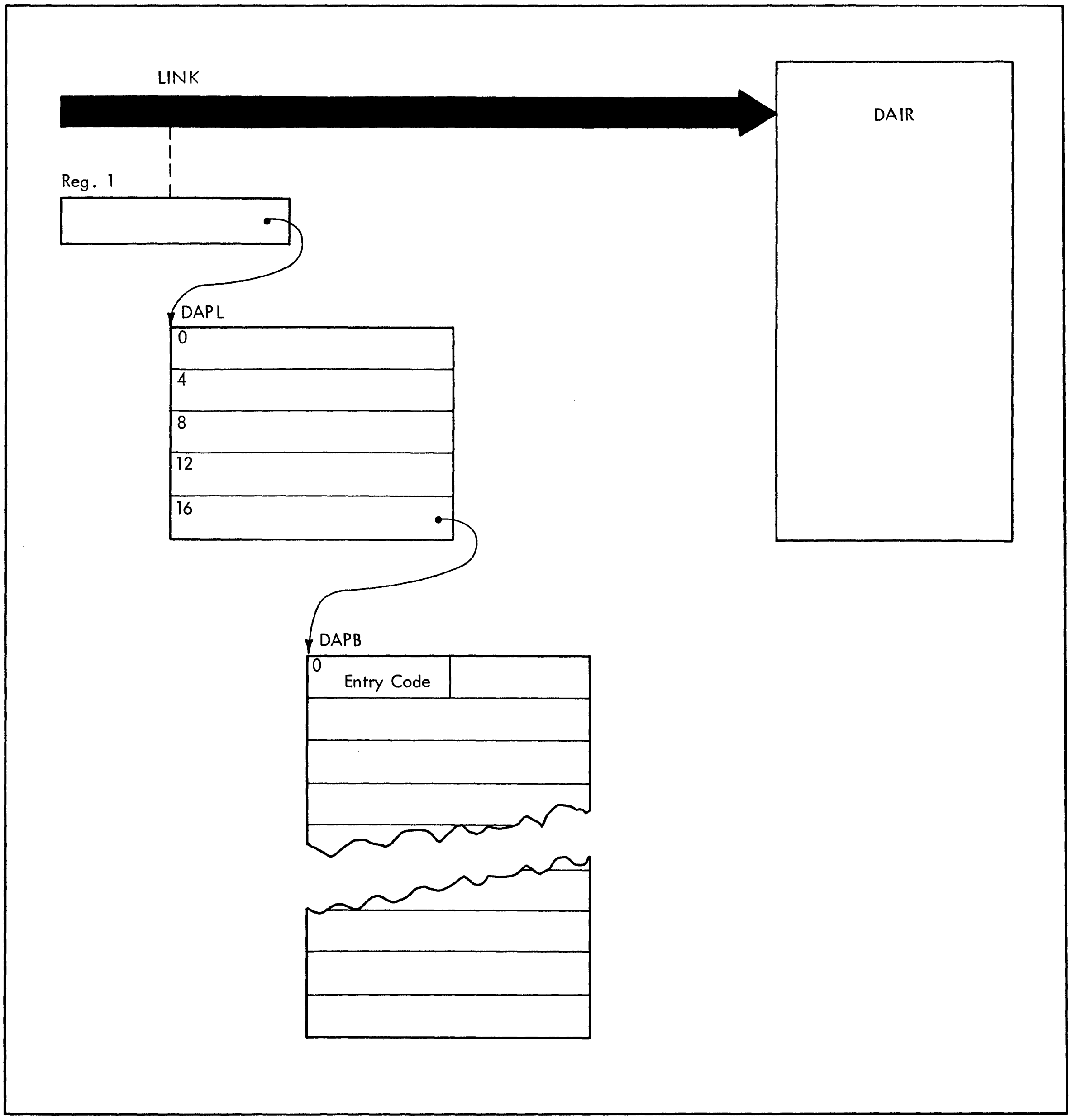


Figure 16. Control Blocks Passed to DAIR

THE DAIR PARAMETER LIST (DAPL)

At entry to DAIR, register 1 must point to a DAIR Parameter List that you have built. Figure 17 shows the format of the DAPL. The addresses of the user profile table, environment control table, and protected step control block may be obtained from the command processor parameter list (CPPL) that the TMP passes to your command processor (See Figure 33). Additional information on the address and creation of the user profile table, environment control table, and protected step control block is shown in Figure 5 (the Test Parameter List).

Number of Bytes	Field	Contents or Meaning
4	DAPLUPT	The address of the User Profile Table.
4	DAPLECT	The address of the Environment Control Table.
4	DAPLECB	The address of the calling program's Event Control Block. The ECB is one word of storage declared and initialized to zero by the calling routine.
4	DAPLPSCB	The address of the Protected Step Control Block.
4	DAPLDAPB	The address of the DAIR Parameter Block, created by the calling routine.

Figure 17. Format of the DAIR Parameter List (DAPL)

THE DAIR PARAMETER BLOCK (DAPB)

The fifth word of the DAIR Parameter List must contain a pointer to a DAIR Parameter Block built by the calling routine.

It is a variable-size parameter block that contains, in the first two bytes, an entry code that defines the operation requested by the calling routine. The remaining bytes contain other information required by DAIR to perform the requested function. Figure 18 is a list of the DAIR entry codes and the functions requested by those codes.

Entry Code	Function Performed by DAIR
X'00'	Search the DSE for information about a data set by DDNAME or DSNAME.
X'04'	Search the DSE for information about a data set by DSNAME. If not found, search the system catalog.
X'08'	Allocate a data set by DSNAME.
X'0C'	Concatenate data sets by DDNAME.
X'10'	Deconcatenate data sets by DDNAME.
X'14'	Search the system catalog for all qualifiers for a DSNAME. (The DSNAME alone represents an unqualified index entry.)
X'18'	Free a data set.
X'1C'	Allocate a DDNAME to a terminal.
X'24'	Allocate a data set by DDNAME or DSNAME.
X'28'	Perform a list of operations.
X'2C'	Mark data sets as not in use.
X'30'	Allocate a SYSOUT data set.
X'34'	Build or delete an attribute control block (ATRCB).

Figure 18. DAIR Entry Codes and Their Functions

The DAIR Parameter Blocks have the formats shown in the following tables. The formats of the blocks depend upon the function requested by the calling routine. The function is indicated by the entry code in the first two bytes of the DAIR Parameter Block.

Code X'00' - Search the DSE for a Data Set Name

Build the DAIR Parameter Block shown in Figure 19 to request that DAIR search the Data Set Extension for a fully qualified data set name.

Number of Bytes	Field	Contents or Meaning
2	DA00CD	Entry code X'0000'
2	DA00FLG	A flag field set by DAIR before returning to the calling routine. The flags have the following meaning:
	Byte 1	
	0000	Reserved. Set to zero.
	.... 1...	DSNAME or DDNAME is permanently allocated.
	.... .1..	DDNAME is a DYNAM.
	.... ..1.	The DSNAME is currently allocated; it appears in the DSE.
	1	The DDNAME is currently allocated to the terminal.
	Byte 2	
	0000 0000	Reserved. Set to zero.
4	DA00PDSN	Place in this field the address of the DSNAME buffer. The DSNAME buffer is a 46 byte field with the following format: The first two bytes contain the length, in bytes of the DSNAME; The next 44 bytes contain the DSNAME, left justified, and padded to the right with blanks.
8	DA00DDN	Contains the DDNAME for the requested data set. If a DSNAME is present, the DAIR service routine ignores the contents of this field.
1	DA00CTL	A flag field:
	00.0 0000	Reserved bits. Set to zero.
	..1.	Prefix userid to DSNAME.
2		Reserved bytes; set these bytes to zero.
1	DA00DSO	A flag field: These flags describe the organization of the data. They are returned to the calling routine by the DAIR service routine.
	1...	Indexed Sequential (IS).
	.1..	Physical Sequential (PS).
	..1.	Direct Organization (DO).
	...0 00..	Reserved bits. Set to zero.
	.... ..1.	Partitioned Organization (PO).
	1	Unmoveable.

Figure 19. DAIR Parameter Block -- Entry Code X'00'

After DAIR searches the Data Set Entry for the fully qualified data set name, register 15 contains one of the following DAIR return codes;

0, 4

See the topic "Return Codes from DAIR" for return code meanings.

Code X'04' - Search the DSE and the System Catalog for Data Set Name

Build the DAIR Parameter Block shown in Figure 20 to request that DAIR search the Data Set Extension for a fully qualified data set name. If the data set is not found in the DSE, DAIR also searches the system catalog.

Number of Bytes	Field	Contents or Meaning
2	DA04CD	Entry code X'0004'.
2	DA04FLG	A flag field set by DAIR before returning to the calling routine. The flags have the following meaning:
	Byte 1	
	0000 0..0	Reserved bits. Set to zero.
	.... .1..	DAIR found the DSNAME in the catalog.
	.... ..1.	The DSNAME is currently allocated in the Data Set Extension.
	Byte 2	
	0000 0000	Reserved. Set to zero.
2		Reserved bytes. Set to zero.
2	DA04CTRC	These two bytes will contain an error code from the catalog management routines if an error was encountered by catalog management.
4	DA04PDSN	Place in this field the address of the DSNAME buffer. The DSNAME buffer is a 46-byte field with the following format: The first two bytes contain the length, in bytes, of the DSNAME; The next 44 bytes contain the DSNAME, left justified, and padded to the right with blanks.
1	DA04CTL	A flag field:
	00.0 0000	Reserved bits. Set to zero.
	..1.	Prefix userid to DSNAME.
2		Reserved bytes; set these bytes to zero.
1	DA04DSO	A flag field. These flags are set by the DAIR Service routine; they describe the organization of the data set to the calling routine. These flags are returned only if the data set is currently allocated in the DSE.
	1...	Indexed Sequential (IS).
	.1..	Physical Sequential (PS).
	..1.	Direct Organization (DO).
	...0 00..	Reserved bits. Set to zero.
	.... ..1.	Partitioned Organization (PO).
	1	Unmoveable.

Figure 20. DAIR Parameter Block -- Entry Code X'04'

After attempting the requested function, DAIR returns one of the following codes in register 15:

0, 4, 8

See the topic "Return Codes from DAIR" for return code meanings.

Code X'08' - Allocate a Data Set by DSNAME

Build the DAIR Parameter Block shown in Figure 21 to request that DAIR allocate a data set. The exact action taken by DAIR depends upon the presence of the optional fields and the setting of bits in the control byte.

If the data set is new and you specify DSNAME, (NEW, CATLG) DAIR catalogs the data set upon successful allocation. If the catalog attempt is unsuccessful, DAIR frees the data set.

If the proper indices are not present, the catalog macro CATBX attempts to build indices for DAIR.

DAIR searches the Data Set Extension in a manner that depends upon the information you supply in the DAIR Parameter Block. DAIR may search on DSNAME alone, DSNAME and DDNAME if both are specified, DDNAME alone if no DSNAME is specified, or DAIR may search for any available entry. If DAIR searches for an available entry in the DSE, the order of selection is:

1. A DYNAM entry.
2. A data set that is currently allocated but not in use and not permanently allocated.
3. A concatenated data set not in use and not permanently allocated.

If neither of the above types of DSE entries can be found, allocation cannot take place. If an entry can be found from number 2 (above) DAIR frees the entry and uses it for the requested allocation. If DAIR can find an entry from number 3 (above), it deconcatenates, then frees the data set.

To allocate a utility data set use DAIR code X'08' and use a DSNAME of the form &name. If the &name is found allocated in the Data Set Extension, that data set is used. If the &name is not found, DAIR attempts to allocate a data set.

To supply DCB information, provide the name of an attribute list that has been defined previously by a X'34' entry into DAIR.

The DAIR Parameter Block required for entry code X'08' has the following format:

Number of Bytes	Field	Contents or Meaning
2	DA08CD	Entry code X'0008'.
2	DA08FLG	A flag field set by DAIR before returning to the calling routine. The flags have the following meaning:
	Byte 1	
	1...	The data set is allocated but a secondary error occurred. Register 15 contains an error code.
	.000 0000	Reserved bits. Set to zero.
	Byte 2	Reserved. Set to zero.
2	DA08DARC	This field contains the error code, if any, returned from the Dynamic Allocation routines. (See "Return Codes from Dynamic Allocation.")
2	DA08CTRC	This field contains the error code, if any, returned from Catalog Management Routines.

Figure 21. DAIR Parameter Block -- Entry Code X'08' (Part 1 of 3)

Number of Bytes	Field	Contents or Meaning
4	DA08PDSN	Place in this field the address of the DSNAME buffer. The DSNAME buffer is a 46 byte field with the following format: The first two bytes contain the length, in bytes, of the DSNAME; the next 44 bytes contain the DSNAME, left justified and padded to the right with blanks.
8	DA08DDN	This field contains the DDNAME for the data set. If a specific DDNAME is not required, fill this field with eight blanks; DAIR will place in this field the DDNAME to which the data is allocated.
8	DA08UNIT	Unit name desired. If name blank, defaults to PSCBGPNM contents. If name is less than eight bytes long, pad it with blanks at right.
8	DA08SER	Serial number desired. Only the first six bytes are significant. If the serial number is less than six bytes, it must be padded to the right with blanks. If the serial number is omitted, the entire field must contain blanks.
4	DA08BLK	Block size requested. This figure represents the average record length desired.
4	DA08PQTY	Primary space quantity desired. The high order byte must be set to zero; the low order three bytes should contain the space quantity required. If the quantity is omitted, the entire field must be set to zero.
4	DA08SQTY	Secondary space quantity desired. The high order byte must be set to zero; the low order three bytes should contain the secondary space quantity required. If the quantity is omitted, the entire field must be set to zero.
4	DA08DQTY	Directory quantity required. The high order byte must be set to zero; the low order three bytes contain the number of Directory blocks desired. If the quantity is omitted, the entire field must be set to zero.
8	DA08MNM	Contains a member name of a partitioned data set. If the name has less than eight characters, pad it to the right with blanks. If the name is omitted, the entire field must contain blanks.
8	DA08PSWD	Contains the password for the data set. If the password has less than eight characters, pad it to the right with blanks. If the password is omitted, the entire field must contain blanks.

Figure 21. DAIR Parameter Block -- Entry Code X'08' (Part 2 of 3)

Number of Bytes	Field	Contents or Meaning
1	DA08DSP1	Flag byte. Set the following bits to indicate the status of the data set:
	0000	Reserved. Set these bits to zero.
	.... 1...	SHR
	.... .1..	NEW
	.... ..1.	MOD
	1	OLD
1	DA08DPS2	Flag byte. Set the following bits to indicate the normal disposition of the data set:
	0000	Reserved bits. Set them to zero.
	.... 1...	KEEP
	.... .1..	DELETE
	.... ..1.	CATLG
	1	UNCATLG
1	DA08DPS3	Flag byte. Set the following bits to indicate the abnormal disposition of the data set:
	0000	Reserved bits. Set them to zero.
	.... 1...	KEEP
	.... .1..	DELETE
	.... ..1.	CATLG
	1	UNCATLG
1	DA08CTL	Flag byte. These flags indicate to the DAIR service routine what operations are to be performed:
	xx..	Indicate the type of units desired for the space parameters, as follows:
	01..	Units are in average block length.
	10..	Units are in tracks (TRKS).
	11..	Units are in cylinders (CYLS).
	..1.	Prefix userid to DSNAME.
	...1	RLSE is desired.
	.... 1...	The data set is to be permanently allocated; it is not to be freed until specifically requested.
	.... .1..	A DUMMY data set is desired
	.... ..1.	Attribute list name supplied.
	0	Reserved bit; set to zero.
3		Reserved bytes; set them to zero.
1	DA08DSO	A flag field. These flags are set by the DAIR service routine; they describe the organization of the data set to the calling routine.
	1...	Indexed Sequential (IS).
	.1..	Physical Sequential (PS).
	..1.	Direct Organization (DO).
	...0 00..	Reserved bits. Set to zero.
	.... ..1.	Partitioned Organization (PO).
	1	Unmoveable.
8	DA08ALN	Attribute list name.

Figure 21. DAIR Parameter Block -- Entry Code X'08' (Part 3 of 3)

After attempting the requested function, DAIR returns one of the following codes in register 15:
0, 4, 8, 12, 16, 20, 28, 32, 44
See the topic "Return Codes from DAIR" for return code meanings.

Code X'0C' - Concatenate the Specified DDNAMES

Build the DAIR Parameter Block shown in Figure 22 to request that DAIR concatenate data sets. Entry code X'0C' indicates that the DDNAMES listed in the DAIR Parameter Block are to be concatenated in the order in which they appear. All data sets listed by DDNAME in the DAIR Parameter Block must be currently allocated.

DAIR marks the DSE entry for each member it concatenates. This is done in case a subsequent request for allocation of a data set requests a member of the group. If the group was concatenated by DAIR, DAIR deconcatenates the group and proceeds with the requested allocation. If the group was concatenated at LOGON, DAIR makes a duplicate entry for the data set; that is, there will be two entries in the DSE for the same data set.

Number of Bytes	Field	Contents or Meaning
2	DA0CCD	Entry code X'000C'
2	DA0CFLG	Reserved. Set this field to zero.
2	DA0CDARC	This field contains the error code, if any, returned from the Dynamic Allocation routines. (See "Return Codes from Dynamic Allocation.")
2		Reserved field . Set this field to zero.
2	DA0CNUMB	Place in this field the number of data sets to be concatenated.
2		Reserved. Set this field to zero.
8	DA0CDDN	Place in this field the DDNAME of the first data set to be concatenated. This field is repeated for each DDNAME to be concatenated.

Figure 22. DAIR Parameter Block -- Entry Code X'0C'

After attempting the requested function, DAIR returns one of the following codes in register 15.

0, 4, 12

See the topic "Return Codes from DAIR" for return code meanings.

Code X'10' - Deconcatenate the Indicated DDNAME

Build the DAIR Parameter Block shown in Figure 23 to request that DAIR deconcatenate a data set. Entry code X'10' indicates that the DDNAME specified within the DAIR Parameter Block has been previously concatenated and is now to be deconcatenated.

Number of Bytes	Field	Contents or Meaning
2	DA10CD	Entry code X'0010'
2	DA10FLG	Reserved. Set this field to zero.
2	DA10DARC	This field contains the error code, if any, returned from the Dynamic Allocation routines. (See "Return Codes from Dynamic Allocation.")
2		Reserved field. Set this field to zero.
8	DA10DDN	Place in this field the DDNAME of the data set to be deconcatenated.

Figure 23. DAIR Parameter Block -- Entry Code X'10'

After attempting the requested function, DAIR returns one of the following codes in register 15:

0, 4, 12

See the topic "Return Codes from DAIR" for return code meanings.

Code X'14' - Return Qualifiers for the Specified DSNAME

Build the DAIR Parameter Block shown in Figure 24 to request that DAIR return all qualifiers for the DSNAME specified.

You must also provide the return area pointed to by the third word of the DAIR Parameter Block. If the area you provide is larger than needed for all returned information, the remaining bytes in the area are set to zero by DAIR. If the area is smaller than required, it is filled to its limit, and the return code specifies this condition.

Number of Bytes	Field	Contents or Meaning
2	DA14CD	Entry code X'0014'.
2	DA14FLG	Reserved. Set this field to zero.
4	DA14PDSN	Place in this field the address of the DSNAME buffer. The DSNAME buffer is a 46 byte field with the following format: The first two bytes contain the length, in bytes, of the DSNAME; The next 44 bytes contain the DSNAME, left justified and padded to the right with blanks. DSNAME alone represents an unqualified index entry.
4	DA14PRET	Place in this field the address of the return area in which DAIR is to place the qualifiers found for the DSNAME. Place the length of the return area in the first two bytes of the return area. Set the next two bytes in the return area to zero. DAIR returns each of the qualifiers it finds in two fullwords of storage beginning at the first word (offset 0) within the return area.
1	DA14CTL	A flag field:
	Byte 1	
	00.0 0000	Reserved bits; set them to zero.
	..1.	Prefix userid to DSNAME.
3		Reserved bytes. Set this field to zero.

Figure 24. DAIR Parameter Block -- Entry Code X'14'

After attempting the requested function, DAIR returns one of the following codes in register 15:

0, 4, 36, 40

See the topic "Return Codes from DAIR' for return code meanings.

Code X'18' - Free the Specified Data set

Build the DAIR Parameter Block shown in Figure 25 to request that DAIR free a data set. Entry code X'18' indicates that the data set name represented by DSNAME is to be freed. If no DSNAME is given, the data set associated with the DDNAME is freed. If both DDNAME and DSNAME are given, DAIR ignores the DDNAME.

If the specified DSNAME appears several times in the Data Set Extension, all such entries are freed.

Number of Bytes	Field	Contents or Meaning
2	DA18CD	Entry code X'0018'.
2	DA18FLG	A flag field set by DAIR before returning to the calling routine. The flags have the following meanings:
	Byte 1	
	1...	The data set is freed but a secondary error occurred. Register 15 contains an error code.
	.000 0000	Reserved bits. Set to zero.
	Byte 2	Reserved. Set to zero.
2	DA18DARC	This field contains the error code, if any, returned from the Dynamic Allocation routines. (See "Return Codes from Dynamic Allocation.")
2	DA18CTRC	This field contains the error code, if any, returned from Catalog Management routines.
4	DA18PDSN	Place in this field the address of the DSNAME buffer. The DSNAME buffer is a 46 byte field with the following format: The first two bytes contain the length, in bytes, of the DSNAME; The next 44 bytes contain the DSNAME, left justified and padded to the right with blanks.
8	DA18DDN	Place in this field the DDNAME of the data set to be freed, or zeros.
8	DA18MNM	Contains the member name of a partitioned data set. If the name has less than eight characters, pad it to the right with blanks. If the name is omitted, the entire field must contain blanks.
2	DA18SCLS	SYSOUT class. An alphabetic or numeric character. If SYSOUT is not specified, this field must contain blanks.

Figure 25. DAIR Parameter Block -- Entry Code X'18' (Part 1 of 2)

Number of Bytes	Field	Contents or Meaning
1	DA18DPS2	Flag byte. Set the following bits to indicate the normal disposition of the data set:
	0000	Reserved bits. Set them to zero.
	.... 1...	KEEP
	.... .1..	DELETE
	.... ..1.	CATLG
	1	UNCATLG
1	DA18CTL	Flag byte. These flags indicate to the DAIR service routine what operations are to be performed:
	..1.	Prefix userid to DSNAME.
	00.. 0000	Reserved bits; set them to zero.
	...1	If this bit is on, permanently allocated data sets are unallocated and marked "not in use." If the bit is off, the data set will be marked "not in use," if it is permanently allocated.
8	DA18JBNM	Place the jobname for enqueuing SYSOUT data sets in this field. If the jobname is omitted, DAIR takes the jobname from the TIOT.

Figure 25. DAIR Parameter Block -- Entry Code X'18' (Part 2 of 2)

After attempting the requested function, DAIR returns one of the following codes in register 15:

0, 4, 8, 12, 24, 28

See the topic "Return Codes from DAIR" for return code meanings.

Code X'1C' - Allocate the Specified DDNAME to the Terminal

Build the DAIR Parameter Block shown in Figure 26 to request that DAIR allocate a DDNAME to the terminal. Entry code X'1C' indicates that the DDNAME specified within the DAIR Parameter Block is to be allocated to the terminal. If the DDNAME field is left blank, DAIR returns the allocated DDNAME in that field. To supply DCB information, provide the name of an attribute list that has been defined previously by a X'34' entry into DAIR.

Number of Bytes	Field	Contents or Meaning
2	DA1CCD	Entry code X'001C'
2	DA1CFLG	Reserved field; set it to zero.
2	DA1CDARC	This field contains the error code, if any, returned from the Dynamic Allocation routines. (See "Return Codes from Dynamic Allocation.")
1		Reserved field; set it to zero.
1	DA1CCTL .... 1... .... ..1. xxxx .x.x	Control byte: The data set is to be permanently allocated; it is not to be freed until specifically requested. Attribute list name supplied. Each x represents a reserved bit.
8	DA1CDDN	Place in this field the DDNAME for the data set to be allocated to the terminal.
8	DA1CALN	Attribute list name.

Figure 26. DAIR Parameter Block -- Entry Code X'1C'

After attempting the requested function, DAIR returns one of the following codes in register 15:

0, 4, 12, 16, 20, 28

See the topic "Return Codes from DAIR" for return code meanings.

Code X'24' - Allocate a Data Set by DDNAME

Build the DAIR Parameter Block shown in Figure 27 to request that DAIR allocate a data set by DDNAME.

DAIR searches the Data Set Extension using as an argument the DDNAME you specify in the DAIR Parameter Block.

If DAIR locates the DDNAME you specify and a DSNAME is currently associated with it, the associated DSNAME is allocated overriding the DSNAME pointed to by third word of your DAIR Parameter Block. DAIR replaces the DSNAME in your DSNAME buffer with the DSNAME found associated with the DDNAME you specified, and updates the buffer length field. The DDNAME must also be permanently allocated when found or allocation will be by DSNAME with a generated DDNAME.

If there is no DSNAME associated with the DDNAME you specified, it is DYNAM or does not exist. DAIR searches the DSE using the DSNAME you specify as an argument. If DAIR cannot allocate by DDNAME, it will give control to code X'08' to allocate by DSNAME and will generate a new DDNAME.

Number of Bytes	Field	Contents or Meaning
2	DA24CD	Entry code X'0024'.
2	DA24FLG	A flag field set by DAIR before returning to the calling routine. The flags have the following meaning:
	Byte 1	
	1...	The data set is allocated but a secondary error occurred. Register 15 contains an error code.
	.... 1...	DDNAME requested is allocated as DUMMY.
	.000 .000	Reserved bits. Set to zero.
	Byte 2	Reserved. Set to zero.
2	DA24DARC	This field contains the error code, if any, returned from the Dynamic Allocation routines. (See "Return Codes from Dynamic Allocation.")
2	DA24CTRC	This field contains the error code, if any, returned from Catalog Management Routines.
4	DA24PDSN	Place in this field the address of the DSNAME buffer. The DSNAME buffer is a 46 byte field with the following format: The first two bytes contain the length, in bytes, of the DSNAME; The next 44 bytes contain the DSNAME, left justified and padded to the right with blanks.
8	DA24DDN	Place here the DDNAME for the data set to be allocated. This DDNAME is required.
8	DA24UNIT	Unit name desired. If blank, defaults to PSCBGPNM contents. If the unit name is less than eight bytes, pad it to the right with blanks.

Figure 27. DAIR Parameter Block -- Entry Code X'24' (Part 1 of 3)

Number of Bytes	Field	Contents or Meaning
8	DA24SER	Serial number desired. Only the first six bytes are significant. If the serial number is less than six bytes, it must be padded to the right with blanks. If the serial number is omitted, the entire field must contain blanks.
4	DA24BLK	Block size requested. This figure represents the average record length desired.
4	DA24PQTY	Primary space quantity desired. The high order byte must be set to zero; the low order three bytes should contain the space quantity required. If the quantity is omitted, the entire field must be set to zero.
4	DA24SQTY	Secondary space quantity desired. The high order byte must be set to zero; the low order three bytes should contain the secondary space quantity required. If the quantity is omitted, the entire field must be set to zero.
4	DA24DQTY	Directory quantity required. The high order byte must be set to zero; the low order three bytes contain the number of Directory blocks desired. If the quantity is omitted, the entire field must be set to zero.
8	DA24MNM	Contains a member name of a partitioned data set. If the name has less than eight characters, pad it to the right with blanks. If the name is omitted, the entire field must contain blanks.
8	DA24PSWD	Contains the password for the data set. If the password has less than eight characters, pad it to the right with blanks. If the password is omitted, the entire field must contain blanks.
1	DA24DSP1	Flag byte. Set the following bits to indicate the status of the data set:
	0000	Reserved. Set these bits to zero.
	.... 1...	SHR
	.... .1..	NEW
	.... ..1.	MOD
	1	OLD
1	DA24DPS2	Flag byte. Set the following bits to indicate the normal disposition of the data set:
	0000	Reserved bits. Set them to zero.
	.... 1...	KEEP
	.... .1..	DELETE
	.... ..1.	CATLG
	1	UNCATLG

Figure 27. DAIR Parameter Block -- Entry Code X'24' (Part 2 of 3)

Number of Bytes	Field	Contents or Meaning
1	DA24DPS3	Flag byte. Set the following bits to indicate the abnormal disposition of the data set:
	0000	Reserved bits. Set them to zero.
	.... 1...	KEEP
	.... .1..	DELETE
	.... ..1.	CATLG
	1	UNCATLG
1	DA24CTL	Flag byte. These flags indicate to the DAIR service routine what operation are to be performed:
	XX..	Indicate the type of units desired for the space parameters, as follows:
	01..	Units are in average block length.
	10..	Units are in tracks (TRKS).
	11..	Units are in cylinders (CYLS).
	..1.	Prefix userid to DSNAME.
	...1	RLSE is desired.
	.... 1...	The data set is to be permanently allocated; it is not to be freed until specifically requested.
	.... .1..	A DUMMY data set is desired
	.... ..1.	Attribute list name supplied.
	0	Reserved bit; set to zero.
3		Reserved bytes; set them to zero.
1	DA24DSO	A flag field. These flags are set by the DAIR service routine; they describe the organization of the data set to the calling routine.
	1...	Indexed Sequential (IS).
	.1..	Physical Sequential (PS).
	..1.	Direct Organization (DO).
	...0 00..	Reserved bits. Set to zero.
	.... ..1.	Partitioned Organization (PO).
	1	Unmoveable.
8	DA24ALN	Attribute list name.

Figure 27. DAIR Parameter Block -- Entry Code X'24' (Part 3 of 3)

After attempting the requested function, DAIR returns one of the following codes in register 15:

0, 4, 8, 12, 16, 20

See the topic "Return Codes from DAIR" for return code meanings.

Code X'28' - Perform a List of DAIR Operations

Build the DAIR Parameter Block shown in Figure 28 to request that DAIR perform a list of operations. This DAIR Parameter Block points to other DAPBs which request the operations to be performed.

All valid DAIR functions are acceptable; however, code X'14' or another code X'28' are ignored.

DAIR processes the requested operations in the order they are requested.

DAIR processing stops with the first operation that fails.

Number of Bytes	Field	Contents or Meaning
2	DA28CD	Entry code X'0028'.
2	DA28NOP	Place in this field the number of operations to be performed.
4	DA28PFOP	DAIR fills this field with the address of the DAIR Parameter Block for the first operation that failed. If all operations are successful, this field will contain zero upon return from the DAIR service routine. If this field contains an address, register fifteen contains a return code.
4	DA28OPTR	Place in this field the address of the DAIR Parameter Block for the first operation you want performed. Repeat this field, filling it with the addresses of the DAPLs, for each of the operations to be performed.

Figure 28. DAIR Parameter Block -- Entry Code X'28'

After attempting the requested function, DAIR returns one of the following codes in register 15:

Any code accepted by any of the other DAIR functions, except '36' and '40'.

For return code meanings see the topic "Return Codes from DAIR."

Code X'2C' - Mark Data Sets as Not in Use

Build the DAIR Parameter Block shown in Figure 29 to request that DAIR mark DSE entries associated with a Task Control Block as not in use. This allows TIOT entries to be reused.

This is the code which the TMP should pass to DAIR prior to detaching a command processor. This code should also be issued by any command processor which attaches another command processor and detaches that command processor directly.

Number of Bytes	Field	Contents or Meaning
2	DA2CCD	Entry code X'002C'.
2	DA2CFLG	A flag field. Set the bits to indicate to the DAIR service routine which data sets you want marked not in use. Hex setting — Meaning 0000 — Mark all data sets of the indicated TCB "not in use". 0001 — Mark the specified DDNAME "not in use". 0002 — Mark all DSEs associated with lower tasks "not in use".
4	DA2CTCB	Place in this field the address of the TCB for the task whose data sets are to be marked "not in use".
8	DA2CDDN	Place in this field the DDNAME to be marked "not in use". DA2CFLG must be set to hex 0001.

Figure 29. DAIR Parameter Block -- Entry Code X'002C'

After attempting the requested function, DAIR returns one of the following codes in register 15:

0, 4

For return code meanings see the topic "Return Codes from DAIR."

Code X'30' - Allocate a SYSOUT Data Set

Build the DAIR Parameter Block shown in Figure 30 to request that DAIR allocate a SYSOUT data set. The exact action taken by DAIR is dependent upon the presence of the optional fields and the setting of bits in the control byte. To supply DCB information, provide the name of an attribute list that has been defined previously by a X'34' entry into DAIR.

Number of Bytes	Field	Contents or Meaning
2	DA30CD	Entry code X'0030'.
2	DA30FLG	A flag field set by DAIR before returning to the calling routine. The flags have the following meaning:
	Byte 1	
	1...	The data set is allocated but a secondary error occurred. Register 15 contains an error code.
	.000 0000	Reserved bits. Set to zero.
	Byte 2	Reserved. Set to zero.
2	DA30DARC	This field contains the error code, if any, returned from the Dynamic Allocation routines. (See "Return Codes from Dynamic Allocation.")
2		Reserved. Set this field to zero.
4	DA30PDSN	Place in this field the address of the DSNAME buffer. The DSNAME buffer is a 46 byte field with the following format: The first two bytes contain the length, in bytes, of the DSNAME; The next 44 bytes contain the DSNAME, left justified and padded to the right with blanks.
8	DA30DDN	This field contains the DDNAME for the data set. If a specific DDNAME is not required, fill this field with eight blanks; DAIR will place in this field the DDNAME to which the data is allocated.
8	DA30UNIT	Unit name desired. If blank, defaults to PSCBGPNM contents. If name is less than eight bytes, pad it at right with blanks.
8	DA30SER	Serial number desired. Only the first six bytes are significant. If the serial number is less than six bytes, it must be padded to the right with blanks. If the serial number is omitted, the entire field must contain blanks.
4	DA30BLK	Block size requested. This figure represents the average record length desired.

Figure 30. DAIR Parameter Block -- Entry Code X'30' (Part 1 of 2)

Number of Bytes	Field	Contents or Meaning
4	DA30PQTY	Primary space quantity desired. The high order byte must be set to zero; the low order three bytes should contain the space quantity required. If the quantity is omitted, the entire field field must be set to zero.
4	DA30SQTY	Secondary space quantity desired. The high order byte must be set to zero; the low order three bytes should contain the secondary space quantity required. If the quantity is omitted, the entire field must be set to zero.
8	DA30PGNM	Place in this field the member name of a special user program to handle SYSOUT operations. Fill this field with blanks if you do not provide a program name.
4	DA30FORM	Form number. This form number indicates that the output should be printed or punched on a specific output form. It is a four character number. This field must be filled with blanks if this parameter is omitted.
2	DA30OCLS	SYSOUT class. Place a single alphameric character in either byte of this field and a blank in the other byte. The data set will be allocated to the message class, regardless of the class that you specify here. To place a SYSOUT data set in a class other than the message class, use DAIR entry code X'30', specifying any valid class. When the output has been written, specify the desired SYSOUT class either by using DAIR entry code X'18' or by issuing the FREE command.
1		Reserved. Set this field to zero.
1	DA30CTL	Flag byte. These flags indicate to the DAIR service routine what operations are to be performed.
	XX..	Indicate the type of units desired for the space parameters, as follows:
	01..	Units are in average block length.
	10..	Units are in tracks (TRKS).
	11..	Units are in cylinders (CYLS).
	..1.	Prefix userid to DSNAME
	...1	RLSE is desired.
	.... 1...	The data set is to be permanently allocated; it is not to be freed until specifically requested.
	.... .1..	A DUMMY data set is desired.
	.... ..1.	Attribute list name specified.
	0	Reserved bit; set to zero.
8	DA30ALN	Attribute list name.

Figure 30. DAIR Parameter Block -- Entry Code X'30' (Part 2 of 2)

After attempting the requested function, DAIR returns one of the following codes in register 15:

0, 4, 12, 16, 20, 28

See the topic "Return Codes from DAIR" for return code meanings.

Code X'34' - Build or Delete an Attribute Control Block (ATRCB)

Build the DAIR Parameter Block shown in Figure 30.1 to request that DAIR construct an ATRCB, delete an ATRCB, or search the chain of ATRCBs for a specific name. The exact action taken by DAIR is dependent upon the setting of bits in the control byte.

Note: When you request that DAIR construct an ATRCB, you must also build a DAIR Attribute Control Block (DAIRACB).

Number of Bytes	Field	Contents or Meaning
2	DA34CD	Entry code X'0034'.
2	DA34FLG	A flag field set by DAIR before returning to the calling routine. The flags have the following meaning:
	Byte 1 DA34FIND	
	1...	An attribute list name was found.
	0...	An attribute list name was not found.
	.000 0000	Reserved bits. Set to zero.
	Byte 2	Reserved.
2	DA34DARC	This field contains the code returned from the Dynamic Allocation routines. (See "Return Codes from Dynamic Allocation.")
1	DA34CTRL	Flag byte. These flags indicate to DAIR what operations are to be performed.
	DA34SRCH 1...	Search the ATRCB chain for the attribute list name specified in field DA34NAME.
	DA34CHN .1..	Build and chain an attribute list (ATRCB).
	DA34UNCH ..1.	Delete an ATRCB from the chain.
	...0 0000	Reserved bits. Set to zero.
1		Reserved.
8	DA34NAME	This field contains the name for the list of attributes.
4	DA34ADDR	This field contains the address of the DAIR Attribute Control Block (DAIRACB).

Figure 31. DAIR Parameter Block -- Entry Code X'34'

DAIRACB - DAIR Attribute Control Block

Build the DAIRACB shown in Figure 32 when you request that DAIR construct an attribute control block (ATRCB). Place the address of the DAIRACB into the DA34ADDR field of the code X'34' DAIR parameter block shown in Figure 31.

Number of Bytes	Field	Contents or Meaning
8		Reserved.
8	DAIMASK	First 6 bytes and eighth byte are reserved.
	DAILABEL	Seventh-byte flags. These flags indicate the INOUT/OUTIN options of the OPEN macro.
	DAIINOUT	
	1...	Use the INOUT option.
	DAIOUTIN	
	.1..	Use the OUTIN option.
	..xx xxxx	Reserved bits.
3		Reserved.
3	DAIEXPDT	This field contains a data set expiration date.
	DAIYEAR	The first byte contains the expiration year.
	DAIDAY	The next 2 bytes contain the expiration day, left justified (x'dddn).
2		Reserved.
1	DAIBUFNO	This field contains the number of buffers required.
1	DAIBFTEK	This field contains the buffer type and alignment.
	.1..	Simple buffering (S).
	.11.	Automatic record area construction (A).
	..1.	Record buffering (R).
	...1	Exchange buffering (E).
	.... ..1.	Doubleword boundary (D).
	1	Fullword boundary (F).
	x... xx..	Reserved bits.
2	DAIBUFL	This field contains the buffer length.
1	DAIEROPT	This field indicates the error options:
	1...	Accept error record.
	.1..	Skip error record.
	..1.	Abnormal ECT.
	...x xxxx	Reserved bits.
1	DAIKEYLE	This field contains the key length.
6		Reserved.

Figure 32. DAIR Attribute Control Block (DAIRACB) (Part 1 of 2)

Number of Bytes	Field	Contents or Meaning
1	DAIRECFM	This field indicates the record format:
	1...	Fixed (F).
	.1..	Variable (V).
	11..	Undefined (U).
	..1.	Track overflow (T).
	...1	Blocked (B).
	.... 1...	Standard Blocks (S).
	.... .1..	ASA printer characters (A).
	.... ..1.	Machine control characters (M).
	x	Reserved bits.
1	DAIOPTCD	This field contains the error option codes:
	1...	Write validity check (W).
	..1.	Chained scheduling (C).
	.... 1...	ANSI translate (Q).
	.... ..1.	User totaling (T).
	.x.x .x.x	Reserved bits.
2	DAIBLKSI	This field contains the maximum block size.
2	DAILRECL	This field contains the logical record length.
1	DAINCP	This field contains the maximum number of channel programs.
4		Reserved.

Figure 32. DAIR Attribute Control Block (DAIRACB) (Part 2 of 2)

The fields that you do not use must be initialized to zero.

Return Codes from DAIR

DAIR returns a code in general register 15 to the calling routine. In addition, DAIR sets certain return codes in the DAxxDARC field of a DAIR Parameter Block. (See items preceded by an asterisk in "Return Codes from Dynamic Allocation.")

The DAIR return codes have the following meaning:

CODE decimal	MEANING
0	DAIR completed successfully.
4	The parameter list passed to DAIR was invalid.
8	An error occurred in a catalog management routine; the catalog management error code is stored in the CTRC field of the DAIR Parameter Block.
12	An error occurred in dynamic allocation; the dynamic allocation error code is stored in the DARC field of the DAIR Parameter Block.
16	No TIOT entries were available for use.
20	The DDNAME requested is unavailable.
24	The DSNAME requested is a member of a concatenated group.
28	The DDNAME or DSNAME specified is not currently allocated, or the attribute list name specified was not found.
32	The requested data set was previously permanently allocated, or was allocated with a disposition of new, and was not deleted. DISP=NEW cannot now be specified.
36	An error occurred in a catalog information routine.
40	The return area you provided for qualifiers was exhausted and more index blocks exist. If you require more qualifiers, provide a larger return area.
44	The previous allocation specified a disposition of DELETE for this non-permanently allocated data set. Request specified OLD, MOD, or SHR with no volume serial number.
48	Returned from DAIR STAE routine when an ABEND has occurred.

Return Codes from Dynamic Allocation

Both DAIR and the Dynamic Allocation routines called by DAIR may return a code in the DAxxDARC field of the DAIR Parameter Block.

Note: Codes that can be returned by DAIR are preceded by an asterisk. The asterisk is not part of the return code.)

The return codes have the following meaning:

RETURN CODE hexadecimal — MEANING

0000 Dynamic Allocation completed successfully.

0004 Dynamic Allocation could not delete a table that was loaded using a LOAD macro instruction. The data set is still allocated.

0008 The temporary data set was freed and deleted. The disposition specified by the calling routine is invalid for a temporary data set.

002w The data set was successfully freed, but the disposition (catalog or uncatalog) was unsuccessful. The hexadecimal digit 'w' is a code indicating the reason for the failure.

w Explanation

1 A control volume was required and a utility program must be used to catalog the data set.

2 The data set to be cataloged had previously been cataloged or the data set to be uncataloged could not be located, or no change was made to the volume serial list of a data set with a disposition of CATLG.

3 A specified index did not exist.

4 The data set could not be cataloged because space was not available on the specified volume.

5 Too many volumes were specified for the data set; because of this, not enough main storage was available to perform the specified cataloging.

6 The data set to be cataloged in a generation index is improperly named.

7 The data set to be cataloged was not opened and no density information was provided. (For dual density tape requests only).

9 An uncorrectable input/output error occurred in reading or writing the catalog

003x The data set was successfully freed, but the requested disposition (delete) was unsuccessful. The hexadecimal digit 'x' is a code indicating the reason for failure.

x Explanation

1 The expiration date had not occurred.

4 No device was available for mounting during deletion.

5 Too many volumes were specified for deletion.

6 Either no volumes were mounted or the mounted volumes could not be demounted to permit the remaining volumes to be mounted.

8 The SCRATCH routine could not delete the data set from the volume.

9 A job was cancelled and was deleted from any one of the following queues:

Input Queues
Background Reader Queue
Hold Queue
Automatic SYSIN Batching (ASB) Queue
Output Queues

0104 Dynamic Allocation encountered an I/O error while attempting to read from SYS1.SYSJOBQE.

0108 Dynamic Allocation encountered an I/O error while attempting to write to SYS1.SYSJOBQE.

010C Dynamic Allocation encountered an I/O error while enqueueing on SYS1.SYSJOBQE.

0204 Reserved.

0208 No space is available on SYS1.SYSJOBQE.

020C The calling routine made a request for the exclusive use of a shared data set. The request can not be honored.

0210 The data set requested is not available. This data set is allocated to another job and its usage attributes conflict with this request.

0214 A direct access device is not available. To be available it must satisfy the following requirements:

- It must be online.
- It must be ready.
- It must not be pending offline.
- It must not be pending an unload.
- It must be shareable.
- A MOUNT message must not be currently outstanding.
- The volume attributes must have been defined.

*0218 The required volume was not mounted on an available device. Either DAIR or Dynamic Allocation can set this return code.
(See Dynamic Allocation return code 214 for the requirements for an available device.)

021C Incorrect unitname supplied.

0220
through
0264 Reserved.

0268 Concatentaion was requested, but the DCBTIOT offset cannot be found in this job's DEB/DCB chain.

0304 The ddname was not specified by the calling routine.

0308 The ddname specified by the calling routine was not found.

030C An invalid function code was specified by the calling routine.

0310 The "exchange" option was specified by the calling program and the TIOT entry for the second (new) ddname could not be found.

0314 Restoring ddnames, as per this request, would have resulted in duplicate ddnames -- duplicate ddnames are not permitted.

0318 Invalid characters are present in the ddname provided by the caller.

031C Invalid characters are present in the membername provided by the caller.

0320 Invalid characters are present in the dsname provided by the caller.

0324 Invalid characters are present in the SYSOUT program name provided by the caller.

0328 Invalid characters are present in the SYSOUT form number provided by the caller.

032C An invalid SYSOUT class was specified by the caller.

*0330 A membername was specified but the data set is not a partitioned data set. DAIR, not Dynamic Allocation, sets this return code.

0334 The supplied data set name exceeded 44 characters in length.

0338 The data set disposition specified by the caller is invalid.

033C More than one mutually exclusive keyword (DSNAME, DUMMY, TERM, or SYSOUT) was specified.

0340 The dsname was not specified and the disposition was not "new". (If the disposition is "new" the dsname may be omitted.)

0344 Dynamic Allocation was specified in a non-TSO environment.

0348
through
034C Reserved.

0350 Jobname field contains zeros. This field may be blank, but may not contain zeros.

0354 Reserved.

0358 DELETE cannot be specified if the data set is shared.

035C-0360 Reserved.

0364 JOBLIB DDNAME or STEPLIB DDNAME can not be specified. These data sets have been opened and thus cannot be allocated.

0404 The device to be freed is not a direct access device. (Only direct access devices are supported for dynamic allocation.)

0408 The new DDNAME is a duplicate of a DDNAME in the TIOT. The calling routine requested allocation of a file name (DDNAME) already used for the job.

040C The specified ddname is associated with a DYNAM entry. DYNAM entries may not be concatenated.

0410 The specified ddname is allocated to a data set. The ddname must be associated with a DYNAM entry.

0414 The specified ddname is already allocated to a terminal entry (TERM=TS).

0418 The referenced data set is a member of a concatenated data group. If the data set was dynamically concatenated it must be deconcatenated before this request can be honored. If concatenated at LOGON, the data set may not be freed until LOGOFF.

*041C The referenced data set is a multi-volume data set. Multi-volume data sets (data sets on more than one volume) are not supported by Dynamic Allocation. Either DAIR or Dynamic Allocation can set this return code.

0420 The specified ddname is associated with an open data set. (A data set must be closed to be used by Dynamic Allocation.)

0424 Reserved.

0428 The specified ddname is part of a previously allocated space. Dynamic Allocation cannot free it.

042C The ddname to be freed is associated with a generation data group. Generation data groups are not supported in Dynamic Allocation.

0430 The specified ddname is associated with a passed data set. Passed data sets cannot be freed or converted.

0504 A serious error of undetermined cause has occurred involving system data.

*x7zz A Dynamic Allocation return code of this form is constructed of an identifier (x) representing the system macro instruction returning the code, and the code itself (zz) returned by the macro instruction.

> If "x" equals 1, the LOCATE macro instruction returned the code. DAIR, not Dynamic Allocation, returns this code.
>
> If "x" equals 4, the DADSM macro instruction returned the code.
>
> If "x" equals 6, the OBTAIN macro instruction returned the code. DAIR, not Dynamic Allocation, returns this code.

"zz" is the low order byte from register 15 as returned by the macro instruction.

The return codes for the LOCATE and the OBTAIN macro instructions are described in Data Management for System Programmers.

The return codes for the DADSM macro instruction are as follows:

Code	Meaning
00	The operation completed successfully.
04	Duplicate name DSCB.
08	No available DSCB's in the VTOC.
0C	A permanent I/O error occurred in reading or writing a DSCB.
10	The absolute track requested is not available.
14	The quantity of space requested is not available.
18	The record length specified is greater than the track length.
30	The number of tracks requested for a split cylinder data set is greater than the number of tracks per cylinder.
34	The disk pack is a DOS volume and the request is not absolute track.
38	The volume does not have enough space for the directory.
80	The directory space requested is larger than the primary space requested.

Using BSAM or QSAM for Terminal I/O

The Basic Sequential and Queued Sequential access methods provide terminal I/O support for programs operating under the Time Sharing Option. For a complete discussion of the use of BSAM and QSAM, see the publication Data Management Services.

The major benefit of using BSAM or QSAM to process terminal I/O under TSO is that programs using these access methods do not become TSO dependent or device dependent and may execute either under TSO or in the batch environment. Therefore, your existing programs that use BSAM or QSAM for I/O may be used under TSO without modification or recompilation.

This section describes:

- The BSAM/QSAM macro instructions
- SAM Terminal routines
- Record formats, buffering techniques, and processing modes
- Specifying the terminal line size
- End of file (EOF) for input processing
- Modifying DD statements for batch or TSO processing

BSAM/QSAM Macro Instructions

Some of the BSAM and QSAM access method routines have been modified to provide special services under TSO; others provide the same function that is provided in a batch environment. Those BSAM/QSAM macro instructions that are not relevent to terminal I/O act as no-ops. All of the BSAM/QSAM macro instructions, when executed in the batch environment, provide the non-terminal functions as explained in Data Management Macro Instructions. Figure 33 shows the functions performed by the BSAM and QSAM macro instructions when used for terminal I/O. Following the table are more detailed explanations of the GET, PUT, PUTX, READ, WRITE, and CHECK macro instructions.

SAM Macro Instruction	BSAM	QSAM	Terminal Interpretation
BSP	X	X	NOP
BUILD	X	X	As in batch processing, the BUILD macro instruction causes a buffer pool to be constructed in a user-provided main storage area.
BUILDRCD		X	NOP
CHECK	X		Takes an EODAD exit after a READ EOF. NOP after a WRITE.
CLOSE	X	X	The CLOSE macro instruction frees the control blocks built to handle I/O and deletes the loaded SAM terminal routines.
CNTRL	X	X	NOP
FEOV	X	X	NOP
FREEBUF	X		As in batch processing, the FREEBUF macro instruction causes the control program to return a buffer to the buffer pool assigned to the specified data control block.
FREEPOOL	X	X	As in batch processing, the FREEPOOL macro instruction causes an area of main storage, previously assigned as a buffer pool for a specified data control block, to be released.
GET		X	The GET macro instruction obtains data from the terminal via the TGET macro instruction.
GETBUF	X		As in batch processing, the GETBUF macro instruction causes the control program to obtain a buffer from the buffer pool assigned to the specified data control block, and to return the address of the buffer in a designated register.
GETPOOL	X	X	As in batch processing, the GETPOOL macro instruction causes a buffer pool to be constructed in a main storage area provided by the control program.

Figure 33. BSAM/QSAM Function under TSO (Part 1 of 2)

SAM Macro Instruction	BSAM	QSAM	Terminal Interpretation
NOTE	X		NOP
OPEN	X	X	The OPEN macro instruction loads the proper SAM terminal I/O routines and constructs the necessary control blocks.
POINT	X		NOP
PRTOV	X	X	NOP
PUT		X	The PUT macro instruction routes data to the terminal via the TPUT macro instruction.
PUTX		X	The PUTX macro instruction routes data to the terminal via the TPUT macro instruction.
READ	X		The READ macro instruction obtains data from the terminal via the TGET macro instruction.
RELSE		X	NOP
SETPRT	X	X	NOP
TRUNC		X	NOP
WRITE	X		The WRITE macro instruction routes data to the terminal via the TPUT macro instruction.

Figure 33. BSAM/QSAM Function under TSO (Part 2 of 2)

SAM TERMINAL ROUTINES

The GET, PUT, PUTX, READ, WRITE, and CHECK macro instructions perform differently in terminal I/O than the way they do in the batch environment. Descriptions of these differences are presented here, but for a detailed explanation of how to use the macro instructions, see Data Management Macro Instructions.

GET

The GET macro instruction causes a record to be retrieved from the terminal and placed in either the first buffer of the buffer pool control block (locate mode) or in a user specified area (substitute or move mode). In either case, the address of the record is returned in register 1.

The record is moved via a TGET macro instruction which does not return control until the transfer of data is completed.

The input to the GET macro instruction consists of the DCB address and the user's area address (omitted for locate mode). The output is edited (i.e., specially-indicated characters are deleted from the message).

When the terminal user types /*, end of file is indicated and control is passed to the problem program's EODAD routine. If no EODAD routine is specified, the job will ABEND with a system code of 337.

PUT and PUTX

Both the PUT and the PUTX macro instructions cause a record to be written to a terminal. This transfer of data is accomplished with the TPUT macro instruction which does not return control until the transfer is completed.

In locate mode, the first use of PUT or PUTX causes an address pointing to a buffer to be returned in register 1. The first record is placed in this buffer by the problem program and is written out when the next PUT or PUTX for the same data control block (DCB) is issued. Succeeding records are written in the same manner. The last record is written at CLOSE time.

In move or substitute mode, the PUT or PUTX macro instruction moves a record from the user-specified work area to the terminal. You must supply the work area address to the PUT macro instruction.

The input to the PUT and PUTX macro instruction consists of the DCB address and the user's area address (omitted for locate mode).

READ

The READ macro instruction causes a block of data to be retrieved from the terminal and placed in a user-designated area in main storage. This transfer of data is done via a TGET macro instruction which does not return control before the transfer is completed.

The input to the READ macro instruction consists of the string of parameters explained in Data Management Macro Instructions.

WRITE

The WRITE macro instruction causes a block of data to be written from the user-specified area to the terminal. This transfer of data is done via a TPUT macro instruction which does not return control before the transfer is completed.

The input to the WRITE macro instruction consists of the string of parameters explained in Data Management Macro Instructions.

CHECK

The CHECK macro instruction used after a WRITE macro instruction results in a NOP. When it is used after a READ macro instruction, it performs as a NOP unless an end of file (EOF) condition is encountered. The end of file signal from the terminal is /*. When end of file is encountered, CHECK takes the EODAD exit specified in the data control block. If no EODAD exit is specified, CHECK will cause the job to ABEND with a system code of 337.

The input to the CHECK macro instruction is the address of the problem program's data event control block (DECB).

Record Formats, Buffering Techniques, and Processing Modes

All record formats -- Fixed (F), Variable (V), and Undefined (U) -- are supported under TSO. Before passing the data to the problem program, TSO automatically generates the first 4 bytes of control information for V format records coming in from the terminal. When you send V format records to the terminal, TSO automatically removes the control information before writing the line.

Both simple and exchange buffering techniques are supported, as are all four processing modes for the queued access method.

Specifying Terminal Line Size

If the LRECL and BLKSIZE fields are not specified in the DCB, the terminal line size default (or the line size the terminal user has specified via the TERMINAL command) is merged into the data control block fields as if it came from the label of the data set.

For BSAM, BLKSIZE is used by TSO to determine the length of the text line it is to process. For both BSAM and QSAM, if the text entered from the terminal is shorter than the value specified for LRECL, and if F format is used, blanks are supplied on the right. For either access technique, if the text entered is longer than BLKSIZE or LRECL, the next GET or READ retrieves the remainder of the message. If the record generated by the problem program is longer than the specified line size, multiple lines are printed at the terminal.

End of File (EOF) for Input Processing

The sequential access method GET and CHECK terminal routines recognize /* from the terminal as an end of file (EOF). The EODAD exit in the data control block is taken for the EOF condition. If no EODAD exit has been specified, and an EOF has been signaled from the terminal, the job ABENDs with a system code of 337.

Modifying DD Statements for Batch or TSO Processing

A new parameter, TERM=TS, has been provided for the JCL Data Definition (DD) statement.

TERM=TS, when added to a DD statement defining an input or an output data set, is ignored in the batch processing environment, but under TSO indicates to the system that the unit to which I/O is being addressed is a time sharing terminal. Thus a user who wants his job to run in either the foreground or the background could provide a DD statement as follows:

```
//DD1   DD   TERM=TS,SYSOUT=A
```

In this example the output device is defined as a terminal under TSO processing, and as the SYSOUT device during batch processing. For a complete description of the TERM=TS parameter, see Job Control Language Reference.

Using the TSO I/O Service Routines for Terminal I/O

The TSO I/O Service Routines process terminal I/O requests initiated by the Terminal Monitor Program (TMP), Command Processors (CPs), and other service routines. If you write your own Command Processors, or replace the TSO-supplied Terminal Monitor Program with one of you own design, you should use the I/O Service Routines to process terminal I/O.

The I/O Service Routines -- STACK, GETLINE, PUTLINE, and PUTGET -- offer the following features:

1. They provide an interface between an I/O request and the TGET and TPUT supervisor calls.
2. They provide a method of selecting sources of input other than the terminal. Requests for input can be directed to an in-storage list as well as to the terminal.
3. They provide a message formatting facility with which you can insert text segments into a basic message format, and print or inhibit the printing of message identifiers at the terminal.
4. They process requests for more information (question mark processing), and they analyze processing conditions to determine if I/O requests should be disregarded or honored.

The I/O Service Routines build, modify, or make use of various control blocks. The following control block DSECTS are provided in SYS1.MACLIB for your use:

IKJCPPL	- The Command Processor Parameter List
IKJIOPL	- The Input Output Parameter List
IKJSTPB	- The STACK Parameter Block
IKJGTPB	- The GETLINE Parameter Block
IKJPTPB	- The PUTLINE Parameter Block
IKJPGPB	- The PUTGET Parameter Block
IKJLSD	- The List Source Descriptor
IKJECT	- The Environment Control Table

You pass control to the I/O Service Routines and indicate the functions you want performed by coding the operands you require in the List and the Execute forms of the I/O Service Routine macro instructions. Each of the I/O Service Routine macro instructions (STACK, GETLINE, PUTLINE, and PUTGET) has a List and an Execute form.

The List form of each Service Routine macro instruction initializes the parameter blocks according to the operands you code into the macro instruction.

The Execute form is used to modify the parameter blocks and to provide linkage to the Service Routines, and can be used to set up the Input Output Parameter List. The Input Output Parameter List contains addresses required by the I/O services routines.

This following paragraphs describe:

- The Interface with the I/O Service Routines
- Passing Control to the I/O Service Routines
- The I/O Service Routines Macro Instructions

  STACK
  GETLINE
  PUTLINE
  PUTGET

Interface with the I/O Service Routines

When the Terminal Monitor Program attaches a Command Processor, register 1 contains a pointer to a Command Processor Parameter List (CPPL) containing addresses required by the Command Processor. The CPPL is located in subpool 1, which is read-only storage for the Command Processors. The control block interface between the TMP and an attached CP is shown in Figure 34.

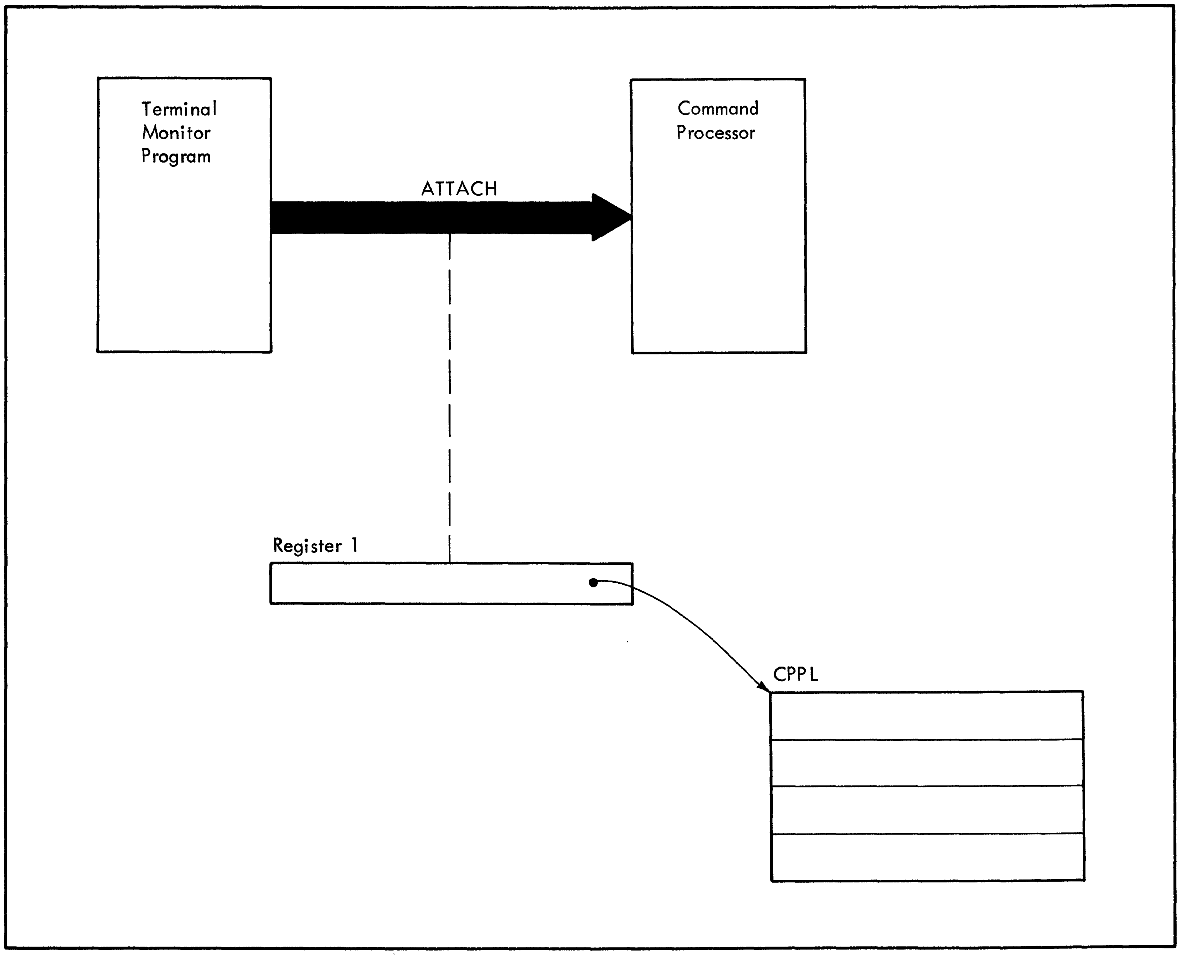


Figure 34. Control Block Interface Between TMP and CP

THE COMMAND PROCESSOR PARAMETER LIST

You must pass certain addresses contained in the CPPL to the I/O Service Routines. Your user-written Command Processors can access the CPPL via the symbolic field names contained in the IKJCPPL DSECT by using the address received in register 1 as a starting address for the DSECT. The use of the DSECT is recommended since it protects the Command Processor from any changes to the CPPL.

The Command Processor Parameter List, as defined by the IKJCPPL DSECT, is a four word parameter list. Figure 35 describes the contents of the CPPL. (See Figure 5, the Test Parameter List, for a definition of each table whose address is in the CPPL.)

Number of Bytes	Field Name	Contents or Meaning
4	CPPLCBUF	The address of the command buffer.
4	CPPLUPT	The address of the User's Profile Table (UPT).
4	CPPLPSCB	The address of the Protected Step Control Block (PSCB).
4	CPPLECT	The address of the Environment Control Table (ECT).

Figure 35. The Command Processor Parameter List (CPPL)

You must place the addresses of the User Profile Table and the Environment Control Table in another control block, the Input Output Parameter List, and pass them to the I/O Service Routines.

THE INPUT OUTPUT PARAMETER LIST

The I/O Service Routines use two of the pointers contained in the Command Processor Parameter List -- the pointer to the User Profile Table and the pointer to the Environment Control Table. These addresses are passed to the Service Routines in another parameter list, the Input Output Parameter List (IOPL). Before executing any of the TSO I/O macro instructions (GETLINE, PUTLINE, PUTGET, or STACK) you must provide an IOPL and pass its address to the I/O Service Routine. There are two ways you can construct an IOPL:

1. You can build and initialize the IOPL within your code and place a pointer to it in the execute form of the I/O macro instruction.

2. You can provide space for an IOPL (4 fullwords), pass a pointer to it together with the addresses required to fill it, to the execute form of the I/O macro instruction, and let the I/O macro instruction build the IOPL for you.

The Input Output Parameter List, as defined by the IKJIOPL DSECT, is a four word parameter list. Figure 36 describes the contents of the IOPL.

Number of Bytes	Field Name	Contents or Meaning
4	IOPLUPT	The address of the User Profile Table from the CPPLUPT field of the Command Processor Parameter List.
4	IOPLECT	The address of the Environment Control Table from the CPPLECT field of the CPPL.
4	IOPLECB	The address of the command processor's Event Control Block (ECB). The ECB is one word of storage, declared and initialized to zero by the command processor. Command processors with attention exits can post this ECB after an attention interruption to cause active service routines to exit.
4	IOPLIOPB	The address of the parameter block created by the list form of the I/O macro instruction. There are four types of parameter blocks, one for each of the I/O Service Routines: Stack Parameter Block (STPB) Getline Parameter Block (GTPB) Putline Parameter Block (PTPB) Putget Parameter Block (PGPB)

Figure 36. The Input Output Parameter List

The Parameter Block pointed to by the fourth word (IOPLIOPB) of the I/O Parameter List is built and modified by the I/O Service routine macros themselves. It is created and initialized by the list form of the I/O macro instruction, and modified by the execute form. Thus you can use the same parameter block to perform different functions. All you need to do is code different parameters in the execute forms of the macro instructions; these parameters provide those options not specified in the list form, and override those which were specified. Each of these parameter blocks -- the STACK, GETLINE, PUTLINE, and PUTGET Parameter blocks -- is described in the separate sections on each of the I/O macro instructions.

Figure 37, an extension of Figure 34, summarizes the control block interfaces established between the Terminal Monitor Program and an I/O Service Routine.

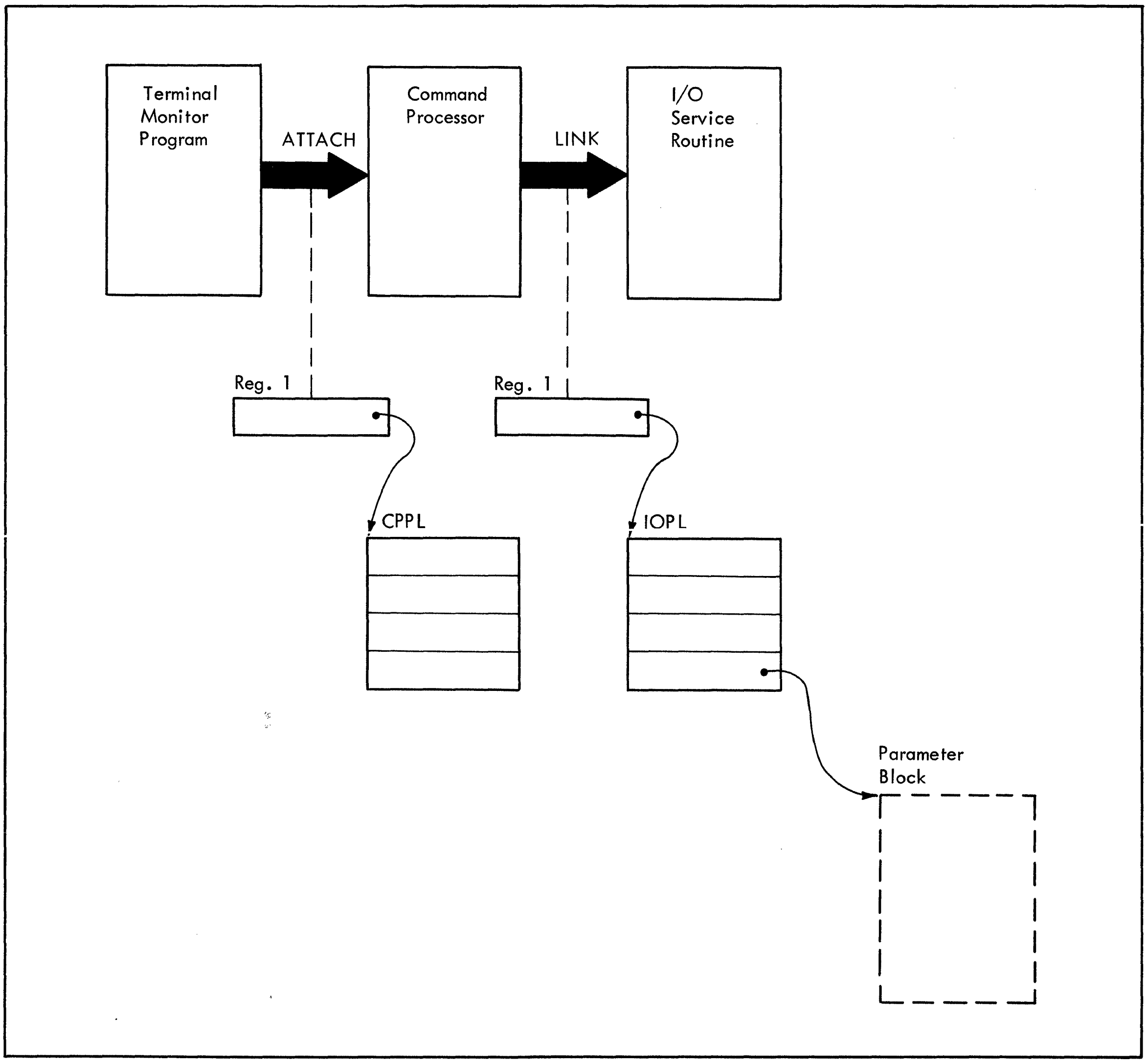


Figure 37. Control Block Interface Between TMP and I/O Service Routine

Passing Control to the I/O Service Routines

There are two ways you can pass control to the I/O Service routines.

1. You can issue a LOAD macro instruction for the load module containing the required service routine, and code the entry point address of that routine in the TSO I/O macro instruction via the ENTRY parameter. In this case, the I/O macro instruction will execute a branch and link register instruction (BALR) using the entry point as the branch address. All of the TSO Terminal I/O Service Routines are contained within the IKJPTGT load module. Their entry points are:

Service Routine	Entry Point
• STACK	IKJSTCK
• GETLINE	IKJGETL
• PUTLINE	IKJPUTL
• PUTGET	IKJPTGT

   If your region space requirements are critical, you can use the DELETE macro instruction to release the main storage area occupied by the load module when you have finished with your terminal I/O.

2. You can issue the I/O macro instruction and not include the ENTRY parameter. In this case, the I/O macro instruction generates a LINK macro instruction to invoke the I/O Service Routine.

The I/O Service Routine Macro Instructions

The I/O Service routines -- STACK, GETLINE, PUTLINE, and PUTGET -- each perform a specific I/O function:

- STACK determines the source of input.

- GETLINE obtains a line of input.

- PUTLINE puts a line of output to the terminal.

- PUTGET puts a line to the terminal and gets a line in response.

In order to perform these functions, the I/O macro instructions use the control blocks explained in the section 'INTERFACE WITH THE I/O SERVICE ROUTINES", and other, more individualized control blocks, the parameter blocks. Each of the I/O macro instructions has a list and an execute form. The list form sets up the Parameter Block required by that I/O service routine; the execute form can be used to set up the Input Output Parameter List, and to modify the parameter block created by the list form of the macro instruction.

The Parameter Block required by each of the I/O service routines is different, and each one may be referenced through a DSECT. The Parameter Blocks and the DSECTS used to reference them are:

• The STACK Parameter Block	IKJSTPB
• The GETLINE Parameter Block	IKJGTPB
• The PUTLINE Parameter Block	IKJPTPB
• The PUTGET Parameter Block	IKJPGPB

Each of these blocks is explained in the section describing the I/O macro instruction that builds it.

STACK - CHANGING THE SOURCE OF INPUT

Use the STACK macro instruction to establish and to change the source of input. The currently active input source is described by the top element of the Input Stack, an internal pushdown list maintained by the I/O service routines. The first element of the Input Stack is initialized by the Terminal Monitor Program (TMP), and cannot thereafter be changed or deleted. The TSO-supplied TMP initializes this first element to indicate the terminal as the current input source. The STACK Service Routine adds an element to the input stack or deletes one or more elements from it, and thereby changes the source of input for the other I/O service routines.

This topic describes:

- The List and Execute forms of the STACK macro instruction.
- The Sources of input.
- The STACK Parameter Block.
- The List Source Descriptor.
- Return codes from STACK.

Coding examples are included where needed.

The STACK Macro Instruction - List Form

The list form of the STACK macro instruction builds and initializes a STACK Parameter Block (STPB), according to the operands you specify in the macro. The STACK parameter Block indicates to the STACK service routine which functions you want performed. Figure 38 shows the list form of the STACK macro instruction; each of the operands is explained following the figure. Appendix B describes the notation used to define macro instructions.

```
┌───────────┬────────┬─────────────────────────────────────────────────────────┐
│           │        │ ┌TERM=*                                  ┐              │
│           │        │ │                                        │              │
│           │        │ │                         (,SOURCE)      │              │
│ [symbol]  │ STACK  │ │STORAGE=(element address{,PROCN  })     │ ,MF=L        │
│           │        │ │                         (,PROCL )      │              │
│           │        │ │                                        │              │
│           │        │ │DELETE=(TOP )                           │              │
│           │        │ │       {PROC}                           │              │
│           │        │ └       (ALL )                           ┘              │
└───────────┴────────┴─────────────────────────────────────────────────────────┘
```

Figure 38. The List Form of the STACK Macro Instruction

TERM=*
    Add a terminal element to the input stack.

STORAGE=element address
    Add an in-storage element to the input stack. The element address is the address of the List Source Descriptor (LSD). The LSD is a control block, pointed to by the Stack Parameter Block, which describes the in-storage list. The in-storage element must be further defined as a SOURCE, PROCN, or PROCL list. SOURCE is the default.

SOURCE
    The element to be added to the Input Stack is an in-storage source data set.

PROCN

The element to be added to the Input Stack is a command procedure and NOLIST option has been specified.

PROCL

The element to be added to the Input Stack is a command procedure and the LIST option has been specified. Each line read from the command procedure is written to the terminal.

DELETE=

Delete an element or elements from the Input Stack. The element to be deleted must be further defined as TOP, PROC, or ALL.

TOP

The topmost element (the element most recently added to the Input Stack) is to be deleted.

PROC

The current procedure element is to be deleted from the Input Stack. If the top element is not a PROC element, all elements down to and including the first PROC element encountered are to be deleted.

ALL

All elements are to be deleted from the Input Stack except the bottom element (the first element).

MF=L

Indicates that this is the List form of the macro instruction.

NOTE: In the List form of the macro instruction, only the following is required:

```
STACK MF=L
```

The other operands and their sublists are optional because they may be supplied by the execute form of the macro instruction:

```
TERM=*

 or

                          (,SOURCE )
STORAGE=(element address <,PROCN   > )
                          (,PROCL  )
 or
        (TOP )
DELETE=< PROC>
        (ALL )
```

The operands you specify in the list form of the STACK macro instruction set up control information used by the STACK Service Routine. The TERM=*, STORAGE=, and DELETE= operands set bits in the STACK Parameter Block. These bit settings indicate to the STACK Service Routine which options you wish performed.

The STACK Macro Instruction - Execute Form

Use the execute form of the STACK macro instruction to perform the following three functions:

1. You can use it to set up the Input Output Parameter List (IOPL).

2. You can use it to initilize those fields of the STACK Parameter Block not initialized by the list form of the macro instruction, or to modify those fields already initialized.

3. You use it to pass control to the STACK Service Routine which modifies the Input Stack.

Figure 39 shows the Execute form of the STACK macro instruction; each of the operands is explained following the figure. Appendix B describes the notation used to define macro instructions.

```
[symbol]   STACK   [PARM=parameter address][,UPT=upt address]

                   [,ECT=ect address][,ECB=ecb address]

                    ┌                                         ┐
                    │ ,TERM=*                                 │
                    │                                ⎧,SOURCE⎫│
                    │ ,STORAGE=(element address⎨,PROCN  ⎬)│
                    │                                ⎩,PROCL  ⎭│
                    │                                         │
                    │           ⎧TOP ⎫                        │
                    │ ,DELETE=⎨PROC⎬                        │
                    │           ⎩ALL ⎭                        │
                    └                                         ┘

                   [,ENTRY=⎧entry address⎫]   ,MF=(E,⎧list address⎫)
                   [        ⎩    (15)     ⎭]          ⎩    (1)     ⎭
```

Figure 39. The Execute form of the STACK Macro Instruction

PARM=parameter address
    Specifies the address of the 2-word STACK Parameter Block (STPB). It may be the address of the list form STACK macro instruction. The address is any address valid in an RX instruction, or the number of one of the general registers 2-12 enclosed in parentheses. This address will be placed in the Input Output Parameter List (IOPL).

UPT=upt address
    Specifies the address of the User Profile Table (UPT). This address may be obtained from the Command Processor Parameter List pointed to by register one when the Command Processor is attached by the Terminal Monitor Program. The address may be any address valid in an RX instruction or the number of one of the general registers 2-12 enclosed in parentheses. This address will be placed in the Input Output Parameter List (IOPL).

ECT=ect address
    Specifies the address of the Environment Control Table (ECT). This address may be obtained from the CPPL pointed to by register 1 when the Command Processor is attached by the Terminal Monitor Program. The address may be any address valid in an RX instruction or the number of one of the general registers 2-12 enclosed in parentheses. This address will be placed in the IOPL.

ECB=ecb address

Specifies the address of an Event Control Block (ECB). This address will be placed into the IOPL. You must provide a one-word Event Control Block and pass its address to the STACK service routine by placing it into the IOPL. The address may be any address valid in an RX instruction or the number of one of the general registers 2-12 enclosed in parentheses.

TERM=*

Add a terminal element to the Input Stack.

STORAGE=element address

Add an in-storage element to the Input Stack. The element address is the address of the List Source Descriptor (LSD). The LSD is a control block, pointed to by the Stack Parameter Block, which describes the in-storage list. The in-storage list must be further defined as a SOURCE, PROCN, or PROCL list. SOURCE is the default.

SOURCE

The element to be added to the Input Stack is an in-storage source data set.

PROCN

The element to be added to the Input Stack is a command procedure and the NOLIST option has been specified.

PROCL

The element to be added to the Input Stack is a command procedure and the LIST option has been specified. Each line read from the command procedure is written to the terminal.

DELETE

Delete one or more elements from the input stack. Specify which element, either TOP, PROC, or ALL.

TOP

The topmost element (the element most recently added to the input stack) is to be deleted.

PROC

The current procedure element is to be deleted from the input stack. If the top element is not a procedure element, all elements down to and including the first procedure element encountered are to be deleted.

ALL

All elements are to be deleted from the input stack except the bottom element (the first element).

ENTRY=entry address or (15)

Specifies the entry point of the STACK service routine. The address may be any address valid in an RX instruction or (15) if the entry point address has been loaded into general register 15. If ENTRY is omitted, a LINK macro instruction will be generated to invoke the STACK Service Routine.

MF=E

Indicates that this is the Execute form of the macro instruction.

listaddr
(1)
The address of the 4-word Input Output Parameter List (IOPL). This may be a completed IOPL that you have built, or it may be 4 words of declared storage that will be filled from the PARM, UPT, ECT, and ECB operands of this Execute form of the STACK macro instruction. The address is any address valid in an RX instruction or (1) if the parameter list address has been loaded into general register 1.

NOTE: In the Execute form of the STACK macro instruction only the following operands are required:

```
STACK MF=(E,{list address})
            {    (1)     }
```

The PARM=, UPT=, ECT=, and ECB= operands are not required if you have built an IOPL in your own code.

The other operands and their sublists are optional because they may be supplied by the list form of the macro instruction:

```
TERM=*
 or
                          {,SOURCE}
STORAGE=(element address  {,PROCN }})
                          {,PROCL }
 or
         {TOP }
DELETE=  {PROC}
         {ALL }
```

The ENTRY= operand need not be coded in the macro instruction. If it is not, a LINK macro instruction will be generated to invoke the I/O Service routine.

The operands you specify in the execute form of the STACK macro instruction are used to set up control information used by the STACK service routine. You can use the PARM=, UPT=, ECT=, and ECB= operands of the STACK macro instruction to complete, build, or alter an IOPL. The TERM=*, STORAGE=, and DELETE= operands set bits in the STACK Parameter Block. These bit settings indicate to the STACK Service Routine which options you want.

Sources of Input

The input sources provided are defined as follows:

1. Terminal.
   If the terminal is specified in the STACK macro instruction as the input source, all input and output requests through GETLINE, PUTLINE, and PUTGET are read from the terminal and written to the terminal. The user at the terminal controls the Time Sharing Option by entering commands; the system processes these commands as they are entered; and returns to the user for another command.

2. In-Storage List
   An in-storage list can be either a list of commands or a source data set. It may contain variable length records (with a length header) or fixed length records (no header and all records the same length). In either case, no one record on an In-storage list may exceed 256 characters.

   The in-storage list can be specified as one of two types through the PROC or SOURCE parameters of the STACK macro instruction.

   - PROC - Indicates that the in-storage list is a command procedure -- a list of commands to be executed in the order specified. If you specify PROC, requests through GETLINE are read from the in-storage list, but PROMPT requests from the executing command processor are suppressed. MODE messages, those messages normally sent to the terminal requesting entry of a command or a sub-command, are not sent but a command is obtained from the in-storage list. If the LIST option was specified in the STACK macro instruction when the command procedure was added to the input stack, the command is displayed at the terminal.

   - SOURCE - Indicates that the in-storage list is a source data set. Requests through GETLINE are read from the in-storage list, but PROMPT requests from the executing command processor are honored if prompting is allowed, and a line is requested from the terminal. MODE messages are handled the same way as with PROC. No LIST facility is provided with SOURCE records.

Building the STACK Parameter Block

When the list form of the STACK macro instruction expands, it builds a two word STACK Parameter Block (STPB). The list form of the macro instruction initializes this STPB according to the operands you have coded. This initialized block, which you may later modify with the execute form of the macro instruction, indicates to the I/O service routine the functions you want performed.

By using the list form of the macro instruction to initialize the block, and the execute form to modify it, you can use the same STPB to perform different STACK functions. Keep in mind however, that if you specify an operand in the execute form of the macro instruction, and that operand has a sublist as a value, the default values of the sublist will be coded into the STPB for any of the sublist values not coded. If you do not want the default values you must code each of the values you require, each time you change any one of them.

As an example: If you code the list form of the STACK macro instruction as follows:

```
STACK    STORAGE=(element address,PROCN),MF=L
```

and then override it with the execute form of the macro instruction as follows:

```
STACK    STORAGE=(new element address),MF=(E,list address)
```

The element code in the STACK Parameter Block would default to SOURCE, the default value. If the new in-storage list was another PROCN list, you would have to respecify PROCN in the execute form of the macro instruction.

The STACK Parameter Block is defined by the IKJSTPB DSECT. Figure 40 describes the contents of the STPB.

Number of Bytes	Field	Contents or Meaning
1	none	Operation code: A flag byte which describes the operation to be performed.
	1...	One element is to be added to the top of the Input Stack.
	.1..	The top element is to be deleted from the Input Stack.
	..1.	The current procedure element is to be deleted from the Input Stack. If the top element is not a PROC element, all elements down to and including the first PROC element encountered are deleted, except the bottom element.
	...1	All elements except the bottom one (the first element) are to be deleted.
	 xxxx	Reserved bits.
1	none	Element code: A flag byte describing the element to be added to the Input Stack.
	1...	A terminal element.
	.1..	An in-storage element.
	.... ..0.	The in-storage element is a source element.
	.... ..1.	The in-storage element is a procedure element.
	1	The list option (PROCL) has been specified.
	..xx xx..	Reserved bits.
2		Reserved
4	STPBALSD	The address of the List Source Descriptor (LSD). An LSD describes an in-storage list. If the input source is the terminal, or if DELETE has been specified, this field will contain zeros.

Figure 40. The STACK Parameter Block

If the TERM or DELETE operands have been coded in the STACK macro instruction, the second word of the Stack Parameter Block will contain zeros and the control block structure will end with the STPB. Figure 41 describes this condition.

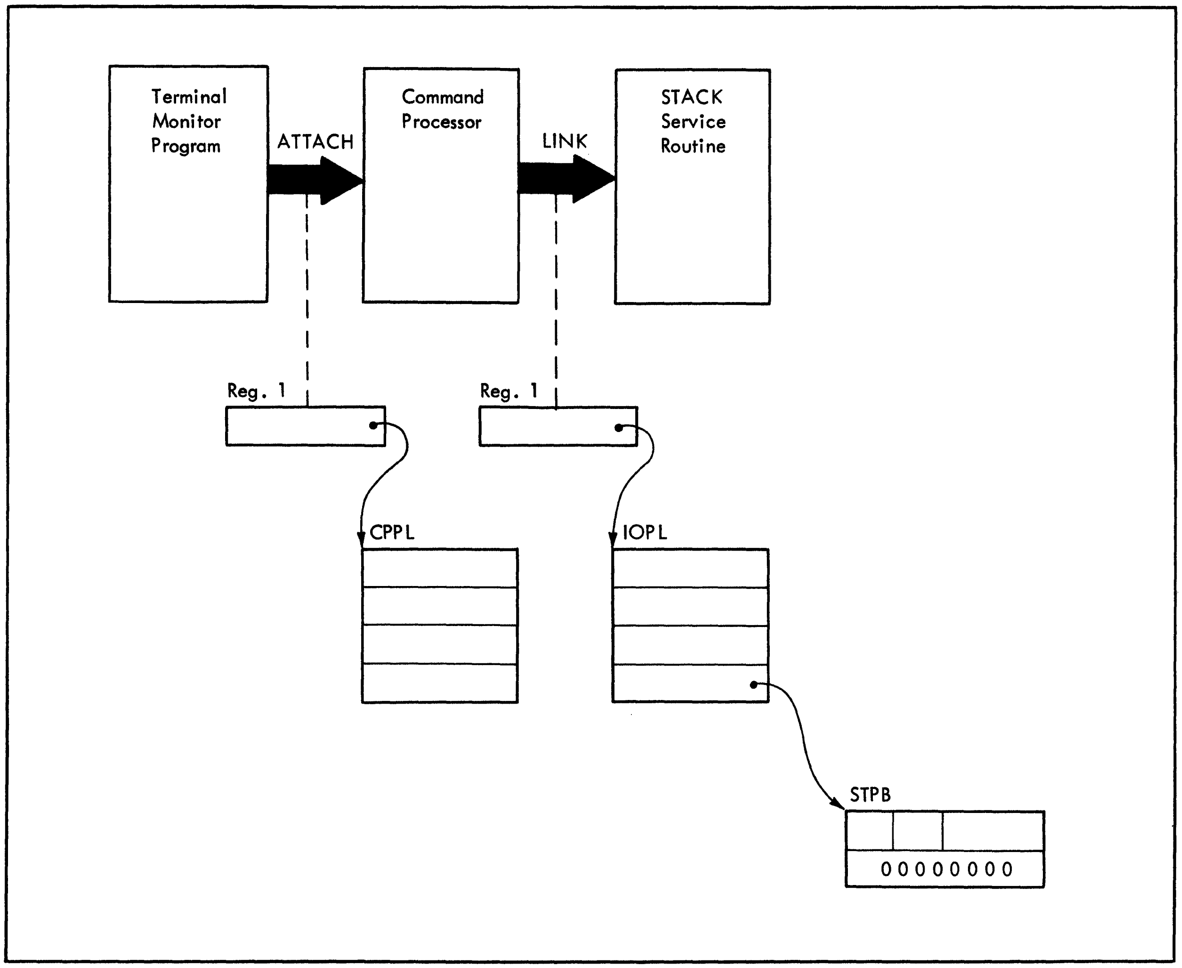
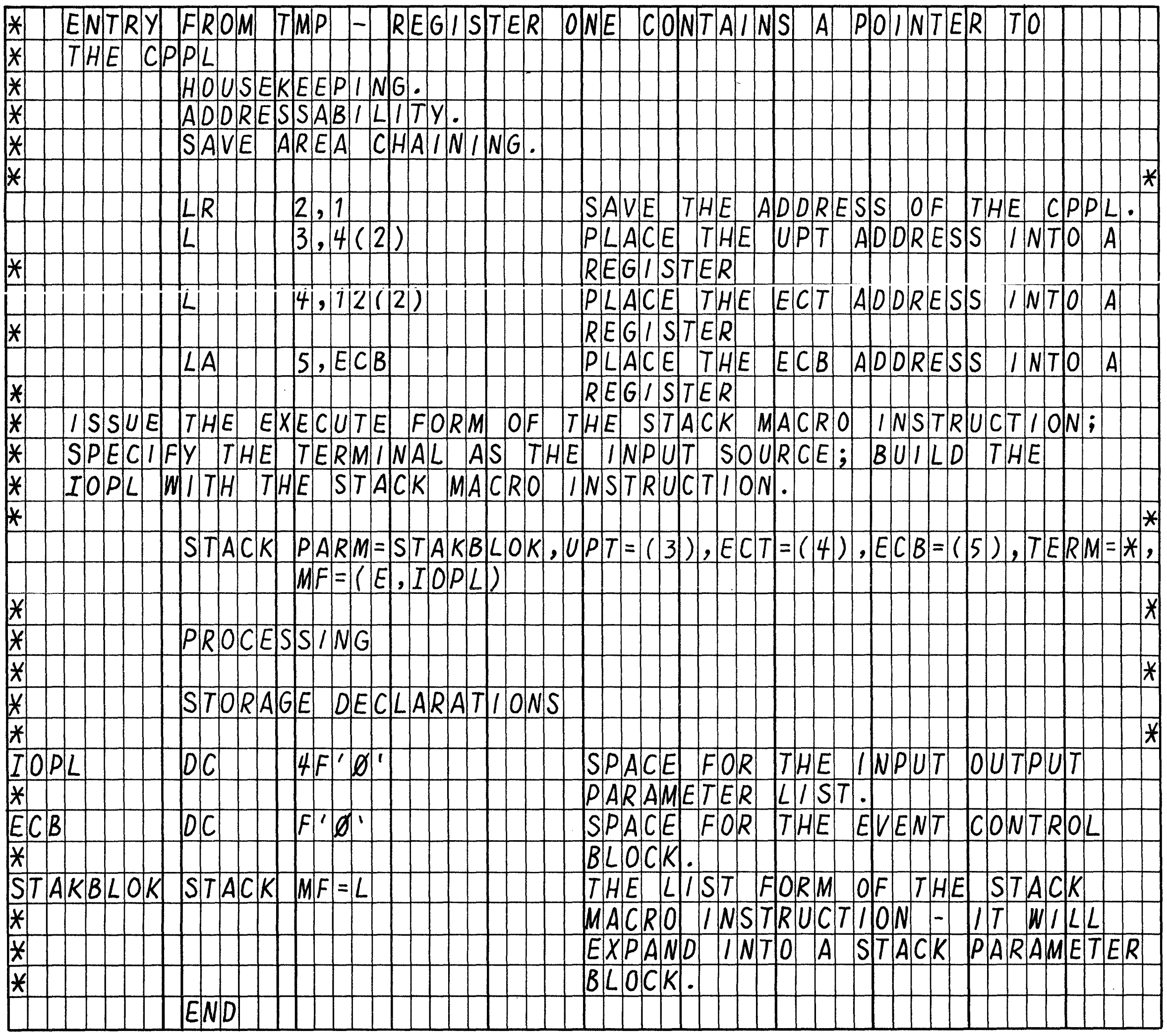


Figure 41. STACK Control Blocks: No In-Storage List

To add an in-storage list element to the input stack, you must describe the in-storage list and pass a pointer to it to the STACK I/O service routine. You do this by building a List Source Descriptor (LSD).

Figure 42 is an example of the code required to add the terminal to the input stack as the current input source. In this example, the execute form of the STACK macro instruction is used to build the Input Output Parameter list for you. The list form of the STACK macro instruction expands into a STACK Parameter Block, and its address is passed to the execute form of the macro instruction as the PARM operand address.



```
*  ENTRY FROM TMP - REGISTER ONE CONTAINS A POINTER TO
*  THE CPPL
*        HOUSEKEEPING.
*        ADDRESSABILITY.
*        SAVE AREA CHAINING.
*                                                           *
         LR    2,1             SAVE THE ADDRESS OF THE CPPL.
         L     3,4(2)          PLACE THE UPT ADDRESS INTO A
*                              REGISTER
         L     4,12(2)         PLACE THE ECT ADDRESS INTO A
*                              REGISTER
         LA    5,ECB           PLACE THE ECB ADDRESS INTO A
*                              REGISTER
*  ISSUE THE EXECUTE FORM OF THE STACK MACRO INSTRUCTION;
*  SPECIFY THE TERMINAL AS THE INPUT SOURCE; BUILD THE
*  IOPL WITH THE STACK MACRO INSTRUCTION.
*                                                           *
         STACK PARM=STAKBLOK,UPT=(3),ECT=(4),ECB=(5),TERM=*,
               MF=(E,IOPL)
*                                                           *
*        PROCESSING
*                                                           *
*        STORAGE DECLARATIONS
*                                                           *
IOPL     DC    4F'Ø'           SPACE FOR THE INPUT OUTPUT
*                              PARAMETER LIST.
ECB      DC    F'Ø'            SPACE FOR THE EVENT CONTROL
*                              BLOCK.
STAKBLOK STACK MF=L            THE LIST FORM OF THE STACK
*                              MACRO INSTRUCTION - IT WILL
*                              EXPAND INTO A STACK PARAMETER
*                              BLOCK.
         END
```

Figure 42. Coding Example -- STACK Specifying the Terminal as the Input Source

This sequence of code does not make use of the IKJCPPL DSECT to access the Command Processor Parameter List, nor does it provide reenterable code.

Building the List Source Descriptor (LSD)

A List Source Descriptor (LSD) is a four word control block which describes the in-storage list pointed to by the new element you are adding to the Input Stack. If you are designating the Terminal as the input source, no LSD is necessary and the second word of the STPB will be zero. If you specify STORAGE as the input source in the STACK macro instruction, your code must build an LSD, and place a pointer to it as a sublist of the STORAGE operand. The LSD must begin on a double word boundary, and must be created in the shared subpool designated by the Terminal Monitor Program; the IBM-supplied TMP shares subpool 78 with the Command Processors. The LSD is defined by the IKJLSD DSECT. Figure 43 describes the contents of the LSD.

Number of Bytes	Field	Contents or Meaning
4	LSDADATA	The address of the in-storage list.
2	LSDRCLEN	The record length if the in-storage list contains fixed length records. Zero if the record lengths are variable.
2	LSDTOTLN	The total length of the in-storage list; the sum of the lengths of all records in the list.
4	LSDANEXT	Pointer to the next record to be processed. Initialize this field to the address of the first record in the list. The field is updated by the GETLINE and PUTGET service routines.
4	LSDRSVRD	Reserved

Figure 43. The List Source Descriptor

If you have provided an LSD, and specified the STORAGE operand in the STACK macro instruction, the second word of the Stack Parameter Block will contain the address of the LSD, and the STACK control block structure will look like Figure 44.

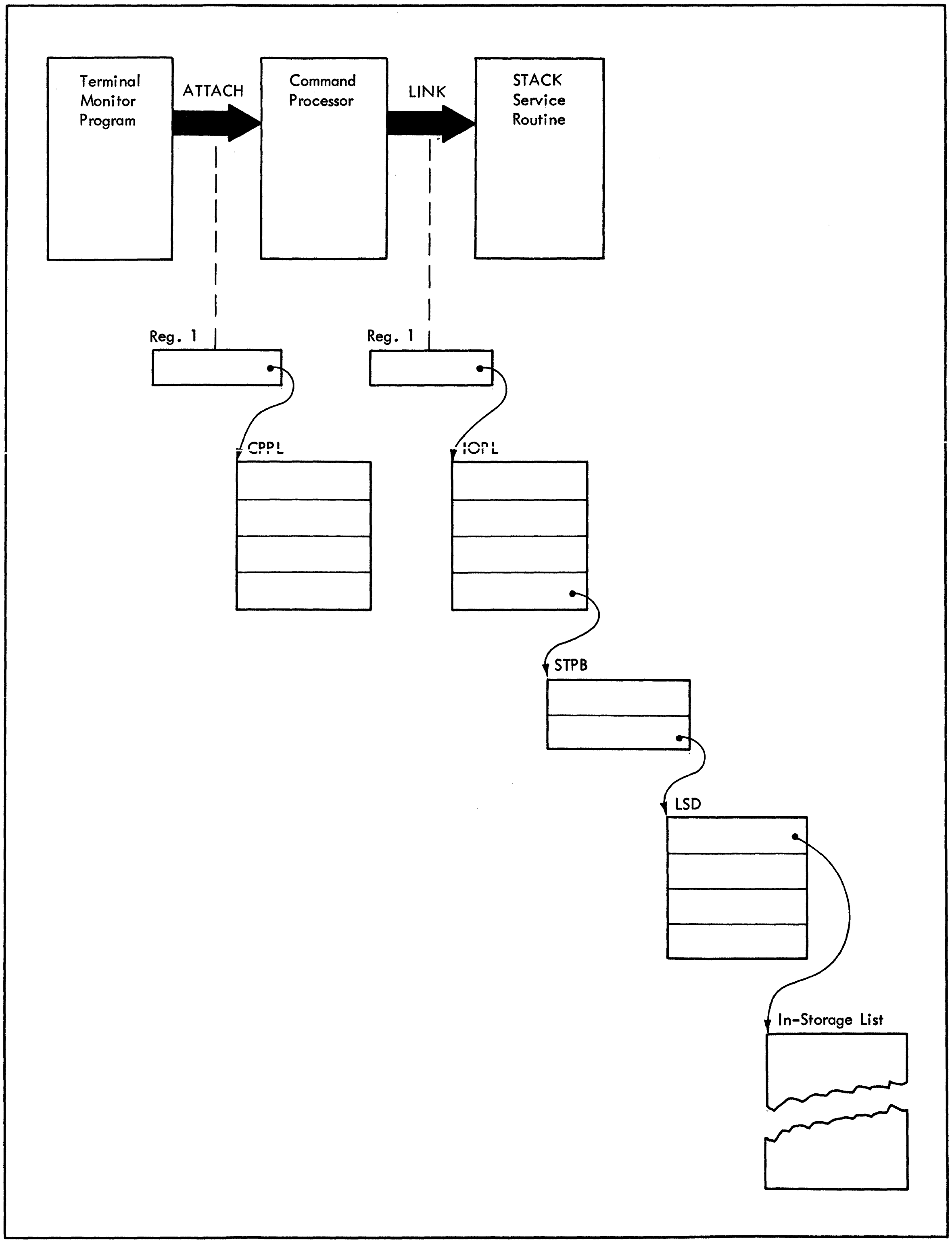
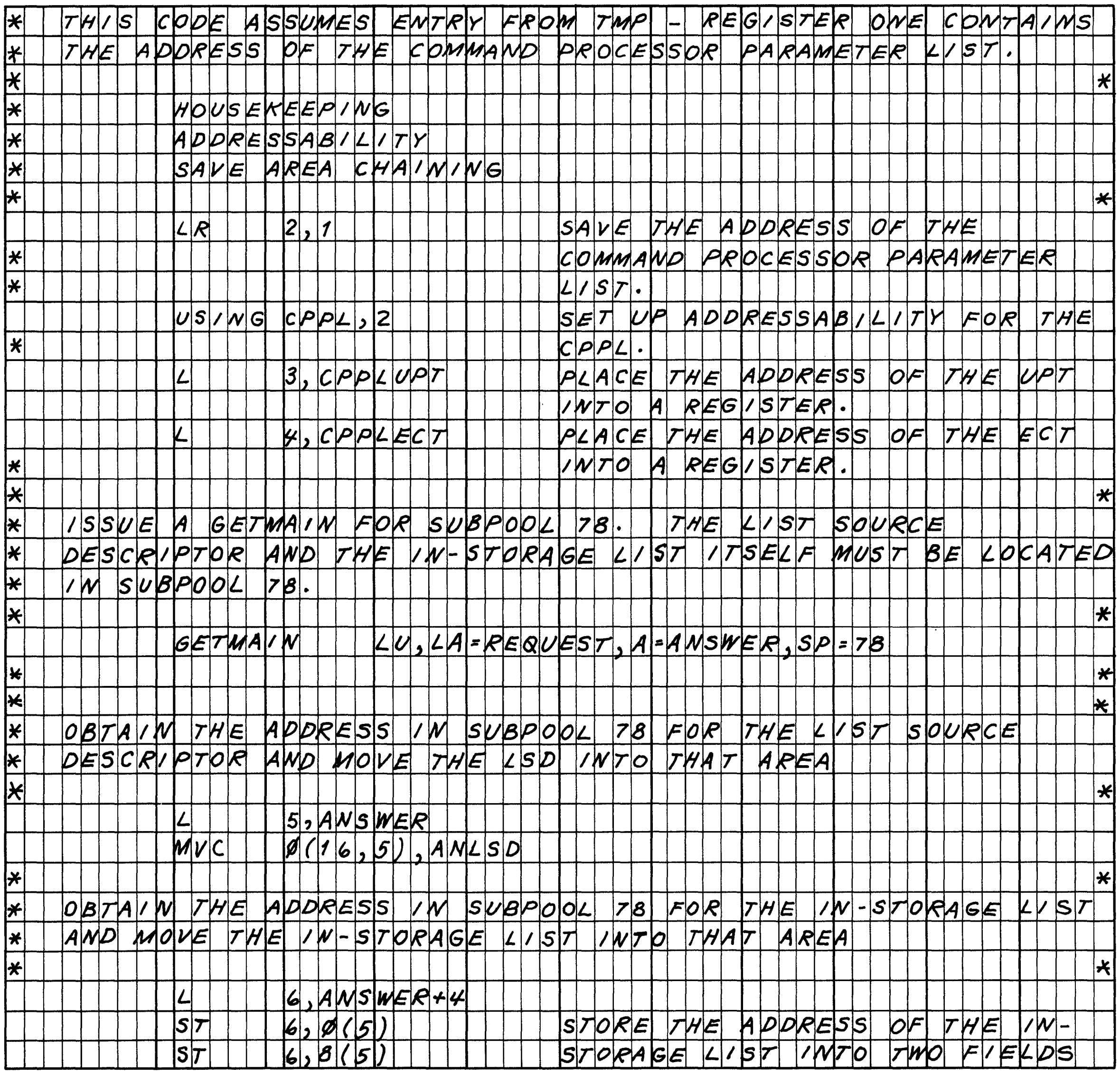


Figure 44. STACK Control Blocks: In-Storage List Specified

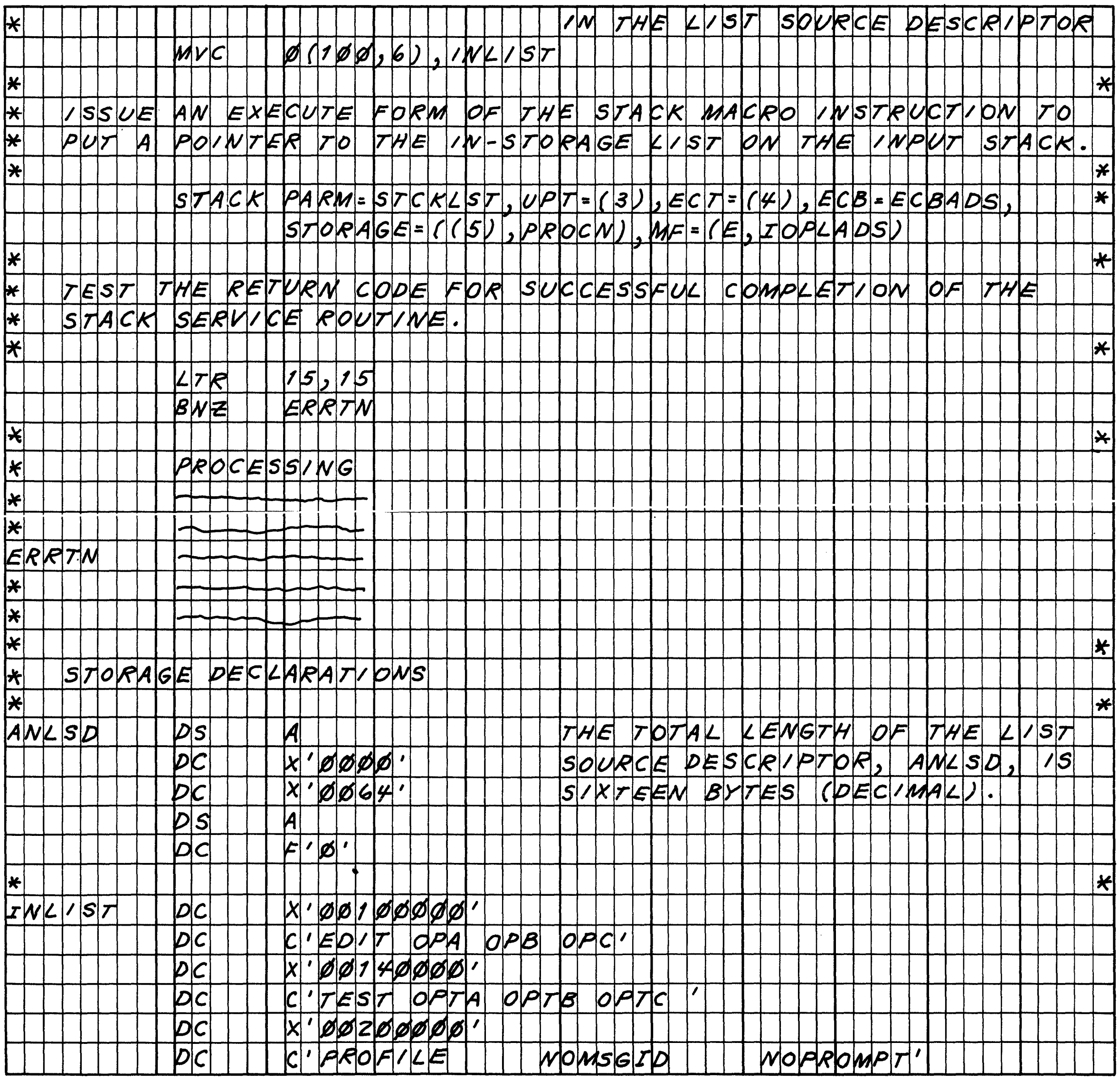
Figure 45 is an example of the code required to use the STACK macro instruction to place a pointer to an in-storage list on the input stack.

In the example, the GETMAIN macro instruction is used to obtain storage in subpool 78 for the List Source Descriptor and the in-storage list itself. The execute form of the STACK macro instruction initializes the Input Output Parameter List required by the STACK service routine. The list form of the STACK macro instruction expands into a STACK Parameter Block, and its address is passed to the STACK service routine via the PARM operand in the execute form of the STACK macro instruction.



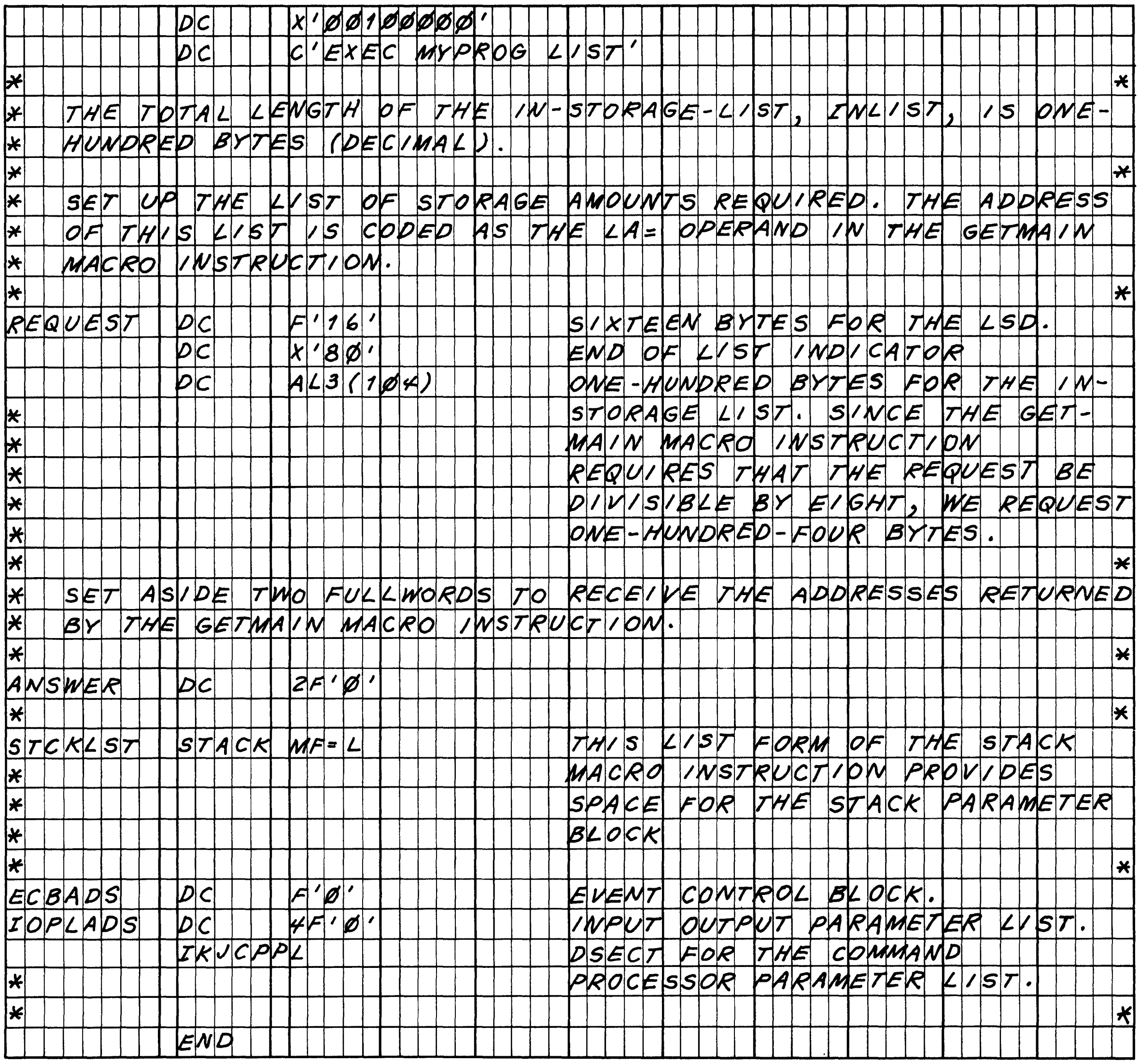
```
*  THIS CODE ASSUMES ENTRY FROM TMP - REGISTER ONE CONTAINS
*  THE ADDRESS OF THE COMMAND PROCESSOR PARAMETER LIST.
*                                                          *
*        HOUSEKEEPING
*        ADDRESSABILITY
*        SAVE AREA CHAINING
*                                                          *
         LR    2,1            SAVE THE ADDRESS OF THE
*                             COMMAND PROCESSOR PARAMETER
*                             LIST.
         USING CPPL,2         SET UP ADDRESSABILITY FOR THE
*                             CPPL.
         L     3,CPPLUPT      PLACE THE ADDRESS OF THE UPT
                              INTO A REGISTER.
         L     4,CPPLECT      PLACE THE ADDRESS OF THE ECT
*                             INTO A REGISTER.
*                                                          *
*  ISSUE A GETMAIN FOR SUBPOOL 78.  THE LIST SOURCE
*  DESCRIPTOR AND THE IN-STORAGE LIST ITSELF MUST BE LOCATED
*  IN SUBPOOL 78.
*                                                          *
         GETMAIN    LU,LA=REQUEST,A=ANSWER,SP=78
*                                                          *
*                                                          *
*  OBTAIN THE ADDRESS IN SUBPOOL 78 FOR THE LIST SOURCE
*  DESCRIPTOR AND MOVE THE LSD INTO THAT AREA
*                                                          *
         L     5,ANSWER
         MVC   Ø(16,5),ANLSD
*                                                          *
*  OBTAIN THE ADDRESS IN SUBPOOL 78 FOR THE IN-STORAGE LIST
*  AND MOVE THE IN-STORAGE LIST INTO THAT AREA
*                                                          *
         L     6,ANSWER+4
         ST    6,Ø(5)         STORE THE ADDRESS OF THE IN-
         ST    6,8(5)         STORAGE LIST INTO TWO FIELDS
```

Figure 45. Coding Example -- STACK Specifying an In-Storage List as the Input Source (Part 1 of 3)



```
*                                        IN THE LIST SOURCE DESCRIPTOR
         MVC   Ø(1ØØ,6),INLIST
*                                                                       *
*   ISSUE AN EXECUTE FORM OF THE STACK MACRO INSTRUCTION TO
*   PUT A POINTER TO THE IN-STORAGE LIST ON THE INPUT STACK.
*                                                                       *
         STACK PARM=STCKLST,UPT=(3),ECT=(4),ECB=ECBADS,                 *
               STORAGE=((5),PROCN),MF=(E,IOPLADS)
*                                                                       *
*   TEST THE RETURN CODE FOR SUCCESSFUL COMPLETION OF THE
*   STACK SERVICE ROUTINE.
*                                                                       *
         LTR   15,15
         BNZ   ERRTN
*                                                                       *
*        PROCESSING
*
*
ERRTN
*
*
*                                                                       *
*   STORAGE DECLARATIONS
*                                                                       *
ANLSD    DS    A                         THE TOTAL LENGTH OF THE LIST
         DC    X'ØØØØ'                   SOURCE DESCRIPTOR, ANLSD, IS
         DC    X'ØØ64'                   SIXTEEN BYTES (DECIMAL).
         DS    A
         DC    F'Ø'
*                                                                       *
INLIST   DC    X'ØØ1ØØØØØ'
         DC    C'EDIT OPA OPB OPC'
         DC    X'ØØ14ØØØØ'
         DC    C'TEST OPTA OPTB OPTC '
         DC    X'ØØ2ØØØØØ'
         DC    C'PROFILE    NOMSGID    NOPROMPT'
```

Figure 45. Coding Example -- STACK Specifying an In-Storage List as the Input Source (Part 2 of 3)



```
         DC    X'ØØ1ØØØØØ'
         DC    C'EXEC MYPROG LIST'
*                                                          *
*  THE TOTAL LENGTH OF THE IN-STORAGE-LIST, INLIST, IS ONE-
*  HUNDRED BYTES (DECIMAL).
*                                                          *
*  SET UP THE LIST OF STORAGE AMOUNTS REQUIRED. THE ADDRESS
*  OF THIS LIST IS CODED AS THE LA= OPERAND IN THE GETMAIN
*  MACRO INSTRUCTION.
*                                                          *
REQUEST  DC    F'16'          SIXTEEN BYTES FOR THE LSD.
         DC    X'8Ø'          END OF LIST INDICATOR
         DC    AL3(1Ø4)       ONE-HUNDRED BYTES FOR THE IN-
*                             STORAGE LIST. SINCE THE GET-
*                             MAIN MACRO INSTRUCTION
*                             REQUIRES THAT THE REQUEST BE
*                             DIVISIBLE BY EIGHT, WE REQUEST
*                             ONE-HUNDRED-FOUR BYTES.
*                                                          *
*  SET ASIDE TWO FULLWORDS TO RECEIVE THE ADDRESSES RETURNED
*  BY THE GETMAIN MACRO INSTRUCTION.
*                                                          *
ANSWER   DC    2F'Ø'
*                                                          *
STCKLST  STACK MF=L           THIS LIST FORM OF THE STACK
*                             MACRO INSTRUCTION PROVIDES
*                             SPACE FOR THE STACK PARAMETER
*                             BLOCK
*                                                          *
ECBADS   DC    F'Ø'           EVENT CONTROL BLOCK.
IOPLADS  DC    4F'Ø'          INPUT OUTPUT PARAMETER LIST.
         IKJCPPL              DSECT FOR THE COMMAND
*                             PROCESSOR PARAMETER LIST.
*                                                          *
         END
```

Figure 45. Coding Example -- STACK Specifying an In-Storage List as the Input Source (Part 3 of 3)

Return Codes From STACK

When it returns to the program which invoked it, the STACK Service Routine will provide one of the following return codes in general register fifteen:

Code	Meaning
0	STACK has completed sucessfully
4	One or more of the parameters passed to STACK were invalid.

GETLINE - GETTING A LINE OF INPUT

You use the GETLINE macro instruction to obtain all input lines other than commands or subcommands, and PROMPT message responses. Commands, subcommands, and PROMPT message responses should be obtained with the PUTGET macro instruction.

When a GETLINE macro instruction is executed, a line is obtained from the current source of input - the terminal or an in-storage list - or optionally, from the terminal, regardless of the current source of input. The processing of the input line varies according to several factors. Included in these factors are the source of input, and the options you specify for logical or physical processing of the input line. The GETLINE Service Routine determines the type of processing to be performed from the operands coded in the GETLINE macro instruction, and returns a line of input.

This topic describes:

- The list and execute forms of the GETLINE macro instruction.
- The sources of input.
- The GETLINE Parameter Block.
- The input line format.
- Examples of GETLINE.
- Return codes from GETLINE.

The GETLINE Macro Instruction - List Form

The list form of the GETLINE macro instruction builds and initializes a GETLINE Parameter Block (GTPB), according to the operands you specify in the GETLINE macro. The GETLINE Parameter Block indicates to the GETLINE service routine which functions you want performed. Figure 46 shows the list form of the GETLINE macro instruction; each of the operands is explained following the figure. Appendix B describes the notation used to define macro instructions.

```
┌──────────┬──────────┬─────────────────────────────────────────────────┐
│ [symbol] │ GETLINE  │┌INPUT=({ISTACK}{,LOGICAL }) ┐                   │
│          │          │└       {TERM  }{,PHYSICAL}) ┘                   │
│          │          │                                                 │
│          │          │┌,TERMGET=({EDIT}{,WAIT  })┐,MF=L                │
│          │          │└          {ASIS}{,NOWAIT})┘                     │
└──────────┴──────────┴─────────────────────────────────────────────────┘
```

Figure 46. The List Form of the GETLINE Macro Instruction

INPUT=
: Indicates that an input line is to be obtained. That input line is further described by the INPUT sublist operands ISTACK, TERM, LOGICAL, and PHYSICAL. ISTACK and LOGICAL are the default values.

ISTACK
: Obtain an input line from the currently active input source indicated by the input stack.

TERM
: Obtain an input line from the terminal. If TERM is coded in the macro instruction, the input stack is ignored and regardless of the currently active input source, a line is returned from the terminal.

LOGICAL
: The input line to be obtained is a logical line; the GETLINE service routine is to perform logical line processing.

PHYSICAL
: The input line to be obtained is a physical line. The GETLINE service routine need not inspect the input line.

NOTE: If the input line you are requesting is a Logical line coming from the input source indicated by the input stack, you need not code the INPUT operand or its sub-list operands. The input line description defaults to ISTACK, LOGICAL.

TERMGET
: Specifies the TGET options requested. GETLINE issues a TGET SVC to bring in a line of data from the terminal, this operand indicates to the TGET SVC which of the TGET options to use. The TGET options are EDIT or ASIS, and WAIT or NOWAIT. The default values are EDIT and WAIT.

EDIT
: Specifies that in addition to minimal editing (see ASIS), the buffer is to be filled out with trailing blanks.

ASIS

Specifies that minimal editing is to be done as follows:

a. Transmission control characters are removed.

b. The line of input is translated from terminal code to EBCDIC.

c. Line deletion and character deletion editing is performed.

d. Line feed and carriage return characters, if present, are removed.

WAIT

Specifies that control is to be returned to the routine that issued the GETLINE macro instruction only after an input message has been read.

NOWAIT

Specifies that control is to be returned to the routine that issued the GETLINE macro instruction whether or not a line of input is available. If a line of input is not available, a return code of 12 decimal is returned in register 15 to the command processor.

MF=L

Indicates that this is the list form of the macro instruction.

<u>NOTE</u>: In the list form of the macro instruction, only

```
GETLINE  MF=L
```

is required. The other operands and their sublists are optional because they may be supplied by the execute form of the macro instruction, or automatically supplied if you want the default values:

```
INPUT=( {ISTACK} {,LOGICAL } )
        {TERM  } {,PHYSICAL}

 and

,TERMGET=( {EDIT} {,WAIT  } )
           {ASIS} {,NOWAIT}
```

The operands you specify in the list form of the GETLINE macro instruction set up control information used by the GETLINE service routine. The INPUT= and TERMGET= operands set bits in the GETLINE Parameter Block to indicate to the GETLINE service routine which options you want performed.

The GETLINE Macro Instruction - Execute Form

Use the execute form of the GETLINE macro instruction to perform the following three functions:

1. You may use it to set up the Input Output Parameter List (IOPL).

2. You may use it to initialize those fields of the GETLINE Parameter Block (GTPB) not initialized by the List form of the macro instruction, or to modify those fields already initialized.

3. You use it to pass control to the GETLINE service routine which gets the line of input.

Figure 47 shows the execute form of the GETLINE macro instruction; each of the operands is explained following the figure. Appendix B describes the notation used to define macro instructions.

```
┌───────────┬──────────┬─────────────────────────────────────────────────┐
│ [symbol]  │ GETLINE  │[PARM=parameter address][,UPT=upt address]       │
│           │          │                                                 │
│           │          │[,ECT=ect address][,ECB=ecb address]             │
│           │          │                                                 │
│           │          │┌                                     ┐          │
│           │          ││,INPUT=( ISTACK  ,LOGICAL  )         │          │
│           │          ││         TERM    ,PHYSICAL           │          │
│           │          │└                                     ┘          │
│           │          │┌                                   ┐            │
│           │          ││,TERMGET=( EDIT  ,WAIT   )         │            │
│           │          ││           ASIS  ,NOWAIT           │            │
│           │          │└                                   ┘            │
│           │          │┌                        ┐                        │
│           │          ││,ENTRY= entry address   │,MF=(E, list address ) │
│           │          ││           (15)         │          (1)          │
└───────────┴──────────┴└────────────────────────┘───────────────────────┘
```

Figure 47. The Execute Form of the GETLINE Macro Instruction

PARM=parameter address
Specifies the address of the 2-word GETLINE Parameter Block (GTPB). It may be the address of a list form GETLINE macro instruction. The address is any address valid in an RX instruction, or the number of one of the general registers 2-12 enclosed in parentheses. This address will be placed in the Input Output Parameter List (IOPL).

UPT=upt address
Specifies the address of the User Profile Table (UPT). You may obtain this address from the Command Processor Parameter List pointed to by register one when the command processor is attached by the Terminal Monitor Program. The address may be any address valid in an RX instruction or the number of one of the general registers 2-12 enclosed in parentheses. This address will be placed in the IOPL.

ECT=ect address
Specifies the address of the Environment Control Table (ECT). You may obtain this address from the CPPL pointed to by register 1 when the Command Processor is attached by the Terminal Monitor Program. The address may be any address valid in an RX instruction or the number of one of the general registers 2-12 enclosed in parentheses. This address will be placed into the IOPL.

ECB=ecb address

Specifies the address of an Event Control Block (ECB). You must provide a one-word Event Control Block and pass its address to the GETLINE Service Routine by placing it into the IOPL. The address may be any address valid in an RX instruction or the number of one of the general registers 2-12 enclosed in parentheses. This address will be placed into the IOPL.

INPUT=

Indicates that an input line is to be obtained. This input line is further described by the INPUT sublist operands ISTACK, TERM, LOGICAL, and PHYSICAL. ISTACK and LOGICAL are the default values.

ISTACK

Obtain an input line from the currently active input source indicated by the input stack.

TERM

Obtain an input line from the terminal. If TERM is coded in the macro instruction, the input stack will be ignored and regardless of the currently active input source, a line is returned from the terminal.

LOGICAL

The input line to be obtained is a logical line; the GETLINE service routine is to perform logical line processing. (See Glossary for the definition of "logical line.")

PHYSICAL

The input line to be obtained is a physical line. The GETLINE service routine need not inspect the input line.

NOTE: If the input line you are requesting is a Logical line coming from the input source indicated by the input stack, you need not code the INPUT operand or its sublist operands. The input line description defaults to ISTACK, LOGICAL.

TERMGET

Specifies the TGET options requested. GETLINE issues a TGET SVC to bring in a line of data from the terminal, this operand indicates to the TGET SVC which of the TGET options to use. The TGET options are EDIT or ASIS, and WAIT or NOWAIT. The default values are EDIT and WAIT.

EDIT

Specifies that in addition to minimal editing (see ASIS), the input buffer is to be filled out with trailing blanks.

ASIS

Specifies that minimal editing is to be done by the TGET SVC. The following editing functions will be performed by TGET:

a. Transmission control characters are removed.

b. The line of input is translated from terminal code to EBCDIC.

c. Line deletion and character deletion editing are performed.

d. Line feed and carriage return characters, if present, are removed.

WAIT

Specifies that control is to be returned to the routine that issued the GETLINE macro instruction, only after an input message has been read.

NOWAIT

Specifies that control is to be returned to the routine that issued the GETLINE macro instruction whether or not a line of input is available. If a line of input is not available, a return code of 12 decimal is returned in register 15 to the command processor.

ENTRY=entry address or (15)

Specifies the entry point of the GETLINE service routine. If ENTRY is omitted, a LINK macro instruction will be generated to invoke the GETLINE service routine. The address may be any address valid in an RX instruction or (15) if the entry point address has been loaded into general register 15.

MF=E

Indicates that this is the execute form of the macro instruction.

listaddr
(1)

The address of the 4-word Output Parameter List (IOPL). This may be a completed IOPL that you have built, or it may be 4 words of declared storage that will be filled from the PARM, UPT, ECT, and ECT operands of this execute form of the GETLINE macro instruction. The address is any address valid in an RX instruction or (1) if the parameter list address has been loaded into general register 1.

NOTE: In the execute form of the GETLINE macro instruction only the following is required:

```
┌──────────────────────────────────────────────────────────────────┐
|GETLINE  MF=(E,{list address})                                    |
|               {   (1)      }                                     |
└──────────────────────────────────────────────────────────────────┘
```

The PARM=, UPT=, ECT=, and ECB= operands are not required if you have built your IOPL in your own code.

The other operands and their sublists are optional because you may have supplied them in the list form of the macro instruction or in a previous execution of GETLINE, or because you are using the default values:

```
┌──────────────────────────────────────────────────────────────────┐
|INPUT=({ISTACK}{,LOGICAL })                                        |
|       {TERM  }{,PHYSICAL}                                        |
|                                                                  |
| and                                                              |
|                                                                  |
|TERMGET=({EDIT}{,WAIT  })                                         |
|         {ASIS}{,NOWAIT}                                          |
└──────────────────────────────────────────────────────────────────┘
```

The ENTRY= operand need not be coded in the macro instruction. If it is not, a LINK macro instruction will be generated to invoke the I/O service routine.

The operands you specify in the execute form of of the GETLINE macro instruction are used to set up control information used by the GETLINE Service Routine. You can use the PARM=, UPT=, ECT=, and ECB= operands of the GETLINE macro instruction to build, complete, or modify an IOPL. The INPUT= and TERMGET= operands set bits in the GETLINE Parameter Block. These bit settings indicate to the GETLINE Service Routine which options you want performed.

Sources of Input

There are two sources of input provided; they are the terminal and an in-storage list.

TERMINAL: Input comes from the terminal under either of the following conditions:

- You have specified the terminal as the input source by including the TERM operand in the GETLINE macro instruction.
- You have specified the current element of the Input Stack by including the ISTACK operand in the GETLINE macro instruction, and the current element is a terminal element.

If you specify terminal as the input source, you have the option of requesting the GETLINE Service Routine to process the input as a logical or physical line by including the LOGICAL or the PHYSICAL operand in the macro instruction. LOGICAL is the default value.

Physical Line Processing: A physical line is a line which is returned to the requesting program exactly as it is received from the input source. The contents of the line are not inspected by the GETLINE service routine.

Logical Line Processing: A logical line is a line which has had additional processing by the GETLINE service routine before it is returned to the requesting program. If logical line processing is requested, each line returned to the routine that issued the GETLINE is inspected to see if the last character of the line is a continuation mark (a dash '-'). A continuation mark signals GETLINE to get another line from the terminal and to concatenate that line with the line previously obtained. The continuation mark is overlaid with the first character of the new line.

IN-STORAGE LIST: If the top element of the input stack is an in-storage list, and you do not specify TERM in the GETLINE macro instruction, the line will be obtained from the in-storage list. The in-storage list is a resident data set which has been previously made available to the I/O Service Routines with the STACK Service Routine. No logical line processing is performed on the lines because it is assumed that each line in the in-storage list is a logical line. It is also assumed that no single record has a length greater than 256 bytes.

End of Data Processing

If you issue a GETLINE macro against an in-storage list from which all the records have already been read, GETLINE senses an end of data (EOD) condition. GETLINE deletes the top element from the Input Stack and passes a return code of 16 in register 15. Return code 16 indicates that no line of input has been returned by the GETLINE service routine. You can use this EOD code (16) as an indication that all input from a particular source has been exhausted and no more GETLINE macro instructions should be issued against this input source. If you reissue a GETLINE macro instruction against the input stack after a return code of 16, a record will be returned from the next input source indicated by the input STACK. You can identify the source of this record by the return code (0 = terminal, 4 = in-storage).

Building the GETLINE Parameter Block

When the list form of the GETLINE macro instruction expands, it builds a two word GETLINE Parameter Block (GTPB). The list form of the macro instruction initializes this GTPB according to the operands you have coded in the macro instruction. This initialized block, which you may later modify with the execute form of the macro instruction, indicates to the GETLINE Service Routine the function you want performed.

You must supply the address of the GTPB to the Execute form of the GETLINE macro instruction. For non-reenterable programs you can do this simply by placing a symbolic name in the symbol field of the list form of the macro instruction, and passing this symbolic name to the execute form of the macro instruction as the PARM value. The GETLINE Parameter Block is defined by the IKJGTPB DSECT. Figure 48 describes the contents of the GTPB.

Number of Bytes	Field	Contents or Meaning
2		Control flags. These bits describe the requested input line to the GETLINE service routine.
	Byte 1	
	..0.	The input line is a logical line.
	..1.	The input line is a physical line.
	...0	The input line is to be obtained from the current input source indicated by the input stack.
	...1	The input line is to be obtained from the terminal.
	xx.. xxxx	Reserved bits.
	Byte 2	
	xxxx xxxx	Reserved.
2		TGET options field. These bits indicate to the TGET SVC which of the TGET options you want to use.
	Byte 1	
	1...	Always set to 1 for TGET.
	...0	WAIT processing has been requested. Control will be returned to the issuer of GETLINE only after an input message has been read.
	...1	NOWAIT processing has been requested. Control will be returned to the issuer of the GETLINE macro instruction whether or not a line of input is available.
	00	EDIT processing has been requested. In addition to the editing provided by ASIS processing, the input buffer is to be filled out with training blanks to the next double-word boundary.
	01	ASIS processing has been requested. (See the ASIS operand of the GETLINE macro instruction description).
	.xx. xx..	Reserved bits.
	Byte 2	
	xxxx xxxx	Reserved.
4	GTPBIBUF	The address of the input buffer. The GETLINE service routine fills this field with the address of the input buffer in which the input line has been placed.

Figure 48. The GETLINE Parameter Block

Input Line Format - The Input Buffer

The second word of the GETLINE Parameter Block contains zeros until the GETLINE service routine returns a line of input. The service routine places the requested input line into an input buffer beginning on a double word boundary located in subpool 1. It then places the address of this input buffer into the second word of the GTPB. The input buffer belongs to the command processor that issued the GETLINE macro instruction. The buffers returned by GETLINE are automatically freed when your C.P. relinquishes control. If space is a consideration, you should free the input buffer with the FREEMAIN macro instruction after you have processed or copied the input line.

Regardless of the source of input, an in-storage list or the terminal, the input line returned to the command processor by the GETLINE Service Routine is in a standard format. All input lines are in a variable length record format with a full-word header followed by the text returned by GETLINE. Figure 49 shows the format of the input buffer returned by the GETLINE service routine.

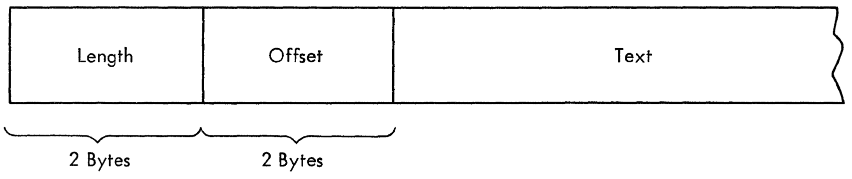


Figure 49. Format of the GETLINE Input Buffer

The two-byte length field contains the length of the input line including the header length (4 bytes). You can use this length field to determine the length of the input line to be processed, and later, to free the input buffer with the R form of the FREEMAIN macro instruction.

The two-byte offset field is always set to zero on return from the GETLINE Service Routine.

Figure 50 shows the GETLINE control block structure after the GETLINE Service Routine has returned an input line.

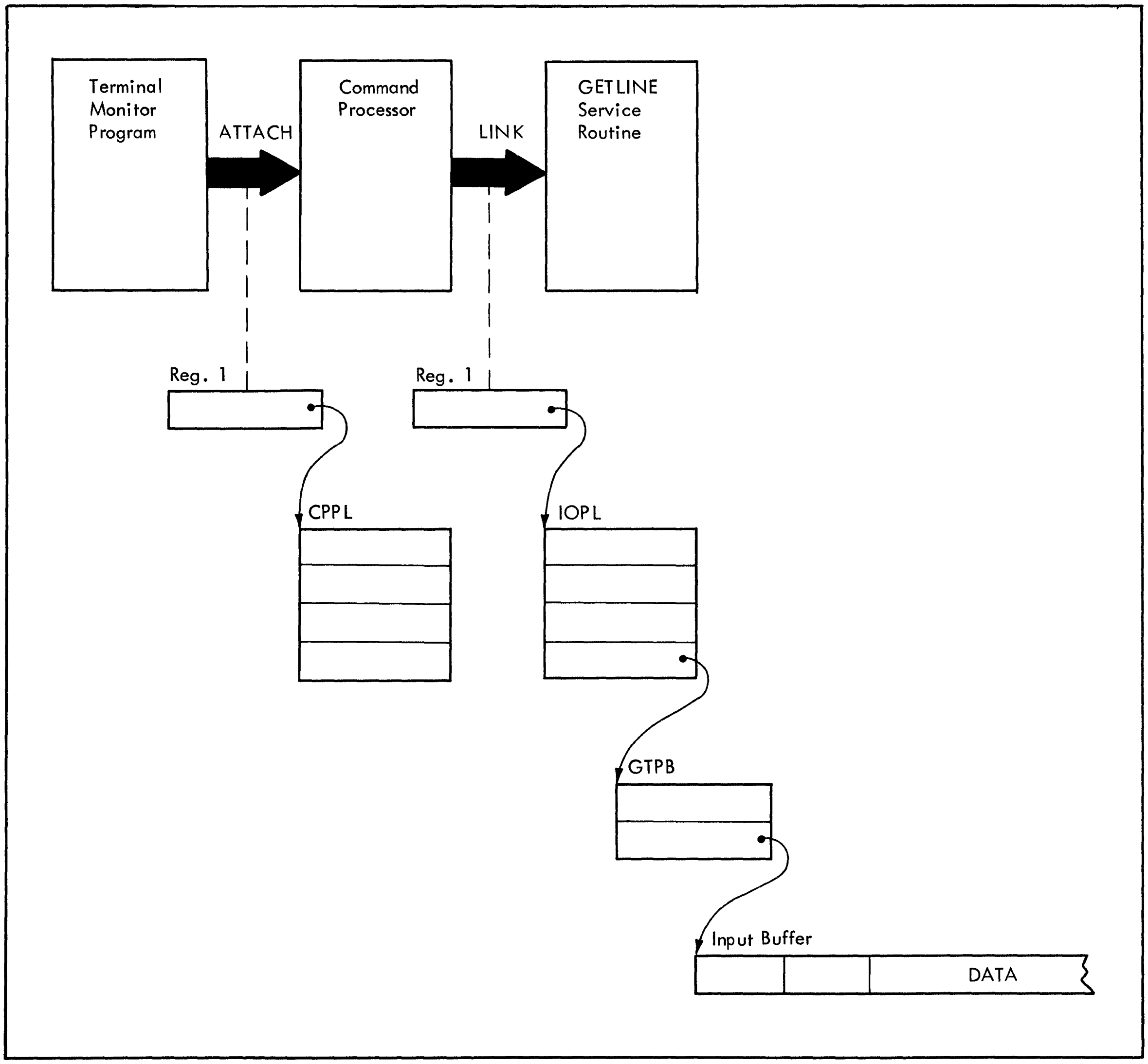


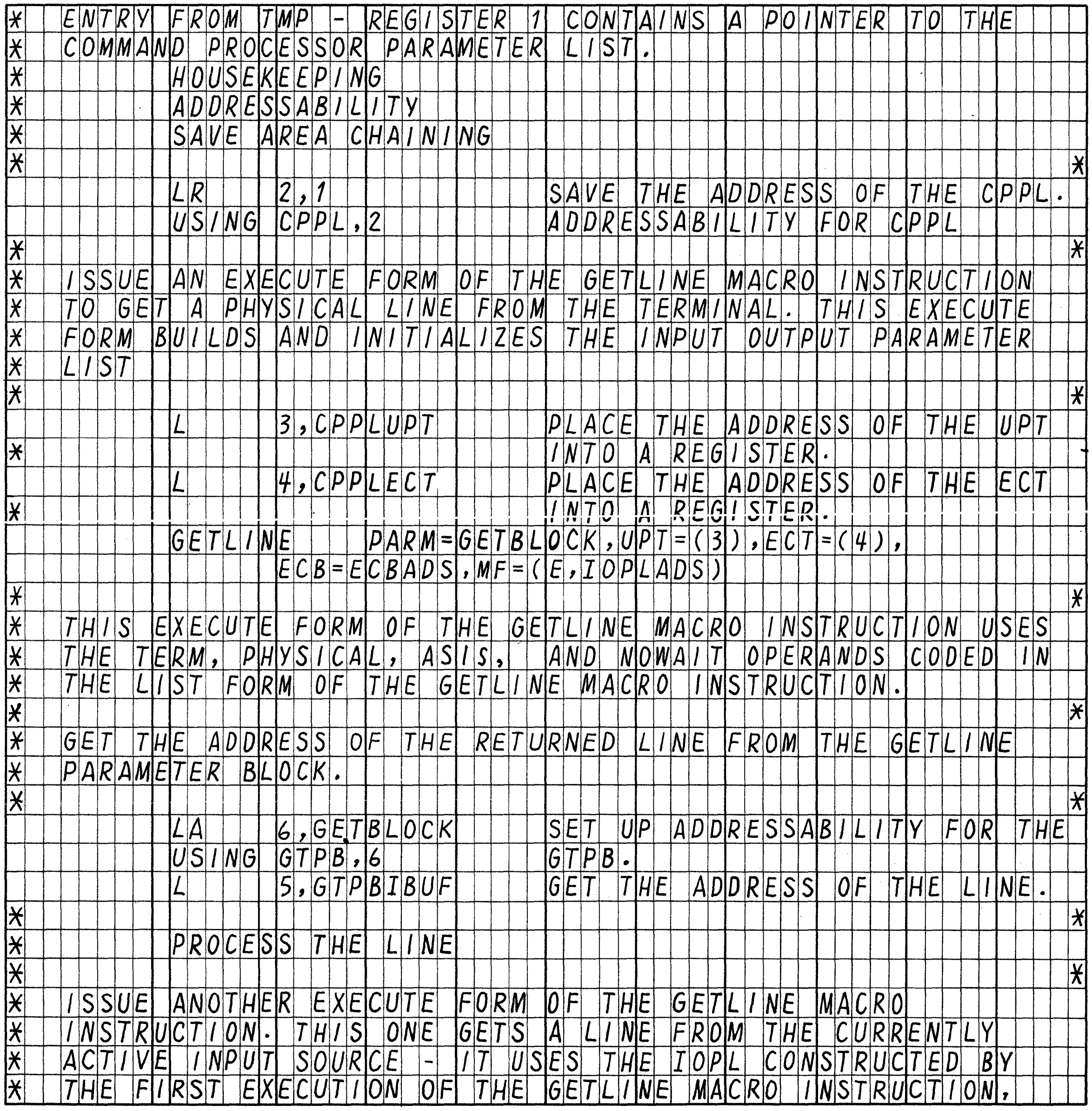
Figure 50. GETLINE Control Blocks - Input Line Returned

Examples of GETLINE

Figure 51 is an example of the code required to execute the GETLINE macro instruction. In this example two execute forms of the GETLINE macro instruction are issued. The first one builds the IOPL, and uses the parameters initialized by the list form of the macro instruction to get a physical line from the terminal with the NOWAIT and ASIS options.

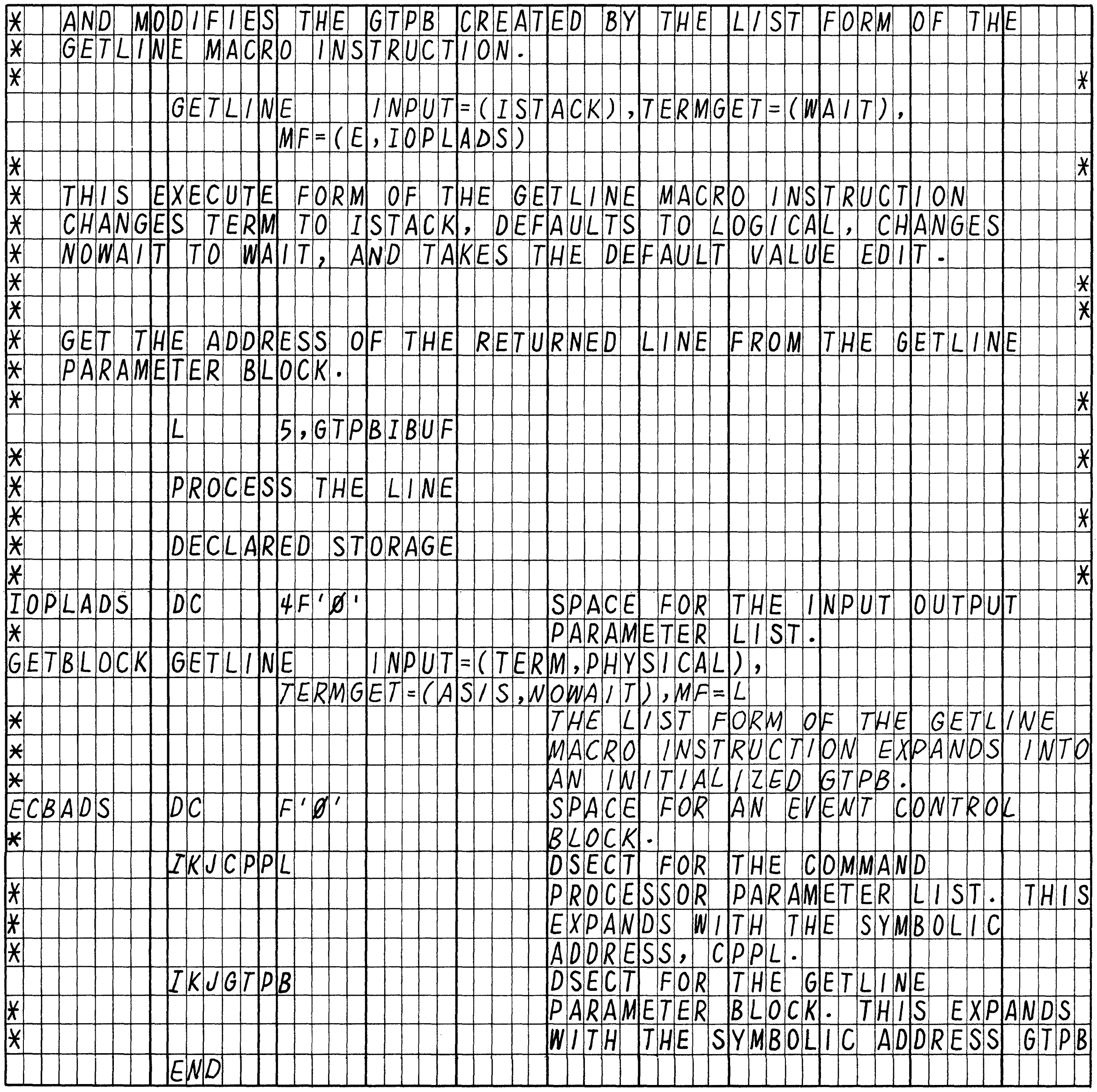
In the second execution of the GETLINE macro instruction, the same IOPL is used, but the GETLINE options are changed from TERM to ISTACK, and from NOWAIT to WAIT explicitly, and from PHYSICAL to LOGICAL and from ASIS to EDIT by default.

Notice also that the IKJCPPL DSECT is used to map the Command Processor Parameter List, and the IKJGTPB DSECT is used to map the GETLINE Parameter Block.



```
*  ENTRY FROM TMP - REGISTER 1 CONTAINS A POINTER TO THE
*  COMMAND PROCESSOR PARAMETER LIST.
*       HOUSEKEEPING
*       ADDRESSABILITY
*       SAVE AREA CHAINING
*                                                  *
        LR    2,1         SAVE THE ADDRESS OF THE CPPL.
        USING CPPL,2      ADDRESSABILITY FOR CPPL
*                                                  *
*  ISSUE AN EXECUTE FORM OF THE GETLINE MACRO INSTRUCTION
*  TO GET A PHYSICAL LINE FROM THE TERMINAL. THIS EXECUTE
*  FORM BUILDS AND INITIALIZES THE INPUT OUTPUT PARAMETER
*  LIST
*                                                  *
        L     3,CPPLUPT   PLACE THE ADDRESS OF THE UPT
*                         INTO A REGISTER.
        L     4,CPPLECT   PLACE THE ADDRESS OF THE ECT
*                         INTO A REGISTER.
        GETLINE  PARM=GETBLOCK,UPT=(3),ECT=(4),
              ECB=ECBADS,MF=(E,IOPLADS)
*                                                  *
*  THIS EXECUTE FORM OF THE GETLINE MACRO INSTRUCTION USES
*  THE TERM, PHYSICAL, ASIS, AND NOWAIT OPERANDS CODED IN
*  THE LIST FORM OF THE GETLINE MACRO INSTRUCTION.
*                                                  *
*  GET THE ADDRESS OF THE RETURNED LINE FROM THE GETLINE
*  PARAMETER BLOCK.
*                                                  *
        LA    6,GETBLOCK  SET UP ADDRESSABILITY FOR THE
        USING GTPB,6      GTPB.
        L     5,GTPBIBUF  GET THE ADDRESS OF THE LINE.
*                                                  *
*       PROCESS THE LINE
*                                                  *
*  ISSUE ANOTHER EXECUTE FORM OF THE GETLINE MACRO
*  INSTRUCTION. THIS ONE GETS A LINE FROM THE CURRENTLY
*  ACTIVE INPUT SOURCE - IT USES THE IOPL CONSTRUCTED BY
*  THE FIRST EXECUTION OF THE GETLINE MACRO INSTRUCTION,
```

Figure 51. Coding Example -- Two Executions of GETLINE (Part 1 of 2)



```
*  AND MODIFIES THE GTPB CREATED BY THE LIST FORM OF THE
*  GETLINE MACRO INSTRUCTION.
*                                                          *
         GETLINE    INPUT=(ISTACK),TERMGET=(WAIT),
               MF=(E,IOPLADS)
*                                                          *
*  THIS EXECUTE FORM OF THE GETLINE MACRO INSTRUCTION
*  CHANGES TERM TO ISTACK, DEFAULTS TO LOGICAL, CHANGES
*  NOWAIT TO WAIT, AND TAKES THE DEFAULT VALUE EDIT.
*                                                          *
*                                                          *
*  GET THE ADDRESS OF THE RETURNED LINE FROM THE GETLINE
*  PARAMETER BLOCK.
*                                                          *
         L     5,GTPBIBUF
*                                                          *
*        PROCESS THE LINE
*                                                          *
*        DECLARED STORAGE
*                                                          *
IOPLADS  DC    4F'Ø'          SPACE FOR THE INPUT OUTPUT
*                             PARAMETER LIST.
GETBLOCK GETLINE    INPUT=(TERM,PHYSICAL),
               TERMGET=(ASIS,NOWAIT),MF=L
*                             THE LIST FORM OF THE GETLINE
*                             MACRO INSTRUCTION EXPANDS INTO
*                             AN INITIALIZED GTPB.
ECBADS   DC    F'Ø'           SPACE FOR AN EVENT CONTROL
*                             BLOCK.
         IKJCPPL              DSECT FOR THE COMMAND
*                             PROCESSOR PARAMETER LIST. THIS
*                             EXPANDS WITH THE SYMBOLIC
*                             ADDRESS, CPPL.
         IKJGTPB              DSECT FOR THE GETLINE
*                             PARAMETER BLOCK. THIS EXPANDS
*                             WITH THE SYMBOLIC ADDRESS GTPB
         END
```

Figure 51. Coding Example -- Two Executions of GETLINE (Part 2 of 2)

Return Codes from GETLINE

When it returns to the program that invoked it, the GETLINE service routine returns one of the following codes in general register fifteen:

CODE	MEANING
0	GETLINE has completed successfully. The line was obtained from the terminal.
4	GETLINE has completed successfully. The line was returned from an in-storage list.
8	The GETLINE function was not completed. An attention interruption occurred during GETLINE processing, and the user's attention routine turned on the completion bit in the communications ECB.
12	The NOWAIT option was specified and no line was obtained.
16	EOD - An attempt was made to get a line from an in-storage list but the list had been exhausted.
20	Invalid parameters passed to the GETLINE Service Routine.
24	A conditional GETMAIN was issued by GETLINE for input buffers and there was not sufficient space to satisfy the request.

PUTLINE - PUTTING A LINE OUT TO THE TERMINAL

Use the PUTLINE macro instruction to prepare a line and write it to the terminal. Use PUTLINE to put out lines that do not require immediate response from the terminal; use PUTGET to put out lines that require immediate response. The types of lines which do not require response from the terminal are defined as data lines and informational message lines.

The PUTLINE service routine prepares a line for output according to the operands you code into the list and execute forms of the PUTLINE macro instruction. The operands of the macro instruction indicate to the PUTLINE service routine the type of line being put out (data line or informational message line), the type of processing to be performed on the line (format only, second level informational message chaining, text insertion), and the TPUT options requested.

This topic describes:

- The list and execute forms of the PUTLINE macro instruction.
- The PUTLINE Parameter Block.
- The types and formats of output lines.
- PUTLINE message processing.
- Return codes from PUTLINE.

Coding examples are included where needed.

The PUTLINE Macro Instruction - List Form

The list form of the PUTLINE macro instruction builds and initializes a PUTLINE Parameter Block (PTPB), according to the operands you specify in the macro instruction. The PUTLINE Parameter Block indicates to the PUTLINE service routine which functions you want performed. Figure 52 shows the list form of the PUTLINE macro instruction; each of the operands is explained following the figure. Appendix B describes the notation used to define macro instructions.

```
┌────────┬──────────┬────────────────────────────────────────────────────────────┐
│        │          │┌                                  ┌,SINGLE ┐             ┐│
│[symbol]│ PUTLINE  ││OUTPUT=(output address ┌,TERM  ┐ ┤,MULTLVL├ ┌,INFOR┐)││
│        │          │└                       └,FORMAT┘ └,MULTLIN┘ └,DATA ┘  ┘│
│        │          │                                                          │
│        │          │┌             ┌EDIT   ┐                                    ┐│
│        │          ││,TERMPUT=(   ┤ASIS   ├ ┌,WAIT  ┐┌,NOHOLD┐┌,NOBREAK┐)  ││
│        │          │└             └CONTROL┘ └,NOWAIT┘└,HOLD  ┘└,BREAKIN┘   ┘│
│        │          │                                                          │
│        │          │ ,MF=L                                                    │
└────────┴──────────┴────────────────────────────────────────────────────────────┘
```

Figure 52. The List Form of the PUTLINE Macro Instruction

OUTPUT=output address

Indicates that an output line is to be written to the terminal. The type of line provided and the processing to be performed on that line by the PUTLINE service routine are described by the OUTPUT sublist operands TERM, FORMAT, SINGLE, MULTLVL, MULTLIN, INFOR and DATA. The default values are TERM, SINGLE, and INFOR.

The output address differs depending upon whether the output line is an informational message or a data line. For DATA requests, it is the address of the beginning (the full-word header) of a data record to be written to the terminal. For informational message requests (INFOR), it is the address of the Output Line Descriptor. The Output Line Descriptor (OLD) describes the message to be put out, and contains the address of the beginning (the full-word header) of the message or messages to be written to the terminal by the PUTLINE Service Routine.

TERM

Write the line out to the terminal.

FORMAT

The output request is only to format a single message and not to put the message out to the terminal. The PUTLINE Service Routine returns the address of the formatted line by placing it in the third word of the PUTLINE Parameter Block.

SINGLE

The output line is a single line.

MULTLVL

The output message consists of multiple levels. INFOR must be specified.

MULTLIN

The output data consists of multiple lines. DATA must be specified.

INFOR

The output line is an informational message.

DATA

The output line is a data line.

TERMPUT

Specifies the TPUT options requested. PUTLINE issues a TPUT SVC to write the line to the terminal, this operand indicates which of the TPUT options you want to use. The TPUT options are EDIT, ASIS, or CONTROL, WAIT or NOWAIT, NOHOLD or HOLD, and NOBREAK or BREAKIN. The default values are EDIT, WAIT, NOHOLD, and NOBREAK.

EDIT

Specifies that in addition to minimal editing (see ASIS), the following TPUT functions are requested:

a. Any trailing blanks are removed before the line is written to the terminal. If a blank line is sent, the terminal vertically spaces one line.

b. Control characters are added to the end of the output line to position the carrier to the beginning of the next line.

c. All terminal control characters (for example: bypass, restore, horizontal tab, new line) are replaced with a printable character. "Backspace" is an exception; see (d.) under ASIS.

ASIS

Specifies that minimal editing is to be performed by TPUT as follows:

a. The line of output is translated from EBCDIC to terminal code. Invalid characters are converted to a printable character to prevent program caused I/O errors. This does not mean that all unprintable characters are eliminated. "Restore", "upper case", "lower case", "bypass", and "bell ring", for example, might be valid but nonprinting characters at some terminals. (See CONTROL).

b. Transmission control characters are added.

c. EBCDIC "NL", placed at the end of the message, indicates to the TPUT SVC that the carrier is to be returned at the end of the line. "NL" is replaced with whatever is necessary for that particular terminal type to cause the carrier to return. This "NL" processing occurs only if you specify ASIS, and the "NL" is the last character in your message.

   If you specify EDIT, "NL" is handled as described in (c.) under EDIT.

   If the "NL" is embedded in your message, it is sent to the terminal as a carriage return. No idle characters are added (see f. below). This may cause overprinting, particularly on terminals that require a line-feed character to position the carrier on a new line.

d. If you have used "backspace" in your output message, but the "backspace" character does not exist on the terminal type to which the message is being routed, TPUT attempts alternate methods to accomplish the backspace.

e. Control characters are added as needed to cause the message to occur on several lines if the output line is longer than the terminal line size.

f. Idle characters are sent at the end of each line to prevent typing as the carrier returns.

CONTROL
Specifies that the output line is composed of terminal control characters and will not print or move the carrier on the terminal. This option should be used for transmission of characters such as "bypass", "restore", or "bell ring".

WAIT
Specifies that control will not be returned until the output line has been placed into a terminal output buffer.

NOWAIT
Specifies that control should be returned whether or not a terminal output buffer is available. If no buffer is available, a return code of 8 (decimal) will be returned in register 15, to the Command Processor.

NOHOLD
Specifies that the control is to be returned to the routine that issued the PUTLINE macro instruction, and that routine can continue processing as soon as the output line has been placed on the output queue.

HOLD
Specifies that the routine that issued the PUTLINE macro instruction cannot continue its processing until this output line has been put out to the terminal or deleted.

NOBREAK
Specifies that if the terminal user has started to enter input, he is not to be interrupted. The output message is placed on the output queue to be printed after the terminal user has completed the line.

BREAKIN
Specifies that output has precedence over input. If the user at the terminal is transmitting, he is interrupted, and this output line is sent. Any data that was received before the interruption is kept and displayed at the terminal following this output line.

MF=L
Indicates that this is the list form of the macro instruction.

Note: In the list form of the macro instruction, only the following is required:

```
PUTLINE  MF=L
```

The output line address is required for each issuance of the PUTLINE macro instruction but it may be supplied in the execute form of the macro instruction:

```
OUTPUT=(output address)
```

The other operands and sublists are optional because you can supply them in the execute form of the macro instruction, or they may be supplied by the macro expansion if you want the default values:

```
OUTPUT=(        {,TERM  } {,SINGLE } {,INFOR})
                {,FORMAT} {,MULTLVL} {,DATA }
                          {,MULTLIN}

  and

,TERMPUT=({EDIT   } {,WAIT  } {,NOHOLD} {,NOBREAK})
          {ASIS   } {,NOWAIT} {,HOLD  } {,BREAKIN}
          {CONTROL}
```

The operands you specify in the list form of the PUTLINE macro instruction set up control information used by the PUTLINE service routine. This control information is passed to the PUTLINE service routine in the PUTLINE Parameter Block, a three word parameter block built and initialized by the list form of the PUTLINE macro instruction.

The PUTLINE Macro Instruction - Execute Form

Use the execute form of the PUTLINE Macro instruction to put a line or lines out to the terminal, to chain second level messages, and to format a line and return the address of the formatted line to the code that issued the PUTLINE macro instruction. The execute form of the PUTLINE macro instruction performs the following functions:

1. It can be used to set up the Input Output Parameter List (IOPL).
2. It can be used to initialize those fields of the PUTLINE Parameter Block (PTPB) not initialized by the List form of the macro instruction, or to modify those fields already initialized.
3. It passes control to the PUTLINE service routine.

The PUTLINE Service Routine makes use of the IOPL and the PTPB to determine which of the PUTLINE functions you want performed.

Figure 53 shows the execute form of the PUTLINE macro instruction; each of the operands is explained following the figure. Appendix B describes the notation used to define macro instructions.

```
 [symbol]  | PUTLINE  |[PARM=parameter address][,UPT=upt address]
           |          |
           |          |[,ECT=ect address][,ECB=ecb address]
           |          |
           |          |[,OUTPUT=(output address {,TERM  } {,SINGLE }
           |          |[                        {,FORMAT} {,MULTLVL}
           |          |[                                  {,MULTLIN}
           |          |
           |          |                         {,INFOR})]
           |          |                         {,DATA }  ]
           |          |
           |          |[           {EDIT   }                                   ]
           |          |[,TERMPUT=( {ASIS   } {,WAIT  } {,NOHOLD} {,NOBREAK})   ]
           |          |[           {CONTROL} {,NOWAIT} {,HOLD  } {,BREAKIN}    ]
           |          |
           |          |[,ENTRY={entry address}],MF=(E,{list address})
           |          |[       {  (15)        }]      {(1)         }
```

Figure 53. The Execute Form of the PUTLINE Macro Instruction

PARM=parameter address
: Specifies the address of the 2-word PUTLINE Parameter Block (PTPB). It may be the address of a List form PUTLINE macro instruction. The address is any address in an RX instruction, or the number of one of the general registers 2-12 enclosed in parentheses. This address will be placed into the IOPL.

UPT=upt address
: Specifies the address of the User Profile Table (UPT). You may obtain this address from the Command Processor Parameter List (CPPL) pointed to by register one when a Command Processor is attached by the Terminal Monitor Program. The address may be any address valid in an RX instruction or it may be placed in one of the general registers 2-12 and the register number enclosed in parentheses. This address will be placed into the IOPL.

ECT=ect address
: Specifies the address of the Environment Control Table (ECT). You may obtain this address from the CPPL pointed to by register 1 when a command processor is attached by the Terminal Monitor Program. The address may be any address valid in an RX instruction or it may be placed in one of the general registers 2-12 and the register number enclosed in parentheses. This address will be placed into the IOPL.

ECB=ecb address
: Specifies the address of the Event Control Block (ECB). You must provide a one-word event Control Block and pass its address to the PUTLINE service routine. This address will be placed into the IOPL. The address may be any address valid in an RX instruction or it may be placed in one of the general registers 2-12 and the register number enclosed in parentheses.

OUTPUT=output address

Indicates that an output line is provided. The type of line provided and the processing to be performed on that line by the PUTLINE service routine are described by the OUTPUT sublist operands TERM, FORMAT, SINGLE MULTLVL, MULTLIN, INFOR and DATA. The default values are TERM, SINGLE, and INFOR.

The output address differs depending upon whether the output line is an informational message or a data line. For DATA requests, it is the address of the beginning (the full-word header) of a data record to be put out to the terminal. For informational message requests (INFOR), it is the address of the Output Line Descriptor. The Output Line Descriptor (OLD) describes the message to be put out, and contains the address of the beginning (the full-word header) of the message or messages to be written to the terminal by the PUTLINE service routine.

TERM

Write the line out to the terminal.

FORMAT

The output request is only to format a single message and not to put the message out to the terminal. The PUTLINE service routine returns the address of the formatted line by placing it in the third word of the PUTLINE Parameter Block.

SINGLE

The output line is a single line.

MULTLVL

The output message consists of multiple levels. INFOR must be specified.

MULTLIN

The output data consists of multiple lines. DATA must be specified.

INFOR

The output line is an informational message.

DATA

The output line is a data line.

TERMPUT

Specifies the TPUT options requested. PUTLINE issues a TPUT SVC to write the line to the terminal, this operand indicates which of the TPUT options you want to use. The TPUT options are EDIT, ASIS, or CONTROL, WAIT or NOWAIT, NOHOLD or HOLD, and NOBREAK or BREAKIN. The default values are EDIT, WAIT, NOHOLD, and NOBREAK.

EDIT

Specifies that in addition to minimal editing (see ASIS), the following TPUT functions are requested:

a. Any trailing blanks are removed before the line is written to the terminal. If a blank line is sent the terminal vertically spaces one line.

b. Control characters are added to the end of the output line to position the carrier to the beginning of the next line.

c. All terminal control characters (for example: bypass, restore, horizontal tab, new line) are replaced with a printable character. "Backspace" is an exception; see (d.) under ASIS.

ASIS

Specifies that minimal editing is to be performed by TPUT as follows:

a. The line of output is translated from EBCDIC to terminal code. Invalid characters are converted to a printable character to prevent program-caused I/O errors. This does not mean that all unprintable characters are eliminated. "Restore", "upper case", "lower case", "bypass", and "bell ring", for example, may be valid but nonprinting characters at some terminals. (See CONTROL).

b. Transmission control characters are added.

c. EBCDIC "NL", placed at the end of the message, indicates to the TPUT SVC that the carrier is to be returned at the end of the line. "NL" is replaced with whatever is necessary for that particular terminal type to cause the carrier to return. This "NL" processing occurs only if you specify ASIS, and the "NL" is the last character in your message.

   If you specify EDIT, "NL" is handled as described in (c.) under EDIT.

   If the "NL" is embedded in your message, a semicolon is substituted for "NL" and sent to the terminal. No idle characters are added (see f. below). This may cause overprinting, particularly on terminals that require a line-feed character to position the carrier on a new line.

d. If you have used "backspace" in your output message, but the "backspace" character does not exist on the terminal type to which the message is being routed, the PUTLINE service routine attempts alternate methods to accomplish the backspace.

e. Control characters are added as needed to cause the message to occur on several lines if the output line is longer than the terminal line size.

f. Idle characters are sent at the end of each line to prevent typing as the carrier returns.

CONTROL

Specifies that the output line is composed of terminal control characters and will not print or move the carrier on the terminal. This option should be used for transmission of characters such- as "bypass", "restore", or "bell ring".

WAIT

Specifies that control will not be returned until the output line has been placed into a terminal output buffer.

NOWAIT

Specifies that control should be returned whether or not a terminal output buffer is available. If no buffer is available, a return code of 8 (decimal) is returned in register 15.

NOHOLD

Specifies that control is returned to the routine that issued the PUTLINE macro instruction, and it can continue processing, as soon as the output line has been placed on the output queue.

HOLD

Specifies that the module that issued the PUTLINE macro instruction is not to resume processing until the output line has been put out to the terminal or deleted.

NOBREAK

Specifies that if the terminal user has started to enter input, he is not to be interrupted. The output message is placed on the output queue to be printed after the terminal user has completed the line.

BREAKIN

Specifies that output has precendence over input. If the user at the terminal is transmitting, he is interrupted, and the output line is sent. Any data that was received before the interruption is kept and displayed at the terminal following the output line.

ENTRY=entry address or (15)

Specifies the entry point of the PUTLINE service routine. If ENTRY is omitted, the PUTLINE macro expansion will generate a LINK macro instruction to invoke the PUTLINE service routine. The address may be any address valid in an RX instruction or (15) if the entry point address has been loaded into general register 15.

MF=E

Indicates that this is the execute form of the PUTLINE macro instruction.

list address
(1)

The address of the 4-word Input Output Parameter List (IOPL). This may be a completed IOPL that you have built, or 4 words of declared storage to be filled from the PARM, UPT, ECT, and ECB operands of this execute form of the PUTLINE macro instruction. The address is any address valid in an RX instruction or (1) if the parameter list address has been loaded into general register 1.

Note: In the execute form of the PUTLINE macro instruction only the following is required:

```
PUTLINE  MF=(E,{list address})
               {  (1)        }
```

The PARM=, UPT=, ECT=, and ECB= operands are not required if you have built your IOPL in your own code.

The output line address is required for each issuance of the PUTLINE macro instruction, but you may supply it in the list form of the macro instruction:

```
OUTPUT=(output address)
```

The other operands and sublists are optional because you may have supplied them in the list form of the macro instruction or in a previous execute form, or because you may want to use the default values which are automatically supplied by the macro expansion itself. The other operands and sublists are:

```
                                 {,SINGLE }
     OUTPUT=(          {,TERM  } {,MULTLVL} {,INFOR})
                       {,FORMAT} {,MULTLIN} {,DATA }

      and

               {EDIT    }
     ,TERMPUT=({ASIS    }{,WAIT  } {,NOHOLD} {,NOBREAK})
               {CONTROL }{,NOWAIT} {,HOLD  } {,BREAKIN}
```

The ENTRY= operand need not be coded in the macro instruction. If it is not, a LINK macro instruction will be generated by the macro expansion to invoke the I/O service routine.

The operands you specify in the execute form of the PUTLINE macro instruction set up control information used by the PUTLINE service routine. You can use the PARM=, UPT=, ECT=, and ECB=, operands of the PUTLINE macro instruction to build, complete or modify an IOPL. The OUTPUT= and TERMPUT= operands and their sublist operands initialize the PUTLINE Parameter Block. The PUTLINE Parameter Block is referenced by the PUTLINE Service Routine to determine which functions you want PUTLINE to perform.

Building the PUTLINE Parameter Block

When the list form of the PUTLINE macro instruction expands, it builds a three-word PUTLINE Parameter Block (PTPB). The list form of the macro instruction initializes the PTPB according to the operands you have coded in the macro instruction. The initialized block, which you may later modify with the execute form of the PUTLINE macro instruction, indicates to the PUTLINE service routine the function you want performed.

You must supply the address of the PTPB to the execute form of the PUTLINE macro instruction. Since the list form of the macro instruction expands into a PTPB, all you need do is pass the address of the list form of the macro instruction to the execute form as the PARM value.

The PUTLINE Parameter Block is defined by the IKJPTPB DSECT. Figure 54 describes the contents of the PTPB.

Number of Bytes	Field	Contents or Meaning
2		Control flags. These bits describe the output line to the PUTLINE Service Routine.
	Byte 1	
	..0.	The output line is a message.
	..1.	The output line is a data line.
	...1	The output line is a single level or a single line.
	.... 1...	The output is multi-line.
	.... .1..	The output is multi-level.
	.... ..1.	The output line is an informational message.
	xx.. ...x	Reserved bits.
	Byte 2	
	..1.	The format only function was requested.
	xx.x xxxx	Reserved bits.
2		TPUT options field. These bits indicate to the TPUT SVC which of the TPUT options you want to use.
	Byte 1	
	0...	Always set to 0 for TPUT.
	...0	WAIT processing has been requested. Control will be returned to the issuer of PUTLINE only after the output line has been placed into a terminal output buffer.
	...1	NOWAIT processing has been requested. Control will be returned to the issuer of PUTLINE whether or not a terminal output buffer is available.
	.... 0...	NOHOLD processing has been requested. The Command Processor that issued the PUTLINE can resume processing as soon as the output line has been placed on the output queue.
	.... 1...	HOLD processing has been requested. The Command Processor that issued the PUTLINE is not to resume processing until the output line has been written to the terminal or deleted.
	.... .0..	NOBREAK processing has been requested. The output line will be printed only when the terminal user is not entering a line.
	.... .1..	BREAKIN processing has been requested. The output line is to be sent to the terminal immediately. If the terminal user is entering a line, he is to be interrupted.
	00	EDIT processing has been requested.
	01	ASIS processing has been requested.
	10	CONTROL processing has been requested.
	.xx.	Reserved.
	Byte 2	Reserved.

Figure 54. The PUTLINE Parameter Block (Part 1 of 2)

Number of Bytes	Field	Contents or Meaning
4	PTPBOPUT	The address of the OUTPUT Line Descriptor (OLD) if the output line is a message. The address of the fullword header preceding the data if the output line is a single data line. The address of a forward-chain pointer preceding the fullword data header, if the output is multiline data.
4	PTPBFLN	Address of the format only line. The PUTLINE service routine places the address of the formatted line into this field.

Figure 54. The PUTLINE Parameter Block (Part 2 of 2)

Types and Formats of Output Lines

There are two types of output lines processed by the PUTLINE Service Routine:

- Data Lines (DATA)
- Message lines (INFOR)

The OUTPUT sublist operands you specify in the PUTLINE macro instruction indicate to the PUTLINE service routine which type of line you want processed (DATA, INFOR), whether the output consists of one line, several lines, or several levels of messages (SINGLE, MULTLIN, MULTLVL,) and whether the line is to be written to the terminal (TERM), or formatted only (FORMAT).

DATA LINES: A data line is the simplest type of output processed by the PUTLINE Service Routine. It is simply a line of text to be written to the terminal. PUTLINE does not format the line or process it in any way; it merely writes the line, as it appears, out to the terminal. There are two kinds of data lines, single line data and multiline data; each is handled differently by the PUTLINE service routine.

Single Line Data: Single line data is one contiguous character string which PUTLINE places out to the terminal as one logical line. If the line of data you provide exceeds the terminal line length, the TPUT Routine segments the line and puts it out as several terminal lines. PUTLINE accepts single line data in the format shown in Figure 55.

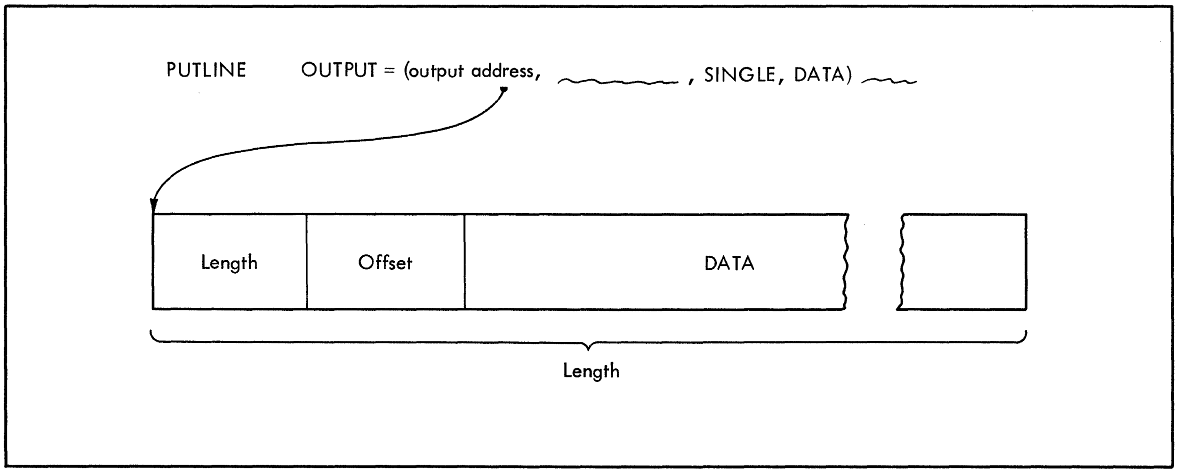
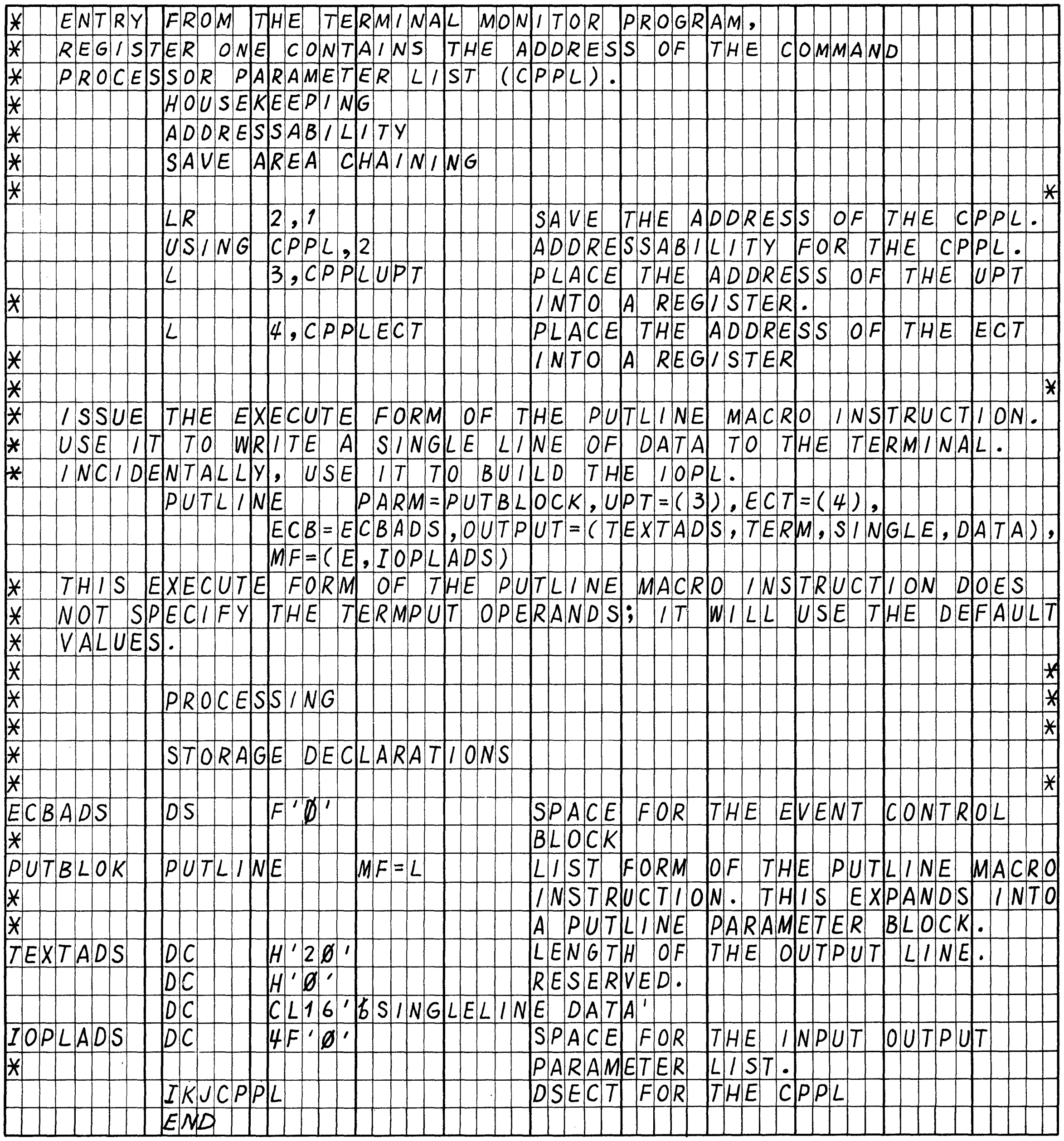


Figure 55. PUTLINE Single Line Data Format

You must precede your line of data with a 4-byte header field. The first two bytes contain the length of the output line, including the header; the second two bytes are reserved for offsets and are set to zero for data lines. You pass the address of the output line to the PUTLINE service routine by coding the beginning address of the four-byte header as the OUTPUT operand address in either the list or the execute form of the macro instruction. When the macro instruction expands, it places this data line address into the second word of the PUTLINE Parameter Block.

Figure 56 is an example of the code that could be used to write a single line of data to the terminal using the PUTLINE macro instruction. Note that the execute form of the PUTLINE macro instruction is used in this example to construct the Input Output Parameter List, and that the TERMPUT operands are not coded in either the list or the execute form of the macro instruction -- the default values will be assumed by the PUTLINE service routine.



```
*  ENTRY FROM THE TERMINAL MONITOR PROGRAM,
*  REGISTER ONE CONTAINS THE ADDRESS OF THE COMMAND
*  PROCESSOR PARAMETER LIST (CPPL).
*        HOUSEKEEPING
*        ADDRESSABILITY
*        SAVE AREA CHAINING
*                                                          *
         LR    2,1            SAVE THE ADDRESS OF THE CPPL.
         USING CPPL,2         ADDRESSABILITY FOR THE CPPL.
         L     3,CPPLUPT      PLACE THE ADDRESS OF THE UPT
*                             INTO A REGISTER.
         L     4,CPPLECT      PLACE THE ADDRESS OF THE ECT
*                             INTO A REGISTER
*                                                          *
*  ISSUE THE EXECUTE FORM OF THE PUTLINE MACRO INSTRUCTION.
*  USE IT TO WRITE A SINGLE LINE OF DATA TO THE TERMINAL.
*  INCIDENTALLY, USE IT TO BUILD THE IOPL.
         PUTLINE    PARM=PUTBLOCK,UPT=(3),ECT=(4),
               ECB=ECBADS,OUTPUT=(TEXTADS,TERM,SINGLE,DATA),
               MF=(E,IOPLADS)
*  THIS EXECUTE FORM OF THE PUTLINE MACRO INSTRUCTION DOES
*  NOT SPECIFY THE TERMPUT OPERANDS; IT WILL USE THE DEFAULT
*  VALUES.
*                                                          *
*        PROCESSING                                        *
*                                                          *
*        STORAGE DECLARATIONS
*                                                          *
ECBADS   DS    F'Ø'           SPACE FOR THE EVENT CONTROL
*                             BLOCK
PUTBLOK  PUTLINE    MF=L      LIST FORM OF THE PUTLINE MACRO
*                             INSTRUCTION. THIS EXPANDS INTO
*                             A PUTLINE PARAMETER BLOCK.
TEXTADS  DC    H'2Ø'          LENGTH OF THE OUTPUT LINE.
         DC    H'Ø'           RESERVED.
         DC    CL16'ƀSINGLELINE DATA'
IOPLADS  DC    4F'Ø'          SPACE FOR THE INPUT OUTPUT
*                             PARAMETER LIST.
         IKJCPPL              DSECT FOR THE CPPL
         END
```

Figure 56. Coding Example -- PUTLINE Single Line Data

Multiline Data: Multiline data is a chain of single lines. Each line of data is processed by the PUTLINE service routine exactly as if it were single line data. Each element of the chain however, begins a new line to the terminal. By specifying multiline data (MULTLIN) in the PUTLINE macro instruction, you can put out several, variable length, non-contiguous lines at the terminal with one execution of the macro instruction. PUTLINE accepts Multiline data in a format similar to that of single line data except that each line is prefaced with a fullword forward chain pointer. Figure 57 shows the format of PUTLINE multiline data.

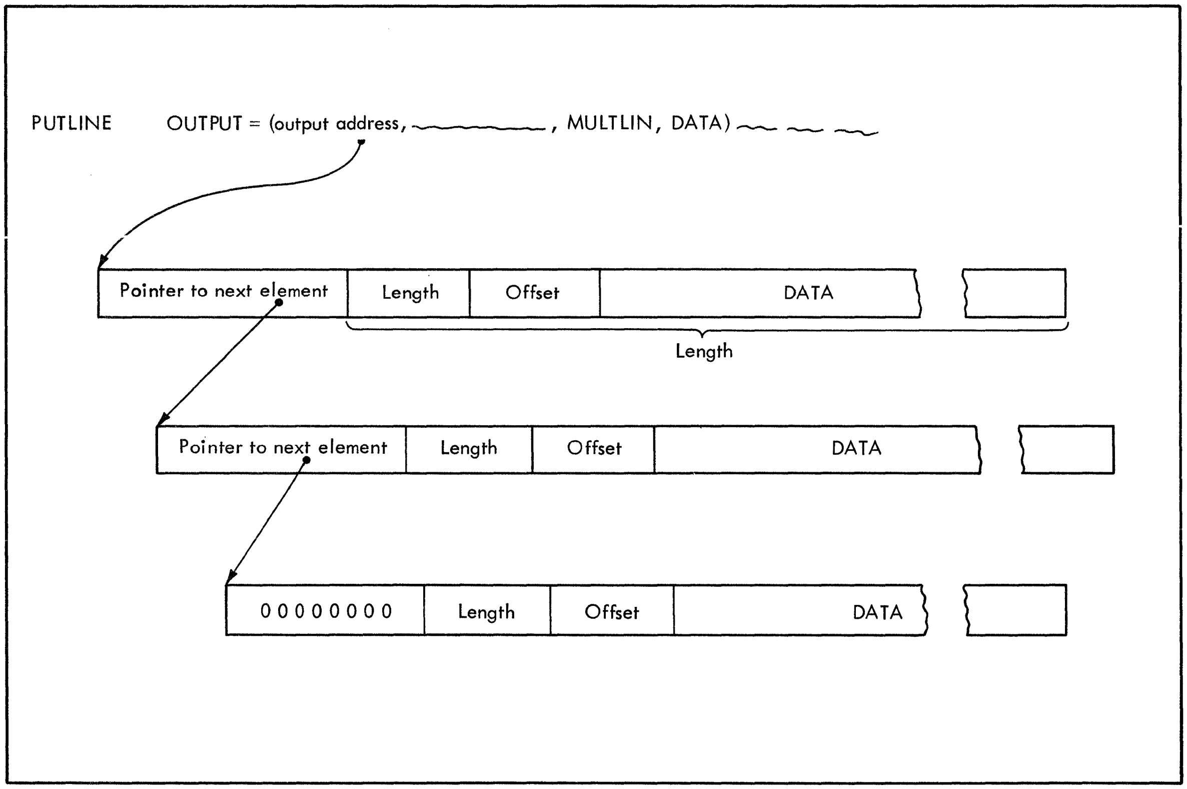
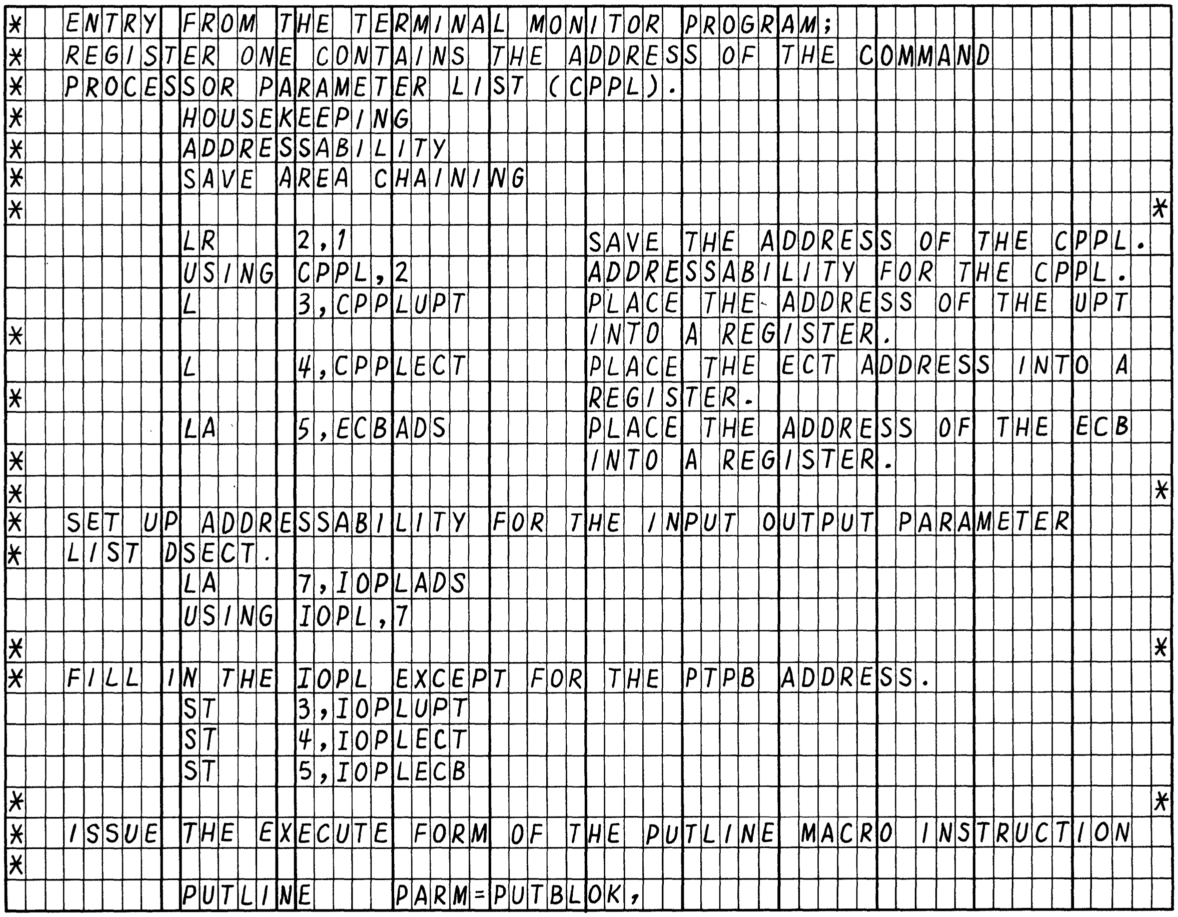


Figure 57. PUTLINE Multi-Line Data Format

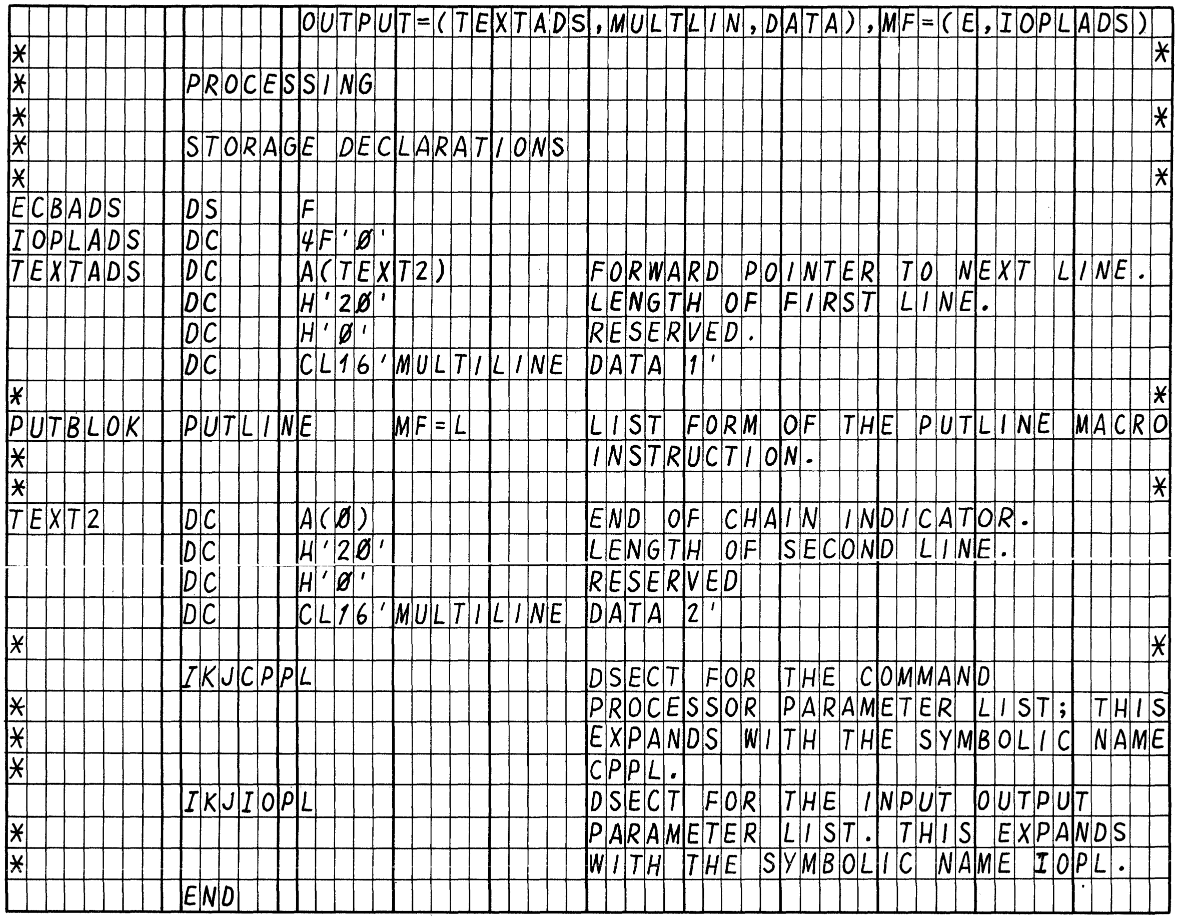
Each of the forward chain pointers points to the next data line to be written to the terminal. The forward chain pointer in the last data line contains zeros. In the case of multiline data, you pass the address of the output line to the PUTLINE service routine by coding the beginning address of the first forward chain pointer as the OUTPUT operand address in either the list or the execute form of the macro instruction. When the macro instruction expands, it will place this multi-line data address into the second word of the PUTLINE Parameter Block.

Figure 58 is an example of the code required to write multiple lines of data to the terminal using the PUTLINE macro instruction. Note that the programmer has built his own IOPL rather than build it with the execute form of the PUTLINE macro instruction. Note also the use of the IKJIOPL and IKJCPPL DESECTS. These provide an easy method of accessing the fields within the IOPL and the CPPL, and they protect your code from changes made to the control blocks.



```
*  ENTRY FROM THE TERMINAL MONITOR PROGRAM;
*  REGISTER ONE CONTAINS THE ADDRESS OF THE COMMAND
*  PROCESSOR PARAMETER LIST (CPPL).
*        HOUSEKEEPING
*        ADDRESSABILITY
*        SAVE AREA CHAINING
*                                                          *
         LR    2,1            SAVE THE ADDRESS OF THE CPPL.
         USING CPPL,2         ADDRESSABILITY FOR THE CPPL.
         L     3,CPPLUPT      PLACE THE ADDRESS OF THE UPT
*                             INTO A REGISTER.
         L     4,CPPLECT      PLACE THE ECT ADDRESS INTO A
*                             REGISTER.
         LA    5,ECBADS       PLACE THE ADDRESS OF THE ECB
*                             INTO A REGISTER.
*                                                          *
*  SET UP ADDRESSABILITY FOR THE INPUT OUTPUT PARAMETER
*  LIST DSECT.
         LA    7,IOPLADS
         USING IOPL,7
*                                                          *
*  FILL IN THE IOPL EXCEPT FOR THE PTPB ADDRESS.
         ST    3,IOPLUPT
         ST    4,IOPLECT
         ST    5,IOPLECB
*                                                          *
*  ISSUE THE EXECUTE FORM OF THE PUTLINE MACRO INSTRUCTION
*
         PUTLINE    PARM=PUTBLOK,
```

Figure 58. Coding Example -- PUTLINE Multi-Line Data (Part 1 of 2)



```
            OUTPUT=(TEXTADS,MULTLIN,DATA),MF=(E,IOPLADS)
*                                                        *
*       PROCESSING
*                                                        *
*       STORAGE DECLARATIONS
*                                                        *
ECBADS  DS  F
IOPLADS DC  4F'Ø'
TEXTADS DC  A(TEXT2)        FORWARD POINTER TO NEXT LINE.
        DC  H'2Ø'           LENGTH OF FIRST LINE.
        DC  H'Ø'            RESERVED.
        DC  CL16'MULTILINE  DATA 1'
*                                                        *
PUTBLOK PUTLINE   MF=L      LIST FORM OF THE PUTLINE MACRO
*                           INSTRUCTION.
*                                                        *
TEXT2   DC  A(Ø)            END OF CHAIN INDICATOR.
        DC  H'2Ø'           LENGTH OF SECOND LINE.
        DC  H'Ø'            RESERVED
        DC  CL16'MULTILINE  DATA 2'
*                                                        *
        IKJCPPL             DSECT FOR THE COMMAND
*                           PROCESSOR PARAMETER LIST; THIS
*                           EXPANDS WITH THE SYMBOLIC NAME
*                           CPPL.
        IKJIOPL             DSECT FOR THE INPUT OUTPUT
*                           PARAMETER LIST. THIS EXPANDS
*                           WITH THE SYMBOLIC NAME IOPL.
        END
```

Figure 58. Coding Example -- PUTLINE Multi-Line Data (Part 2 of 2)

MESSAGE LINES: If you code INFOR in the PUTLINE macro instruction, the PUTLINE service routine writes the information you supply as an informational message and provides additional functions not applicable to data lines. An informational message is a line of output from the program in control to the user at the terminal. It is used solely to pass output to the terminal; no input from the terminal is required after an informational message.

There are two types of informational messages processed by the PUTLINE service routine:

- Single Level Messages (SINGLE)
- Multi-Level Message (MULTLVL)

Single Level Messages: A single level message is composed of one or more message segments to be formatted and written to the terminal with one execution of the PUTLINE macro instruction.

MultiLevel Messages: Multilevel messages are composed of one or more message segments to be formatted and written to the terminal, and one or more message segments to be formatted and placed on an internal chain in shared subpool 78. This internal chain can either be put out to the terminal or purged by a second execution of the PUTLINE macro instruction.

Passing the Message Lines to PUTLINE: You must build each of the message segments to be processed by the PUTLINE Service Routine as if it were a line of single line data. The segment must be preceded by a four-byte header field -- the first two bytes containing the length of the segment, including the header, and the second two bytes containing zeros or an offset value if you use the text insertion facility. See "Using the PUTLINE Text Insertion Function" for a discussion of offset values. This message line format is required whether the message is a single level message or a multi-level message.

Because of the additional operations performed on message lines however, you must provide the PUTLINE service routine with a description of the line or lines that are to be processed. This is done with an Output Line Descriptor (OLD).

There are two types of Output Line Descriptors, depending on whether the messages are single level or multilevel.

The OLD required for a single level message is a variable length control block which begins with a full word value representing the number of segments in the message, followed by full word pointers to each of the segments.

The format of the OLD for multilevel messages varies from that required for single level messages in only one respect. You must preface the OLD with a full word forward chain pointer. This chain pointer points to another output line descriptor or contains zero to indicate that it is the last OLD on the chain. Figure 59 shows the format of the Output Line Descriptor.

Number of Bytes	Field Name	Contents or Meaning
4	none	The address of the next OLD, or zero if this is the last one on the chain. This field is present only if the message pointed to is a multi-level message.
4	none	The number of message segments pointed to by this OLD.
4	none	The address of the first message segment.
4	none	The address of the next message segment.
4	none	The address of the nth message segment.

Figure 59. The Output Line Descriptor

You must build the Output Line Descriptor and pass its address to the PUTLINE Service Routine as the OUTPUT operand address in either the list or the execute form of the macro instruction. When the macro instruction expands, it places the address of the Output Line Descriptor into the second word of the PUTLINE Parameter Block.

Figure 60 shows the two control block structures possible when processing a message line with PUTLINE. Note that the second word of the PUTLINE Parameter Block points to an Output Line Descriptor rather than directly to the message line itself.

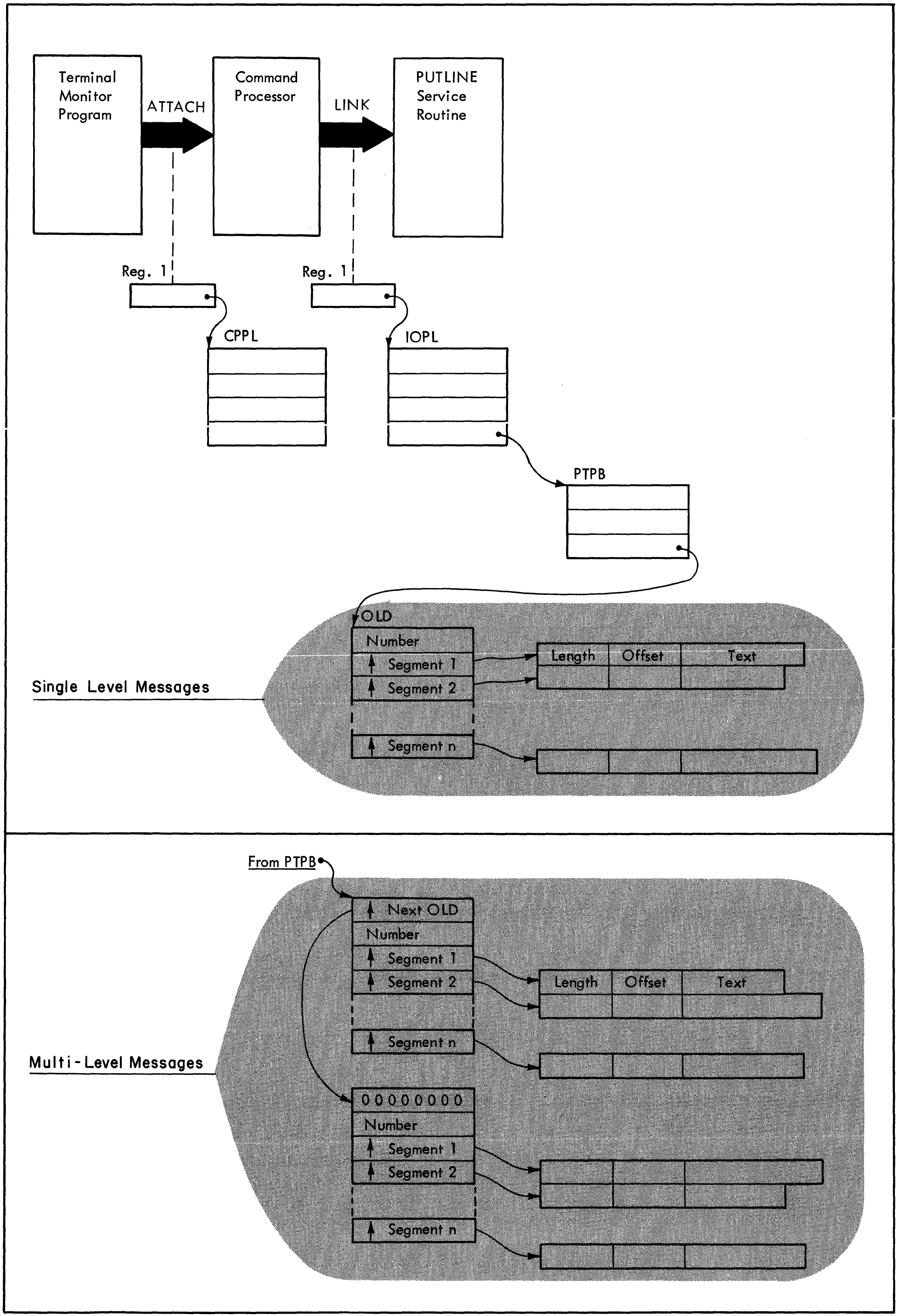


Figure 60. Control Block Structures for PUTLINE Messages

PUTLINE Message Line Processing:

In addition to writing a message out to the terminal, the PUTLINE service routine provides the following additional functions for message line (INFOR) processing:

- Message Identification Stripping
- Text Insertion
- Formatting Only
- Second level Informational Chaining

Figure 61 shows the distribution of these PUTLINE Service Routine functions over the two output message types.

FUNCTIONS	MESSAGE TYPES	
	Single Level	Multi-Level
Message I.D Stripping	X	X
Text Insertion	X	X
Formatting Only	X	
Second Level Informational Chaining		X

Figure 61. PUTLINE Functions and Message Types

STRIPPING MESSAGE IDENTIFIERS: The user at the terminal indicates whether or not he wants message identifiers displayed at the terminal. He does this with the PROFILE command. (See the publications Command Language Reference and Terminal User's Guide.) If the terminal user has indicated that he does not want message identifiers displayed, the PUTLINE service routine strips them off the message before writing the message to the terminal.

A message identifier must be a variable length character string, containing no leading or embedded blanks, must not exceed a maximum length of 255 characters, and must be terminated by a blank.

Messages without message identifiers must begin with a blank. A message beginning with a blank is handled by the PUTLINE service routine as a message that does not require message identifier stripping, regardless of what the user at the terminal has requested. If you do not provide a message identifier, and do not begin your message with a blank, the beginning of your message up to the first blank, will be stripped off by the PUTLINE service routine if message identifier stripping is requested from the terminal.

The following examples show the effects of the PUTLINE message identifier stripping function.

If you provide message identifiers on your messages, and the terminal user does not request message I.D. stripping, your message will appear at the terminal exactly as it appears here:

```
MESSAGE0010 THIS IS A MESSAGE.
```

If the user at the terminal requests message I.D. stripping, the message will appear as:

```
THIS IS A MESSAGE.
```

If you do not want to use message identifiers on your output messages, begin your message with a blank:

```
 THIS IS A MESSAGE.
```

A message beginning with a blank is unaffected by a terminal user's request for message I.D. stripping, and will appear as you wrote it, minus the blank; that is:

```
THIS IS A MESSAGE.
```

USING THE PUTLINE TEXT INSERTION FUNCTION: The text insertion function of the PUTLINE service routine allows you to build or modify messages at the time you put them out to the terminal. With text insertion you can respond to different output message requirements with one basic message (the primary segment). You can insert text into this primary segment or add text to it, and thereby build an output message to meet the current processing situation.

To use text insertion you pass your messages to the PUTLINE service routine as a variable number of text segments -- from 1 to 255 segments are permissible. Each segment may contain from 0 to 255 characters, however, the total number of characters in all the segments must not exceed 256. You must precede each of these text segments with a four byte header -- the first two bytes containing the length of the message, including the header, and the second two bytes containing an offset value. The offset value in the primary segment must be zero. The offset in any secondary segments may be from zero to the length of the primary segment's text field. An offset of zero in a secondary segment implies that the segment is to be placed before the primary segment. An offset that is equal to the length of the primary segment's text field implies that the secondary segment is to be placed after the primary segment. An offset of n, where n represents a value greater than zero but less than the total length of the primary segment, implies that the segment is to be inserted after the nth byte of the primary segment. PUTLINE places the secondary segment after that character, completes the message, and puts it out to the terminal.

If you specify an offset in a secondary segment, greater than the length of the primary segment, PUTLINE cannot handle the request and returns a code of 12 (invalid parameters) in register 15.

If you provide more than one secondary segment to be inserted into a primary segment, the offset fields in each of the secondary segments must indicate the position within the original primary segment at which you want them to appear. PUTLINE determines the points of insertion by counting the characters of the original primary segment only. As an example, if you provided one primary and two secondary segments as shown:

2 bytes	2 bytes	28 bytes
32	0	PLEASE ENTER TO PROCESSING

9	13	TEXT

13	17	CONTINUE

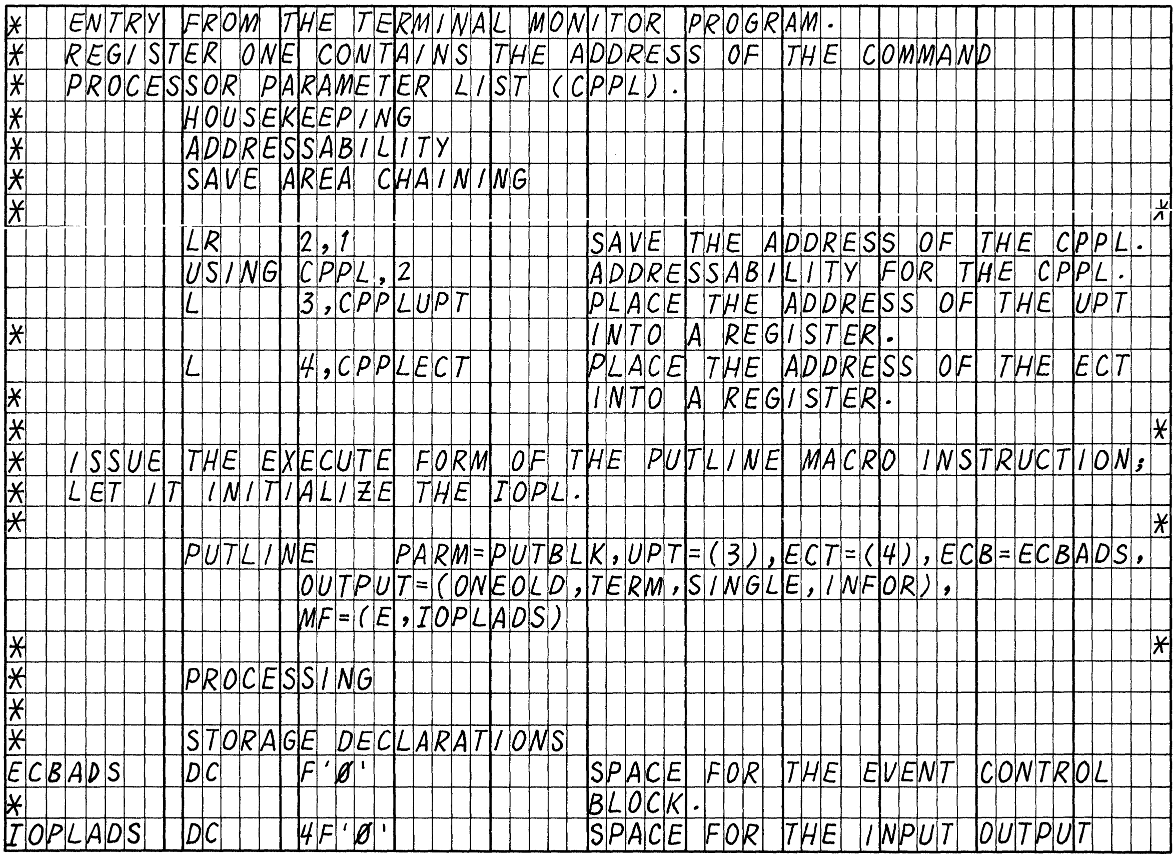
PUTLINE would place the first insert, TEXT, at the 14th character, and the second insert, CONTINUE, after the 17th character of the text field of the Primary segment. After PUTLINE inserts the two text segments, the message would read:

PLEASE ENTER TEXT TO CONTINUE PROCESSING

The leading and trailing blanks are automatically stripped off before the message is written to the terminal.

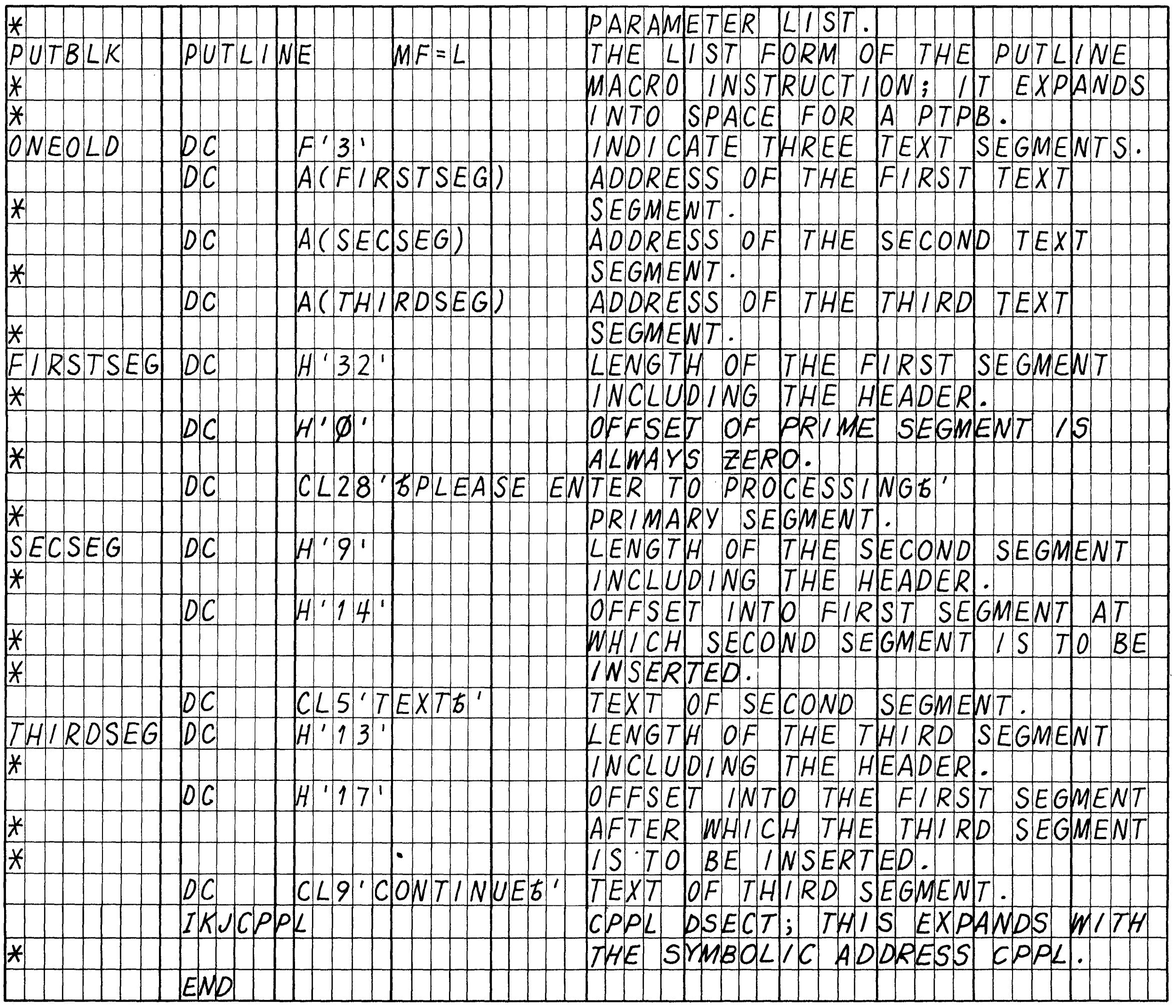
Figure 62 is an example of the code required to make use of the text insertion feature of the PUTLINE Service Routine; it uses the text segments shown above.

Note that the operand INFOR, which indicates to the PUTLINE service routine that the text segments are to be processed as informational messages, requires an output line descriptor to point to the message segments. Only one Output Line Descriptor (ONEOLD) is required to point to the 3 messages segments because the 3 segments are to be combined into one single level message.



```
*  ENTRY FROM THE TERMINAL MONITOR PROGRAM.
*  REGISTER ONE CONTAINS THE ADDRESS OF THE COMMAND
*  PROCESSOR PARAMETER LIST (CPPL).
*        HOUSEKEEPING
*        ADDRESSABILITY
*        SAVE AREA CHAINING
*                                                          *
         LR    2,1            SAVE THE ADDRESS OF THE CPPL.
         USING CPPL,2         ADDRESSABILITY FOR THE CPPL.
         L     3,CPPLUPT      PLACE THE ADDRESS OF THE UPT
*                             INTO A REGISTER.
         L     4,CPPLECT      PLACE THE ADDRESS OF THE ECT
*                             INTO A REGISTER.
*                                                          *
*  ISSUE THE EXECUTE FORM OF THE PUTLINE MACRO INSTRUCTION;
*  LET IT INITIALIZE THE IOPL.
*                                                          *
         PUTLINE    PARM=PUTBLK,UPT=(3),ECT=(4),ECB=ECBADS,
               OUTPUT=(ONEOLD,TERM,SINGLE,INFOR),
               MF=(E,IOPLADS)
*                                                          *
*        PROCESSING
*
*        STORAGE DECLARATIONS
ECBADS   DC    F'0'           SPACE FOR THE EVENT CONTROL
*                             BLOCK.
IOPLADS  DC    4F'0'          SPACE FOR THE INPUT OUTPUT
```

Figure 62. Coding Example -- PUTLINE Text Insertion (Part 1 of 2)



```
*                             PARAMETER LIST.
PUTBLK   PUTLINE     MF=L     THE LIST FORM OF THE PUTLINE
*                             MACRO INSTRUCTION; IT EXPANDS
*                             INTO SPACE FOR A PTPB.
ONEOLD   DC    F'3'           INDICATE THREE TEXT SEGMENTS.
         DC    A(FIRSTSEG)    ADDRESS OF THE FIRST TEXT
*                             SEGMENT.
         DC    A(SECSEG)      ADDRESS OF THE SECOND TEXT
*                             SEGMENT.
         DC    A(THIRDSEG)    ADDRESS OF THE THIRD TEXT
*                             SEGMENT.
FIRSTSEG DC    H'32'          LENGTH OF THE FIRST SEGMENT
*                             INCLUDING THE HEADER.
         DC    H'Ø'           OFFSET OF PRIME SEGMENT IS
*                             ALWAYS ZERO.
         DC    CL28'ƀPLEASE ENTER TO PROCESSINGƀ'
*                             PRIMARY SEGMENT.
SECSEG   DC    H'9'           LENGTH OF THE SECOND SEGMENT
*                             INCLUDING THE HEADER.
         DC    H'14'          OFFSET INTO FIRST SEGMENT AT
*                             WHICH SECOND SEGMENT IS TO BE
*                             INSERTED.
         DC    CL5'TEXTƀ'     TEXT OF SECOND SEGMENT.
THIRDSEG DC    H'13'          LENGTH OF THE THIRD SEGMENT
*                             INCLUDING THE HEADER.
         DC    H'17'          OFFSET INTO THE FIRST SEGMENT
*                             AFTER WHICH THE THIRD SEGMENT
*                             IS TO BE INSERTED.
         DC    CL9'CONTINUEƀ' TEXT OF THIRD SEGMENT.
         IKJCPPL              CPPL DSECT; THIS EXPANDS WITH
*                             THE SYMBOLIC ADDRESS CPPL.
         END
```

Figure 62. Coding Example -- PUTLINE Text Insertion (Part 2 of 2)

USING THE FORMAT ONLY FUNCTION: You can also use the PUTLINE service routine to format a message but not write it at the terminal. To do this, code the FORMAT operand in the PUTLINE macro instruction and pass PUTLINE the same message segment structure required for the text insertion function. The PUTLINE service routine performs text insertion if requested and places the finished message in subpool 1, which is not shared. It then places the address of the formatted line into the third word of the PUTLINE Parameter Block. The storage occupied by the formatted message belongs to your program and, if space is a consideration, must be freed by it. The returned formatted line is in the variable length record format; that is, it is preceded by a four byte header. You can use the first two bytes of this header to determine the length of the returned message, and later, to free the storage occupied by the message with the R form of the FREEMAIN macro instruction.

One difference between format only processing and text insertion processing is that format only processing can be used only on single level messages. You cannot use the format only feature to format multilevel messages. You can however, use the second level informational chaining function of PUTLINE to format second level messages and place them on an internal chain.

BUILDING A SECOND LEVEL INFORMATIONAL CHAIN: PUTLINE can accept two levels of informational messages at each execution of the service routine. It formats the first level message and puts it out to the terminal. The second level message is formatted and a copy of it is placed on an internal chain in shared subpool 78. This internal chain, the second level informational chain, is maintained by the I/O Service routines for the duration of one command or subcommand processor. You can use the PUTLINE service routine to purge this chain or to put it out to the terminal in its entirety.

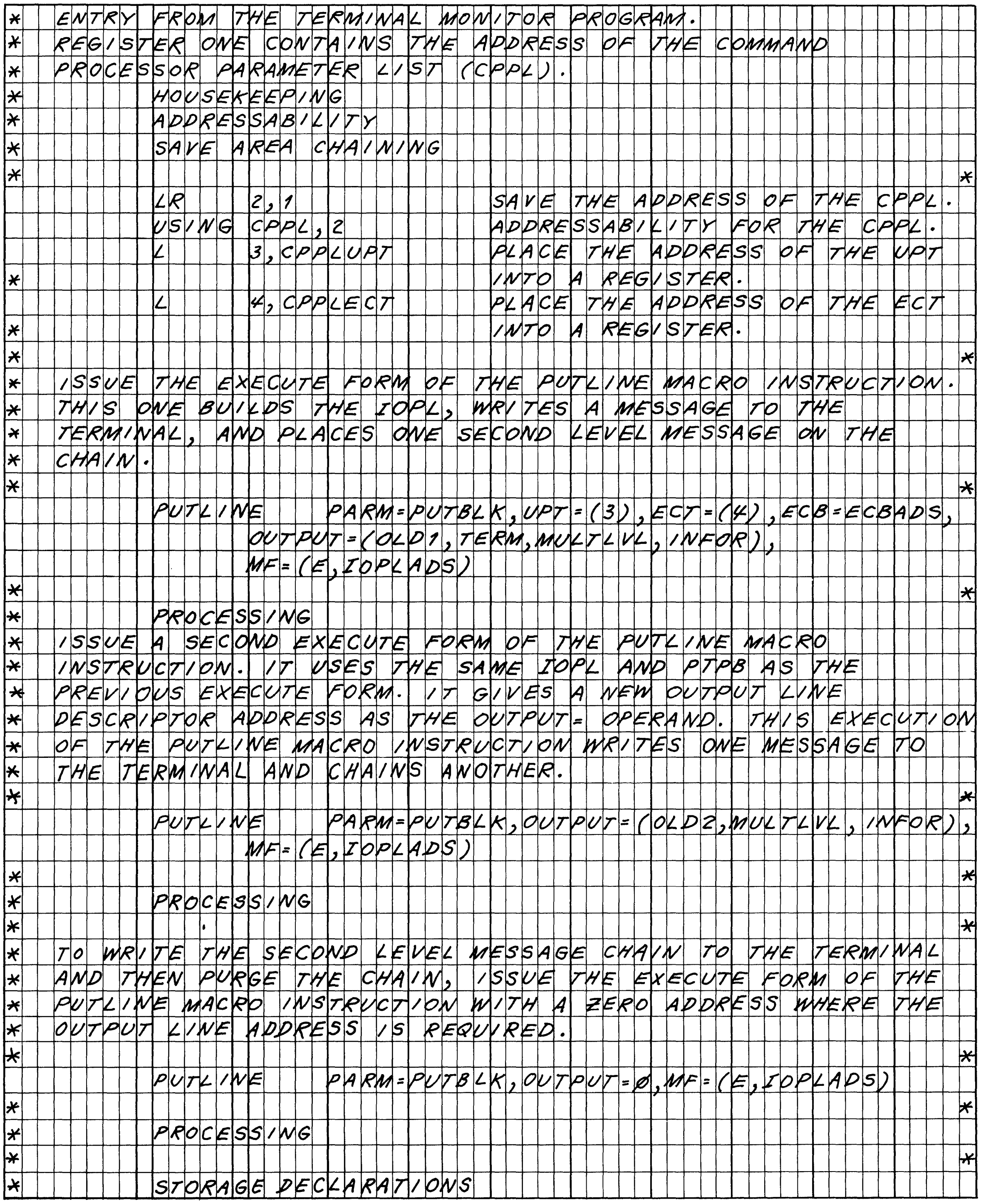
To purge the chain without putting it out to the terminal, you must turn on the high order bit in the first byte (ECTMSGF) of the third word of the Environment Control Table (ECT). The ECT is pointed to by the second word of the Input Output Parameter List, and may be mapped by the IKJECT DSECT. See Appendix A for a description of the ECT. The next time any chaining or unchaining is requested with PUTLINE or PUTGET, the second level informational chain will be eliminated.

To put the entire chain out to the terminal, use the PUTLINE macro instruction and place a zero address where the output line address is normally required. This will cause the PUTLINE service routine to write the chain to the terminal and eliminate the internal chain. You will normally use this procedure only if your attention exit routine is using the PUTLINE macro instruction to process a question mark entered from the terminal.

Figure 63 is an example of the code required to build a second level informational chain. It executes the PUTLINE service routine by using two different execute form macro instructions to modify the Putline Parameter Block built by the list form of the PUTLINE macro instruction.

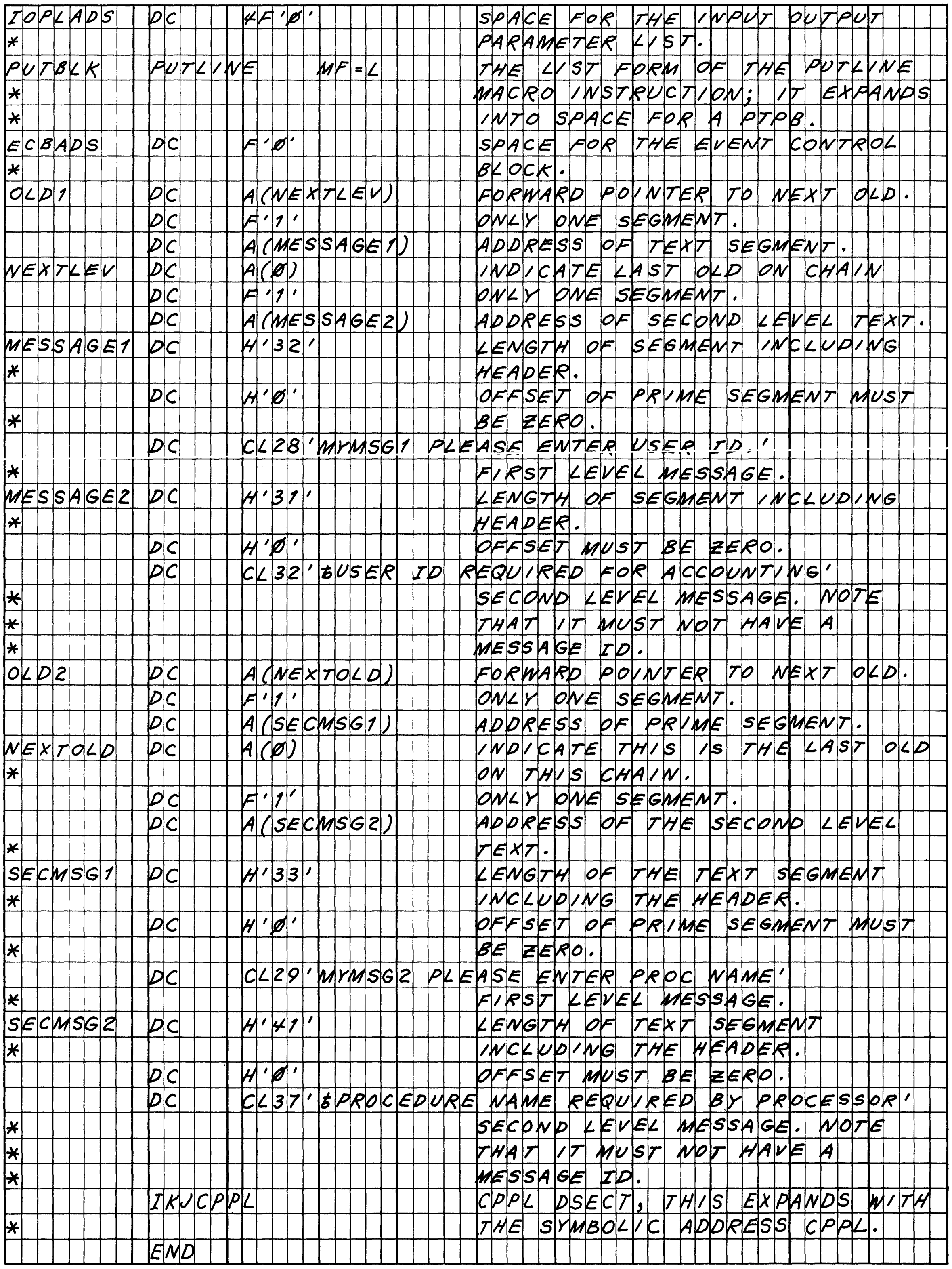
The code shown puts two messages out to the terminal and places two second level messages on an internal chain. It then executes a third execute form of the PUTLINE macro instruction with a zero OUTPUT= address to put the second level chain out to the terminal.

Note that the offset value for the primary message segment must always be zero, and when placing second level messages on an internal chain, the offset value for the second level message must also be zero. Note also that you do not place a message identifier on a second level message.



```
*  ENTRY FROM THE TERMINAL MONITOR PROGRAM.
*  REGISTER ONE CONTAINS THE ADDRESS OF THE COMMAND
*  PROCESSOR PARAMETER LIST (CPPL).
*        HOUSEKEEPING
*        ADDRESSABILITY
*        SAVE AREA CHAINING
*                                                         *
         LR    2,1           SAVE THE ADDRESS OF THE CPPL.
         USING CPPL,2        ADDRESSABILITY FOR THE CPPL.
         L     3,CPPLUPT     PLACE THE ADDRESS OF THE UPT
*                            INTO A REGISTER.
         L     4,CPPLECT     PLACE THE ADDRESS OF THE ECT
*                            INTO A REGISTER.
*                                                         *
*  ISSUE THE EXECUTE FORM OF THE PUTLINE MACRO INSTRUCTION.
*  THIS ONE BUILDS THE IOPL, WRITES A MESSAGE TO THE
*  TERMINAL, AND PLACES ONE SECOND LEVEL MESSAGE ON THE
*  CHAIN.
*                                                         *
         PUTLINE   PARM=PUTBLK,UPT=(3),ECT=(4),ECB=ECBADS,
               OUTPUT=(OLD1,TERM,MULTLVL,INFOR),
               MF=(E,IOPLADS)
*                                                         *
*        PROCESSING
*  ISSUE A SECOND EXECUTE FORM OF THE PUTLINE MACRO
*  INSTRUCTION. IT USES THE SAME IOPL AND PTPB AS THE
*  PREVIOUS EXECUTE FORM. IT GIVES A NEW OUTPUT LINE
*  DESCRIPTOR ADDRESS AS THE OUTPUT= OPERAND. THIS EXECUTION
*  OF THE PUTLINE MACRO INSTRUCTION WRITES ONE MESSAGE TO
*  THE TERMINAL AND CHAINS ANOTHER.
*                                                         *
         PUTLINE   PARM=PUTBLK,OUTPUT=(OLD2,MULTLVL,INFOR),
               MF=(E,IOPLADS)
*                                                         *
*        PROCESSING
*                                                         *
*  TO WRITE THE SECOND LEVEL MESSAGE CHAIN TO THE TERMINAL
*  AND THEN PURGE THE CHAIN, ISSUE THE EXECUTE FORM OF THE
*  PUTLINE MACRO INSTRUCTION WITH A ZERO ADDRESS WHERE THE
*  OUTPUT LINE ADDRESS IS REQUIRED.
*                                                         *
         PUTLINE   PARM=PUTBLK,OUTPUT=0,MF=(E,IOPLADS)
*                                                         *
*        PROCESSING
*                                                         *
*        STORAGE DECLARATIONS
```

Figure 63. Coding Example -- PUTLINE Second Level Informational Chaining (Part 1 of 2)



```
IOPLADS  DC    4F'Ø'                SPACE FOR THE INPUT OUTPUT
*                                   PARAMETER LIST.
PUTBLK   PUTLINE    MF=L            THE LIST FORM OF THE PUTLINE
*                                   MACRO INSTRUCTION; IT EXPANDS
*                                   INTO SPACE FOR A PTPB.
ECBADS   DC    F'Ø'                 SPACE FOR THE EVENT CONTROL
*                                   BLOCK.
OLD1     DC    A(NEXTLEV)           FORWARD POINTER TO NEXT OLD.
         DC    F'1'                 ONLY ONE SEGMENT.
         DC    A(MESSAGE1)          ADDRESS OF TEXT SEGMENT.
NEXTLEV  DC    A(Ø)                 INDICATE LAST OLD ON CHAIN
         DC    F'1'                 ONLY ONE SEGMENT.
         DC    A(MESSAGE2)          ADDRESS OF SECOND LEVEL TEXT.
MESSAGE1 DC    H'32'                LENGTH OF SEGMENT INCLUDING
*                                   HEADER.
         DC    H'Ø'                 OFFSET OF PRIME SEGMENT MUST
*                                   BE ZERO.
         DC    CL28'MYMSG1 PLEASE ENTER USER ID.'
*                                   FIRST LEVEL MESSAGE.
MESSAGE2 DC    H'31'                LENGTH OF SEGMENT INCLUDING
*                                   HEADER.
         DC    H'Ø'                 OFFSET MUST BE ZERO.
         DC    CL32'ƀUSER ID REQUIRED FOR ACCOUNTING'
*                                   SECOND LEVEL MESSAGE. NOTE
*                                   THAT IT MUST NOT HAVE A
*                                   MESSAGE ID.
OLD2     DC    A(NEXTOLD)           FORWARD POINTER TO NEXT OLD.
         DC    F'1'                 ONLY ONE SEGMENT.
         DC    A(SECMSG1)           ADDRESS OF PRIME SEGMENT.
NEXTOLD  DC    A(Ø)                 INDICATE THIS IS THE LAST OLD
*                                   ON THIS CHAIN.
         DC    F'1'                 ONLY ONE SEGMENT.
         DC    A(SECMSG2)           ADDRESS OF THE SECOND LEVEL
*                                   TEXT.
SECMSG1  DC    H'33'                LENGTH OF THE TEXT SEGMENT
*                                   INCLUDING THE HEADER.
         DC    H'Ø'                 OFFSET OF PRIME SEGMENT MUST
*                                   BE ZERO.
         DC    CL29'MYMSG2 PLEASE ENTER PROC NAME'
*                                   FIRST LEVEL MESSAGE.
SECMSG2  DC    H'41'                LENGTH OF TEXT SEGMENT
*                                   INCLUDING THE HEADER.
         DC    H'Ø'                 OFFSET MUST BE ZERO.
         DC    CL37'ƀPROCEDURE NAME REQUIRED BY PROCESSOR'
*                                   SECOND LEVEL MESSAGE. NOTE
*                                   THAT IT MUST NOT HAVE A
*                                   MESSAGE ID.
         IKJCPPL                    CPPL DSECT, THIS EXPANDS WITH
*                                   THE SYMBOLIC ADDRESS CPPL.
         END
```

Figure 63. Coding Example -- PUTLINE Second Level Informational Chaining (Part 2 of 2)

Return Codes From PUTLINE

When the PUTLINE service routine returns control to the program that invoked it, it provides one of the following return codes in general register fifteen:

CODE decimal	MEANING
0	PUTLINE completed normally.
4	The PUTLINE service routine did not complete. An attention interruption occurred during its execution, and the attention handler turned on the completion bit in the communications ECB.
8	The NOWAIT option was specified and the line was not written to the terminal.
12	Invalid parameters were supplied to the PUTLINE service routine.
16	A conditional GETMAIN was issued by PUTLINE for output buffers and there was not sufficient space to satisfy the request.

PUTGET - PUTTING A MESSAGE OUT TO THE TERMINAL AND OBTAINING A LINE OF INPUT IN RESPONSE

Use the PUTGET macro instruction to put messages out to the terminal and to obtain a response to those messages. A message to the user at the terminal which requires a response is called a conversational message. There are two types of conversational messages:

- Mode messages - Those which tell the user at the terminal which processing mode he is in so that he can enter a response applicable to that processing mode. Examples of mode messages are the READY sent to the terminal by the Terminal Monitor Program to indicate that it expects a Command to be entered, and the Command name (EDIT, TEST, etc.) sent by a Command Processor to indicate that it is ready to accept a subcommand name.

- Prompt messages - Those which prompt the user at the terminal to enter parameters required by the program in control, or to reenter those parameters which were previously entered incorrectly. Prompt information can only be obtained from the user at the terminal.

The input line returned by the PUTGET service routine can come from the terminal or from an in-storage list; PUTGET determines the source of input from the top element of the input stack unless you have specified the TERM or ATTN operands in the PUTGET macro instruction.

PUTGET, like PUTLINE and GETLINE has many parameters. The parameters are passed to the PUTGET service routine according to the operands you code in the List and the Execute forms of the PUTGET macro instruction.

This topic describes:

- The list and execute forms of the PUTGET macro instruction.

- Building the PUTGET Parameter Block.

- Types and formats of the output line.

- Passing the message lines to PUTGET.

- PUTGET processing.

- Input line format - the input buffer.

- An example of PUTGET.

- Return codes from PUTGET.

The PUTGET Macro Instruction - List Form

The list form of the PUTGET macro instruction builds and initializes a PUTGET Parameter Block (PGPB), according to the operands you specify in the PUTGET macro instruction. The PUTGET Parameter Block indicates to the PUTGET service routine which of the PUTGET functions you want performed. Figure 64 shows the list form of the PUTGET macro instruction; each of the operands is explained following the figure. Appendix B describes the notation used to define macro instructions.

```
┌──────────┬─────────┬──────────────────────────────────────────────────────┐
│          │         │┌                                     ┌,PROMPT ┐┐    │
│ [symbol] │ PUTGET  ││OUTPUT=(output address  ┌,SINGLE ┐ │,MODE   │)    │
│          │         ││                        └,MULTLVL┘ │,PTBYPS │     │
│          │         ││                                   │,TERM   │     │
│          │         │└                                   └,ATTN   ┘┘    │
│          │         │                                                    │
│          │         │┌            ┌EDIT   ┐                            ┐ │
│          │         ││,TERMPUT=(  │ASIS   │ ┌,WAIT  ┐┌,NOHOLD┐┌,NOBREAK┐)│ │
│          │         │└            └CONTROL┘ └,NOWAIT┘└,HOLD  ┘└,BREAKIN┘┘ │
│          │         │                                                    │
│          │         │┌,TERMGET=( ┌EDIT┐┌,WAIT  ┐)┐ ,MF=L                  │
│          │         │└           └ASIS┘└,NOWAIT┘ ┘                       │
└──────────┴─────────┴──────────────────────────────────────────────────────┘
```

Figure 64. The List Form of the PUTGET Macro Instruction

OUTPUT=output address

Specify the address of the Output Line Descriptor or a zero. The Output Line Descriptor (OLD) describes the message to be put out, and contains the address of the beginning (the one-word header) of the message or messages to be written to the terminal.

You have the option under MODE processing to provide or not provide an output message. If you do not provide an output line, code OUTPUT=0, and only the GET functions will take place. If you do provide an output message, the type of message and the processing to be performed by the PUTGET service routine are described by the OUTPUT sublist operands SINGLE, MULTLVL, PROMPT, MODE, PTBYPS, TERM, and ATTN. SINGLE and PROMPT are the default values .

SINGLE

The output message is a single level message.

MULTLVL

The output message consists of multiple levels. The first level message is written to the terminal, the secondary level messages are printed at the terminal, one at a time, in response to question marks entered from the terminal. Prompt must also be specified or defaulted to.

PROMPT

The output line is a prompt message.

MODE

The output line is a mode message.

PTBYPS

The output line is a prompt message and the terminal user's response will not print at those terminals that support the print inhibit feature. A terminal user can override bypass processing by hitting an attention followed by a carriage return before entering his input.

TERM

Specifies that the output line (a mode message) is to be written to the terminal, and a line is to be returned from the terminal, regardless of the top element of the input stack.

ATTN

Specifies that the output line (a mode message) is to be initially suppressed but an input line is to be returned from the terminal.

TERMPUT=

Specifies the TPUT options requested. Since PUTGET issues a TPUT SVC to write the message to the terminal, this operand is used to indicate which of the TPUT options you want to use. The TPUT options are EDIT, ASIS or CONTROL, WAIT, or NOWAIT, NOHOLD, or HOLD, and NOBREAK or BREAKIN. The default values are EDIT, WAIT, NOHOLD, and NOBREAK.

EDIT

Specifies that in addition to minimal editing (see ASIS), the following TPUT functions are requested:

a. Any trailing blanks are removed before the line is written to the terminal. If a blank line is sent, the terminal vertically spaces one line.

b. Control characters are added to the end of the output line to position the carrier to the beginning of the next line.

c. All terminal control characters (for example: bypass, restore, horizontal tab, new line) are replaced with a printable character. "Backspace" is an exception; see (d.) under ASIS.

ASIS

Specifies that minimal editing is to be performed by TPUT as follows:

a. The line of output is to be translated from EBCDIC to terminal code. Invalid characters will be converted to a printable character to prevent program caused I/O errors. This does not mean that all unprintable characters will be eliminated. "Restore", "upper case", "lower case", "bypass", and "bell ring", for example, might be valid but nonprinting characters at some terminals. (See CONTROL).

b. Transmission control characters will be added.

c. EBCDIC "NL", placed at the end of the message, indicates to the TPUT SVC that the carrier is to be returned at the end of the line. "NL" is replaced with whatever is necessary for that particular terminal type to cause the carrier to return. This "NL" processing occurs only if you specify ASIS, and the "NL" is the last character in your message.

If you specify EDIT, "NL" is handled as described in (c.) under EDIT.

If the "NL" is embedded in your message, it is sent to the terminal as a carriage return. No idle characters are added (see f. below). This may cause overprinting, particularly on terminals that require a line-feed character to position the carrier on a new line.

d. If you have used "backspace" in your output message, but the "backspace" character does not exist on the terminal type to which the message is being routed, TPUT attempts alternate methods to accomplish the backspace.

e. Control characters are added as needed to cause the message to occur on several lines if the output line is longer than the terminal line size.

f. Idle characters are sent at the end of each line to prevent typing as the carrier returns.

CONTROL

Specifies that the output line is composed of terminal control characters and will not print or move the carrier on the terminal. This option should be used for transmission of characters such as "bypass", "restore", or "bell ring".

WAIT

Specifies that control will not be returned to the program that issued the PUTGET until the output line has been placed into a terminal output buffer.

NOWAIT

Specifies that control should be returned to the program that issued the PUTGET macro instruction, whether or not a terminal output buffer is available. If no buffer is available a return code of 18 (decimal) is returned.

NOHOLD

Specifies that control is to be returned to the issuer of the PUTGET macro instruction, and that program can resume processing as soon as the output line has been placed on the output queue.

HOLD

Specifies that the program that issued the PUTGET macro instruction cannot continue its processing until this output line has been put out to the terminal or deleted.

NOBREAK

Specifies that if the terminal user has started to enter input, he is not to be interrupted. The output message is placed on the output queue to be printed after the terminal user has completed the line.

BREAKIN

Specifies that output has precedence over input. If the user at the terminal is transmitting, he is interrupted, and this output line is sent. Any data that was received before the interruption is kept and displayed at the terminal following this output line.

TERMGET=

Specifies the TGET options requested. Since PUTGET issues a TGET SVC to bring in a line of data, this operand is used to indicate to the TGET SVC which of the TGET options you wish to use. The TGET options are EDIT or ASIS, and WAIT or NOWAIT. The default values are EDIT and WAIT.

EDIT

Specifies that in addition to minimal editing (see ASIS), the buffer is to be filled out with trailing blanks.

ASIS

Specifies that minimal editing is to be done as follows:

a. Transmission control characters are removed.

b. The line of input is translated from terminal code to EBCDIC.

c. Line deletion and character deletion editing is performed.

d. Line feed and carriage return characters, if present, are removed.

WAIT

Specifies that control is to be returned to the program that issued the PUTGET macro instruction, only after an input message has been read.

NOWAIT

Specifies that control should be returned to the program that issued the PUTGET macro instruction whether or not a line of input is available. If a line of input is not available, a return code of 20 (decimal) is returned in register 15 to the command processor.

MF=L

Indicates that this is the list form of the macro instruction.

Note: In the list form of the PUTGET macro instruction, only

```
PUTGET  MF=L
```

is required.

The output line address is not specifically required in the list form of the PUTGET macro instruction, but must be coded in either the list or the execute form:

```
OUTPUT=(output address)
```

The other operands and their sublists, as shown below, are optional because you can supply them in the execute form of the macro instruction, or if you want the default values, they are supplied automatically by the expansion of the macro instruction.

```
          OUTPUT=(     {,SINGLE } {,PROMPT }
                       {,MULTLVL} {,MODE   })
                                  {,PTBYPS }
                                  {,TERM   }
                                  {,ATTN   }


                    {EDIT   }
          ,TERMPUT=({ASIS   } {,WAIT   } {,NOHOLD} {,NOBREAK})
                    {CONTROL} {,NOWAIT } {,HOLD  } {,BREAKIN}


          ,TERMGET=({EDIT} {,WAIT  })
                    {ASIS} {,NOWAIT}
```

The operands you specify in the list form of the PUTGET macro instruction set up control information used by the PUTGET service routine. This control information is passed to the PUTGET service routine in the PUTGET Parameter Block, a four word parameter block built and initialized by the list form of the PUTGET macro instruction.

The PUTGET Macro Instruction - Execute Form

Use the execute form of the PUTGET macro instruction to prepare a mode or a prompt message for output to the terminal, to determine whether or not that message should be sent to the terminal, and to return a line of input, from the source indicated by the top element of the input stack to the program that issued the PUTGET macro instruction.

You can use the execute form of the PUTGET macro instruction to build and initiate the Input Output Parameter List required by the PUTGET service routine, and to request PUTGET functions not already requested by the list form of the macro instruction, or to change those functions previously requested in either a list form or a previous execute form of the PUTGET macro instruction.

Figure 65 shows the execute form of the PUTGET macro instruction; each of the operands is explained following the figure. Appendix B describes the notation used to define macro instructions.

```
[symbol]   PUTGET   [PARM=parameter address][,UPT=upt address]

                    [,ECT=ect address][,ECB=ecb address]

                    ┌                                         ┌,PROMPT ┐ ┐
                    │,OUTPUT=(output address ┌,SINGLE ┐ │,MODE   │)│
                    │                        └,MULTLVL┘ │,PTBYPS │ │
                    │                                   │,TERM   │ │
                    └                                   └,ATTN   ┘ ┘

                    ┌         ┌EDIT   ┐                                  ┐
                    │,TERMPUT=(│ASIS   │ ┌,WAIT  ┐┌,NOHOLD┐┌,NOBREAK┐)│
                    └         └CONTROL┘ └,NOWAIT┘└,HOLD  ┘└,BREAKIN┘ ┘

                    ┌,TERMGET=(┌EDIT┐┌,WAIT  ┐)┐
                    └          └ASIS┘└,NOWAIT┘ ┘

                    ┌,ENTRY=┌entry address┐┐ ,MF=(E,┌list address┐)
                    └       └    (15)     ┘┘       └    (1)     ┘
```

Figure 65. The Execute Form of the PUTGET Macro Instruction

PARM=parameter address

Specifies the address of the 4-word PUTGET Parameter Block (PGPB). This address is placed into the Input Output Parameter List (IOPL). It may be the address of a list form PUTGET macro instruction. The address is any address valid in an RX instruction, or you can put it in one of the general registers 2-12, and use that register number, enclosed in parentheses, as the PARM= address.

UPT=upt address

Specifies the address of the User Profile Table (UPT). This address is placed into the IOPL when the execute form of the PUTGET macro instruction expands. You can obtain this address from the Command Processor Parameter List (CPPL) pointed to by register one when the Command Processor is attached by the Terminal Monitor Program. The address can be used as received in the CPPL or you can put it in one of the general registers 2-12, and use that register number, enclosed in parentheses , as the UPT address.

ECT=ect address

Specifies the address of the Environment Control Table (ECT). This address is placed into the IOPL when the Execute form of the PUTGET macro instruction expands. You can obtain this address from the Command Processor Parameter List (CPPL) pointed to by register one when the Command Processor is attached by the Terminal Monitor Program. The address can be used as received in the CPPL or you can put it in one of the general registers 2-12, and use that register number, enclosed in parentheses, as the ECT address.

ECB=ecb address

Specifies the address of the Command Processor Event Control Block (ECB). This address is placed into the IOPL by the execute form of the Putget macro instruction when it expands.

You must provide a one-word Event Control Block and pass its address to the PUTGET service routine by placing the address into the IOPL. If you code the address of the ECB in the execute form of the PUTGET macro instruction, the macro instruction places the address into the IOPL for you. The address can be any address valid in an RX instruction, or you can put in one of the general registers 2-12, and use that register number, enclosed in parentheses, as the ECB address.

OUTPUT=output address

Is the address of the Output Line Descriptor or a zero. The Output Line Descriptor (OLD) describes the message to be put out, and contains the address of the beginning (the one-word header) of the message or messages to be written to the terminal.

You have the option under MODE processing to provide or not provide an output message. If you do not provide an output line, code OUTPUT=0, and only the GET functions will take place. If you do provide an output message, the type of message and the processing to be performed by the PUTGET Service Routine are described by the OUTPUT sublist operands SINGLE, MULTLVL, PROMPT, MODE, PTBYPS, TERM, and ATTN. The default values are SINGLE and PROMPT.

SINGLE

The output message is a single level message.

MULTLVL

The output message consists of multiple levels. The first level message is written to the terminal, the secondary level messages are printed at the terminal, one at a time, in response to question marks entered from the terminal. PROMPT must also be specified or defaulted to.

PROMPT

The output line is a prompt message.

MODE

The output line is a mode message.

PTBYPS

The output line is a prompt message and the terminal user's response will not print at those terminals that support the print inhibit feature. A terminal user can override bypass processing by hitting an attention followed by a carriage return before entering his input.

TERM

Specifies that the output line (a mode message) is to be written to the terminal, and a line is to be returned from the terminal, regardless of the top element of the input stack.

ATTN

specifies that the output line (a mode message) is to be initially suppressed but an input line is to be returned from the terminal.

TERMPUT=

Specifies the TPUT options requested. PUTGET issues a TPUT SVC to write the message to the terminal, this operand is used to indicate which of the TPUT options you want to use. The TPUT options are EDIT, ASIS or CONTROL, WAIT or NOWAIT, NOHOLD or HOLD, and NOBREAK or BREAKIN. The default values are EDIT, WAIT, NOHOLD and NOBREAK.

EDIT

Specifies that in addition to minimal editing (see ASIS), the following TPUT functions are requested:

a. Any trailing blanks are removed before the line is written to the terminal. If a blank line is sent, the terminal vertically spaces one line.
b. Control characters are added to the end of the output line to position the carrier to the beginning of the next line.
c. All terminal control characters (for example: bypass, restore, horizontal tab, new line) are replaced with a printable character. "Backspace" is an exception; see (d.) under ASIS.

ASIS
Specifies that minimal editing is to be performed by TPUT as follows:

a. The line of output is translated from EBCDIC to terminal code. Invalid characters are converted to a printable character to prevent program caused I/O errors. This does not mean that all unprintable characters will be eliminated. "Restore", "upper case", "lower case", "bypass", and "bell ring", for example, might be valid but nonprinting characters at some terminals. (See CONTROL).

b. Transmission control characters are added.

c. EBCDIC "NL", placed at the end of the message, indicates to the TPUT SVC that the carrier is to be returned at the end of the line. "NL" is replaced with whatever is necessary for that particular terminal type to cause the carrier to return. This "NL" processing occurs only if you specify ASIS, and the "NL" is the last character in your message.

   If you specify EDIT, "NL" is handled as described in (c.) under EDIT.

   If the "NL" is embedded in your message, it is sent to the terminal as a carriage return. No idle characters are added (see f. below). This may cause overprinting, particularly on terminals that require a line-feed character to position the carrier on a new line.

d. If you have used "backspace" in your output message, but the "backspace" character does not exist on the terminal type to which the message is being routed, TPUT attempts alternate methods to accomplish the backspace.

e. Control characters are added as needed to cause the message to occur on several lines if the output line is longer than the terminal line size.

f. Idle characters are sent at the end of each line to prevent typing as the carrier returns.

CONTROL
Specifies that this line is composed of terminal control characters and will not print or move the carrier on the terminal. This option should be used for transmission of characters such as "bypass", "restore", or "bell ring".

WAIT
Specifies that control will not be returned to the program that issued the PUTGET until the output line has been placed into terminal output buffer.

NOWAIT
Specifies that control should be returned to the program that issued the PUTGET macro instruction, whether or not a terminal output buffer is available. If no buffer is available, a return code of 18 (decimal) is returned.

NOHOLD
Specifies that control is to be returned to the program that issued the PUTGET macro instruction, and it can continue processing as soon as the output line has been placed on the output queue.

HOLD

Specifies that the program that issued the PUTGET macro instruction cannot continue its processing until the output line has been put out to the terminal or deleted.

NOBREAK

Specifies that if the terminal user has started to enter input, he is not to be interrupted. The output message is placed on the output queue to be printed after the terminal user has completed the line.

BREAKIN

Specifies that output has precedence over input. If the user at the terminal is transmitting, he is interrupted, and this output line is sent. Any data that was received before the interruption is kept and displayed at the terminal following this output line.

TERMGET=

Specifies the TGET options requested. PUTGET issues a TGET SVC to bring in a line of data, this operand is used to indicate to the TGET SVC which of the TGET options you want to use. The TGET options are EDIT or ASIS, and WAIT or NOWAIT. The default values are EDIT and WAIT.

EDIT

Specifies that in addition to minimal editing (see ASIS), the buffer is filled out with trailing blanks.

ASIS

Specifies that minimal editing is done as follows:

a. Transmission control characters are removed.

b. The line of input is translated from terminal code to EBCDIC.

c. Line deletion and character deletion editing is performed.

d. Line feed and carriage return characters, if present, are removed.

WAIT

Specifies that control is to be returned to the program that issued the PUTGET macro instruction, only when an input message has been read.

NOWAIT

Specifies that control should be returned to the program that issued the PUTGET macro instruction whether or not a line of input is available. If a line of input is not available, a return code of 20 (decimal) is returned in register 15.

ENTRY= entry point address
(15)

Specifies the entry point of the PUTGET service routine. If ENTRY is omitted, the PUTGET macro expansion generates a LINK macro instruction to invoke the PUTGET service routine. The address may be any address valid in an RX instruction or (15) if you load the entry point address into general register 15.

MF=E

Indicates that this is the execute form of the PUTGET macro instruction.

listaddr
  (1)
    The address of the 4-word Input Output Parameter List (IOPL). This can be a completed IOPL that you have built, or it may be 4 words of declared storage that will be filled from the PARM, UPT, ECT, and ECB operands of this execute form of the PUTGET macro instruction. The address must be any address valid in an RX instruction or (1) if you have loaded the parameter list address into general register 1.

Note: In the execute form of the PUTGET macro instruction, only the following is required:

```
PUTGET   MF=(E,{list address})
               {     (1)     }
```

The PARM=, UPT=, ECT=, and ECB= operands are not required if you have built your IOPL in your own code.

The output line address is not specifically required in the execute form of the PUTGET macro instruction, but must be coded in either the list or the execute form:

```
OUTPUT=(output address)
```

The other operands and sublists are optional because you may have supplied them in the list form of the macro instruction or in a previous execute form; or because you may want to use the default values which are automatically supplied by the macro expansion itself. The other operands and sublists are:

```
                                  {,PROMPT }
            OUTPUT=(   {,SINGLE } {,MODE   })
                       {,MULTLVL} {,PTBYPS }
                                  {,TERM   }
                                  {,ATTN   }

                      {EDIT   }
            ,TERMPUT=({ASIS   }{,WAIT  }{,NOHOLD}{,NOBREAK})
                      {CONTROL}{,NOWAIT}{,HOLD  }{,BREAKIN}

            ,TERMGET=({EDIT}{,WAIT  })
                      {ASIS}{,NOWAIT}
```

The ENTRY= operand need not be coded in the macro instruction. If it is not, a LINK macro instruction is generated by the PUTGET macro expansion to invoke the PUTGET service routine.

The operands you specify in the execute form of the PUTGET macro instruction set up control information used by the PUTGET service routine. You can use the PARM=, UPT=, ECT=, and ECB= operands of the PUTGET macro instruction to build, complete, or modify an IOPL. The OUTPUT=, TERMPUT=, and TERMGET= operands and their sublist operands initiate the PUTGET Parameter Block. The PUTGET Parameter Block is referenced by the PUTGET service routine to determine which functions you want PUTGET to perform.

Building the PUTGET Parameter Block (PGPB)

When the list form of the PUTGET macro instruction expands, it builds a four-word PUTGET Parameter Block (PGPB). This PGPB combines the functions of the PUTLINE and the GETLINE parameter blocks and contains information used by the PUT and the GET functions of the PUTGET service routine. The list form of the PUTGET macro instruction initializes this PGPB according to the operands you have coded in the macro instruction. This initialized block, which you may later modify with the execute form of the PUTGET macro instruction, indicates to the PUTGET service routine the functions you want performed. It also contains a pointer to the Output Line Descriptor that describes the output message and it provides a field into which the PUTGET service routine places the address of the input line returned from the input source.

You must pass the address of the PGPB to the execute form of the PUTGET macro instruction. Since the list form of the macro instruction expands into a PGPB, all you need do is pass the address of the list form of the macro instruction to the execute form as the PARM value.

The PUTGET Parameter Block is defined by the IKJPGPB DSECT. Figure 66 describes the contents of the PUTGET parameter block.

Number of Bytes	Field	Contents or Meaning
2		PUT Control flags. These bits describe the output line to the PUTGET Service Routine.
	Byte 1	
	..0.	Always zero for PUTGET.
	...1	The output line is a single level message.
	.... 0...	Must be zero for PUTGET.
	.... .1..	The output line is a multilevel message.
	1	The output line is a PROMPT message.
	xx.. ..x.	Reserved.
	Byte 2	
	1...	The output line is a MODE message.
	...1	BYPASS processing is requested.
	.... 1...	ATTN processing is requested.
	.xx. .xxx	Reserved.
2		TPUT options field. These bits indicate to the TPUT SVC which of the TPUT options you want to use.
	Byte 1	
	0...	Always set to 0 for TPUT.
	...0	WAIT processing has been requested. Control will be returned to the issuer of TPUT only after the output line has been placed into a terminal output buffer.
	...1	NOWAIT processing has been requested. Control will be returned to the issuer of TPUT whether or not a terminal output buffer is available.
	.... 0...	NOHOLD processing has been requested. The issuer of the TPUT can resume processing as soon as the output line has been placed on the output queue.
	.... 1...	HOLD processing has been requested. The issuer of the TPUT is not to resume processing until the output line has been written to the terminal or deleted.

Figure 66. The PUTGET Parameter Block (Part 1 of 2)

Number of Bytes	Field	Contents or Meaning
	.... .0..	NOBREAK processing has been requested. The output line will be printed only when the terminal user is not entering a line.
	.... .1..	BREAKIN processing has been requested. The output line is to be sent to the terminal immediately. If the terminal user is entering a line, he is to be interrupted.
	00	EDIT processing has been requested.
	01	ASIS processing has been requested.
	10	CONTROL processing has been requested.
	.xx.	Reserved
	Byte 2	Reserved.
4		The address of the Output Line Descriptor.
2		GET control flags.
	Byte 1	
	.00.	Always zero for PUTGET.
	...1	TERM processing is requested.
	x... xxxx	Reserved bits.
	Byte 2	
	xxxx xxxx	Reserved.
2		TGET options field. These bits indicate to the TGET SVC which of the TGET options you wish to use.
	Byte 1	
	1...	Always set to 1 for TGET.
	...0	WAIT processing has been requested. Control will be returned to the issuer of the TGET SVC only after an input message has been read.
	...1	NOWAIT processing has been requested. Control will be returned to the issuer of the TGET SVC whether or not a line of input is available. If no line was available, PUTGET returns a code of 20 (decimal) in general register 15.
	00	EDIT processing has been requested. In addition to the editing provided by ASIS processing, the input buffer is to be filled out with trailing blanks to the next doubleword boundary.
	01	ASIS processing has been requested. (See the ASIS operand of the PUTGET macro instruction description).
	.xx. xx..	Reserved bits.
	Byte 2	
	xxxx xxxx	Reserved.
4	PGPBIBUF	The address of the input buffer. The PUTGET Service Routine fills this field with the address of the input buffer in which the input line has been placed.

Figure 66. The PUTGET Parameter Block (Part 2 of 2)

Types and Formats of the Output Line

The PUTGET Service Routine writes only conversational messages to the terminal; it does not handle data lines. For information on how to write a data line or a nonconversational message to the terminal, see the section on the PUTLINE macro instruction.

PUTGET accepts two output line formats depending upon whether the message you provide is a single level message or a multilevel message.

SINGLE LEVEL MESSAGES: A single level message is composed of one or more message segments to be formatted and written to the terminal with one execution of the PUTGET macro instruction.

MULTI-LEVEL MESSAGES: Multilevel messages are composed of one or more message segments to be formatted and written to the terminal, and one or more message segments to be formatted and printed to the terminal in response to question marks entered from the terminal. Note however, that if you specify MODE in the PUTGET macro instruction, you can process only single level messages. If you specify PROMPT in the PUTGET macro instruction, then these second level messages will be written to the terminal, one at a time, in response to successive question marks entered from the terminal. If these PROMPT messages are to be available to the user at the terminal however, the top element of the input stack must not specify a procedure element as the current source of input, and the terminal user must not have inhibited prompting. (See the PROFILE command in the TSO Command Language Reference.

Passing the Message Lines to PUTGET

You must build each of the message segments to be processed by the PUTGET service routine as if it were a line of single line data. The segment must be preceded by a four-byte header field -- the first two bytes containing the length of the segment including the header, and the second two bytes containing zeros or an offset value if you use the text insertion facility provided by PUTGET. This message line format is required whether the message is a single level message or a multilevel message.

Because of the additional functions performed on message lines -- message ID stripping, text insertion, and multi-level processing -- you must provide the PUTGET Service Routine with a description of the line or lines that are to be processed. This is done with an OUTPUT Line Descriptor (OLD).

There are two types of Output Line Descriptors depending on whether the messages are single level or multilevel.

The OLD required for a single level message is a variable length control block which begins with a fullword value representing the number of segments in the message, followed by fullword pointers to each of the segments.

The format of the OLD for multilevel messages varies from that required for single level messages in only one respect. You must preface the OLD with a fullword forward chain pointer. This chain pointer points to another Output Line descriptor or contains zero to indicate that it is the last OLD on the chain. Figure 67 shows the format of the Output Line Descriptor.

Number of Bytes	Field	Contents or Meaning
4		The address of the next OLD, or zero if this is the last one on the chain. This field is present only if the message pointed to is a multi-level message.
4		The number of message segments pointed to by this OLD.
4		The address of the first message segment.
4		The address of the next message segment.
4		The address of the nth message segment.

Figure 67. The Output Line Descriptor (OLD)

You must build the Output Line Descriptor and pass its address to the PUTLINE service routine as the OUTPUT operand address in either the list or the execute form of the macro instruction. When the macro instruction expands, it places this OLD address into the second word of the PUTLINE Parameter Block.

Figure 68 shows the two control block structures possible when passing an output message to the PUTGET service routine. Note that MODE, TERM, or ATTN may not be coded in the PUTGET macro instruction if you want to provide multilevel messages to the terminal. (Mode messages can have only one level.)

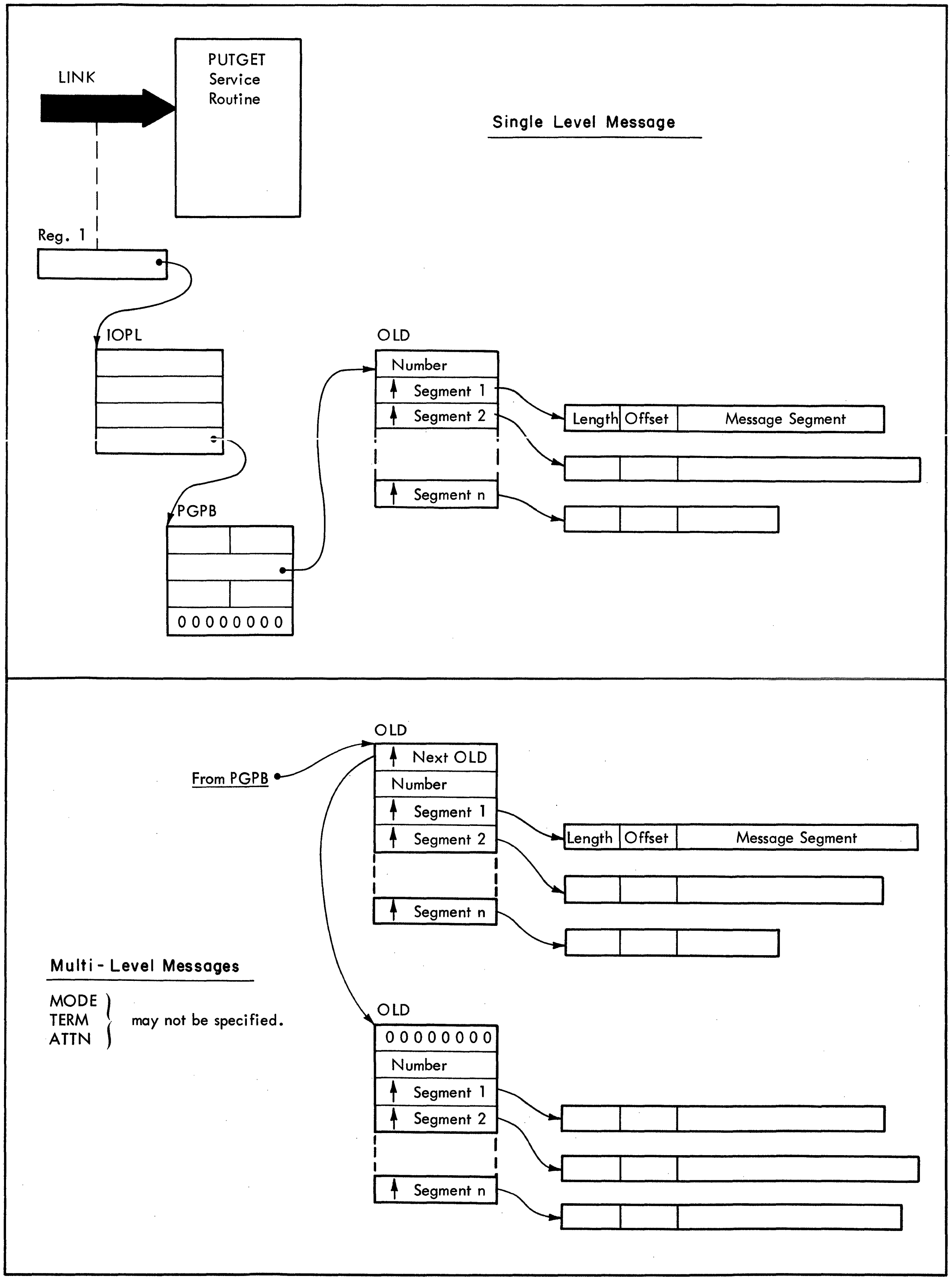


Figure 68. Control Block Structures for PUTGET Output Messages

PUTGET Processing

Text insertion and message identifier stripping are available to all output messages processed by the PUTGET service routine. For a detailed description of these functions see the section headed "PUTLINE Message Line Processing."

The PUTGET service routine provides other processing capabilities dependent upon whether the message is a MODE or a PROMPT message.

MODE MESSAGE PROCESSING: A MODE message is a message put out to the terminal when a command or a subcommand is anticipated. The processing of MODE messages by the PUTGET service routine is dependent upon the following two conditions:

1. Are you providing an output line?
2. From what source is the input line coming?

Is an Output Line Present: You need not provide an output line to the PUTGET service routine. If you do provide an output line address then PUT processing will take place. Whether your output line is written to the terminal is then dependent upon the input source indicated by the input stack. If you do not provide an output line (OUTPUT=0) then only the GET function of the PUTGET service routine takes place.

What is the Input Source: The source of the input line, as determined by the top element of the input stack, determines the type of processing performed by the PUTGET service routine. You may however override the input stack by coding the TERM or ATTN operands in the PUTGET macro instruction. The two sources of input supported are:

1. Terminal
2. In-storage

If the current source of input is the terminal, and you provide an output line, the PUTGET service routine writes the line to the terminal, returns a line from the terminal, and places the address of the returned line into the fourth word of the PUTGET Parameter Block. If the line returned from the terminal is a question mark however, the PUTGET service routine causes the secondary level informational message chain (if one exists) to be written to the terminal, again puts out the mode message, and then returns a line from the terminal. If the user at the terminal enters a question mark in response to a mode message, and no second level message chain exists, PUTGET puts out the message "IKJ66760I NO INFORMATION AVAILABLE", puts the mode message out again, and returns a line from the terminal.

Note that if the user enters a question mark from the terminal, the second level chain returned to the terminal is not related to the current mode message but to the Command Processor just terminated; mode messages can have only one level.

If the current source of input is an in-storage list, the output line (if you provide one) is ignored and the PUTGET Service Routine normally obtains an input line from the in-storage list and places a pointer to that line in the fourth word of the PGPB. If however, a second level information chain exists, PUTGET will only return a line if the user at the terminal has access to the information in the chain through the PAUSE mechanism. If the chain is not available to the user, no line is obtained by PUTGET, and it returns a code of 12 in register 15. You can test this return code, and if you want, recover from this error condition by turning on the high order bit of the ECTMSGF field of the Environment Control Table (see Appendix A) and reissuing the PUTGET. The second level information chain is then purged and a line is obtained from the in-storage list.

PAUSE PROCESSING: If the user at the terminal has requested the PAUSE option on the PROFILE command, the PUTGET Service Routine makes the chained second level informational messages available to him, even if the current input source is not the terminal.

PAUSE processing works as follows. If a second level informational chain does exist, PUTGET puts out the message 'IKJ56762A PAUSE' to the terminal informing the terminal user that PAUSE processing is in effect. At this point the terminal user can enter either a question mark to indicate that he wishes to have the chained second level messages put out to the terminal, or a carriage return to indicate that the information is not needed. If the user enters a carriage return, the second level informational message chain is eliminated. If he enters any response other than a question mark or a carriage return, PUTGET prompts him for a correct response.

PROMPT MESSAGE PROCESSING: A PROMPT message is a message put out to the terminal when the program in control requires input from the terminal user. PROMPT information must come from the terminal and can not be obtained from any other source of input. There are two cases when a request for PROMPT processing is denied by PUTGET:

1. When the current source of input, as determined by the top element of the input stack, is an in-storage procedure.

2. When the terminal user has requested via the PROFILE command that no prompting be done.

If PROMPT processing is allowed, the PUTGET service routine writes the first level message to the terminal and obtains an input line from the terminal. If the input line is a question mark, PUTGET either returns the next level message provided, or a message informing the user that no information is available. PUTGET continues to respond to question marks entered from the terminal by writing one more secondary level message to the terminal in response to each question mark entered until the chain is exhausted; at that point PUTGET issues a message informing the user at the terminal that no more information is available. The prompt message is not repeated and the task goes into an input wait until the terminal user enters a line. When a line is obtained from the terminal, PUTGET places the address of the line into the fourth word of the PGPB.

Input Line Format - the Input Buffer

The fourth word of the PUTGET Parameter Block contains zeros until the PUTGET service routine returns a line of input. The service routine places the requested input line into an input buffer beginning on a doubleword boundary located in subpool 1. It then places the address of this input buffer into the fourth word of the PGPB. The input buffer belongs to the program that issued the PUTGET macro instruction. The buffer or buffers returned by PUTGET are automatically freed when your code relinquishes control. If space is limited, you should free the input buffer with the FREEMAIN macro instruction after you have processed or copied the input line.

Regardless of the source of input, an in-storage list or the terminal, the input line returned by the PUTGET service routine is in a standard format. All input lines are in the variable length record format with a fullword header followed by the text returned by PUTGET. Figure 69 shows the format of the input buffer returned by the PUTGET Service Routine.

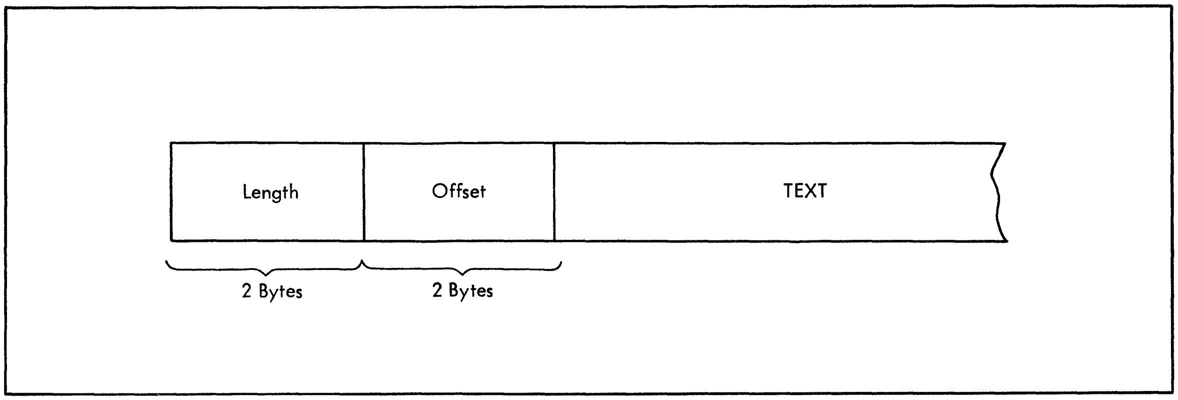


Figure 69. Format of the PUTGET Input Buffer

The two-byte length field contains the length of the returned input line including the header (4 bytes). You can use this length field to determine the length of the input line to be processed, and later, to free the input buffer with the R form of the FREEMAIN macro instruction. The two-byte offset field is always set to zero on return from the PUTGET Service Routine.

Figure 70 shows the PUTGET control block structure for a multilevel PROMPT message after the PUTGET service routine has returned an input line.

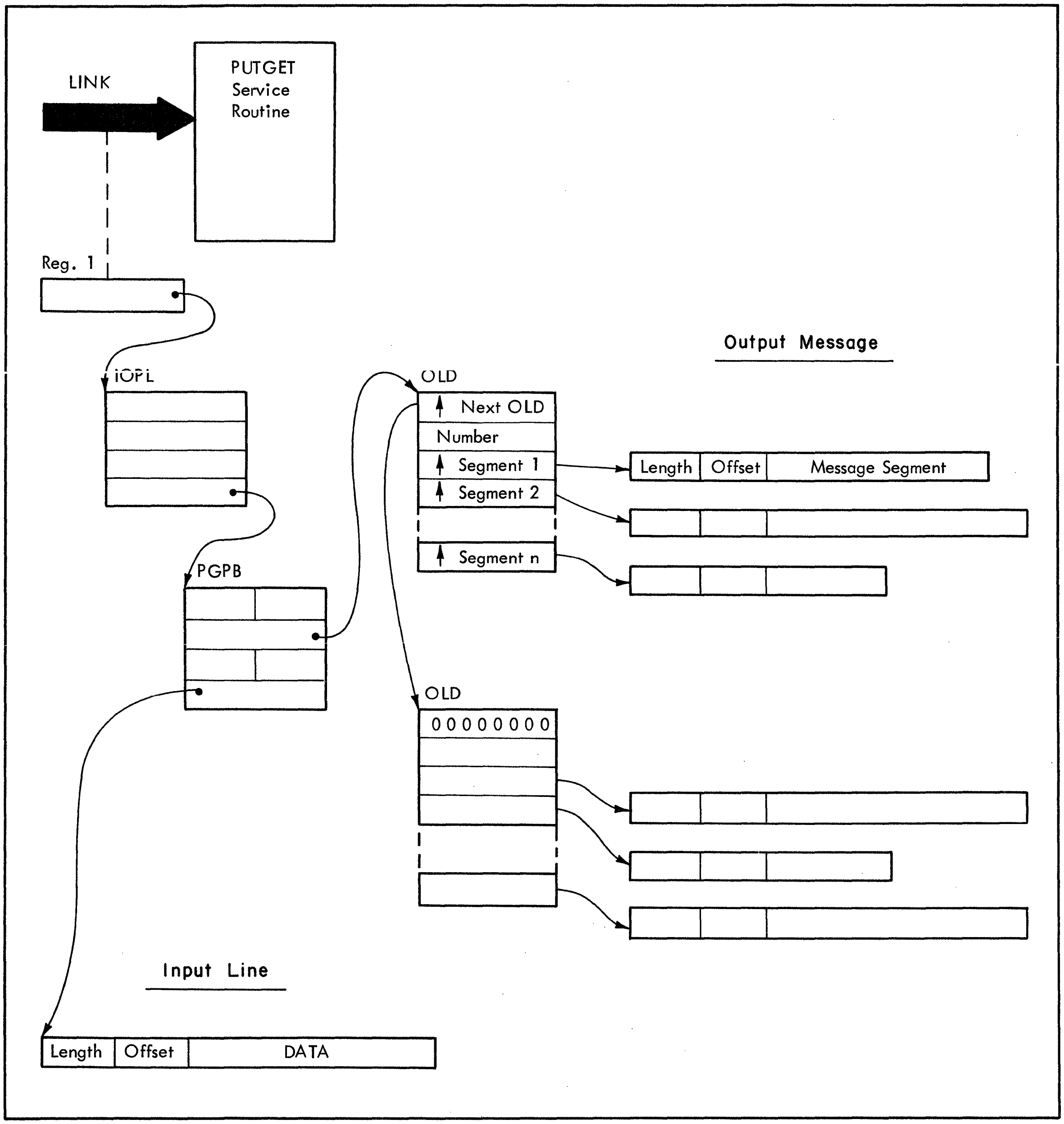


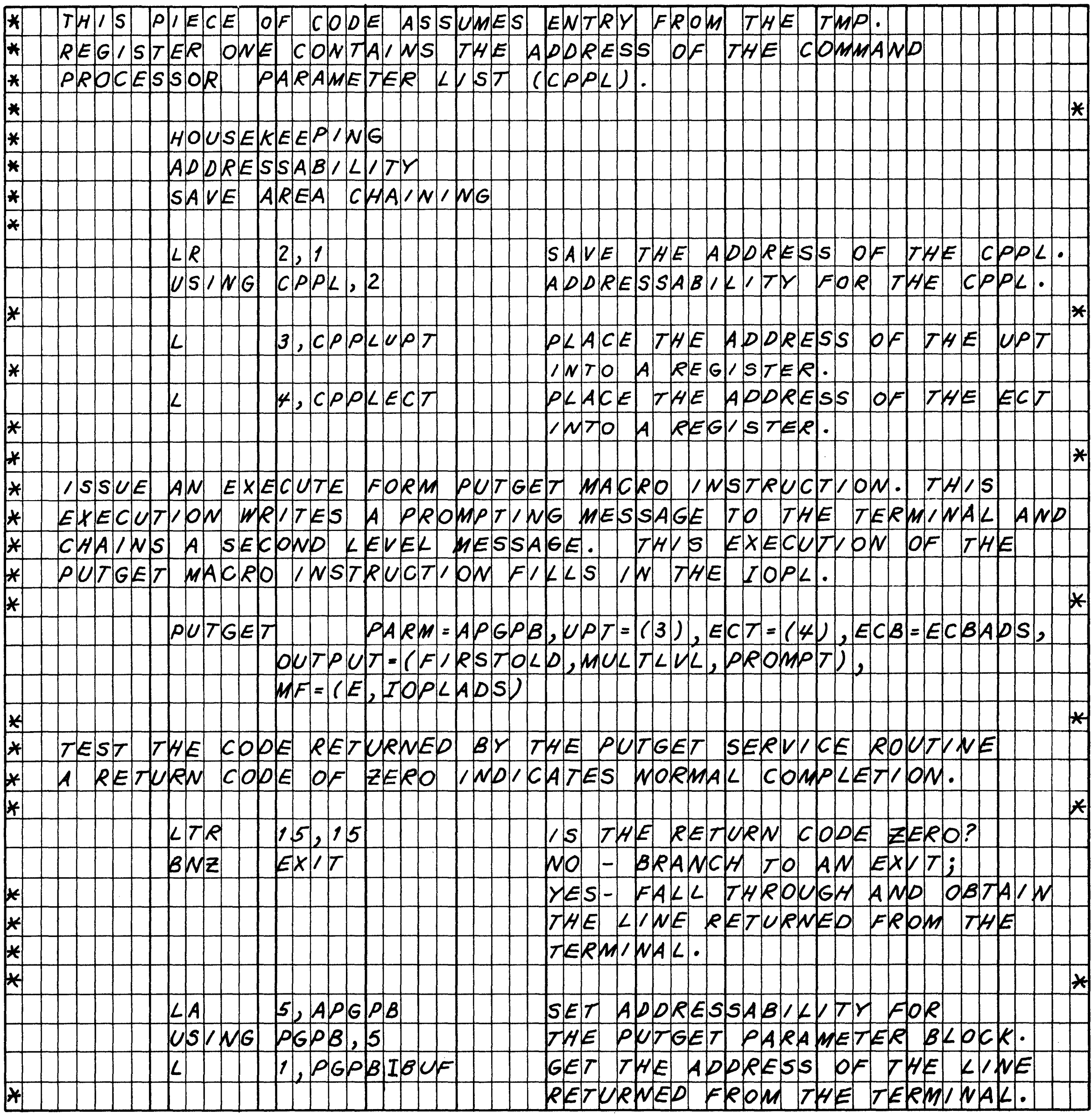
Figure 70. PUTGET Control Block Structure - Input Line Returned

An Example of PUTGET

Figure 71 is an example of the code required to execute the PUTGET macro instruction. The code uses a multilevel PROMPT message as the PUTGET output line. It assumes that a line of input will be returned from the terminal and tests only for a zero return code (PUTGET completed normally).

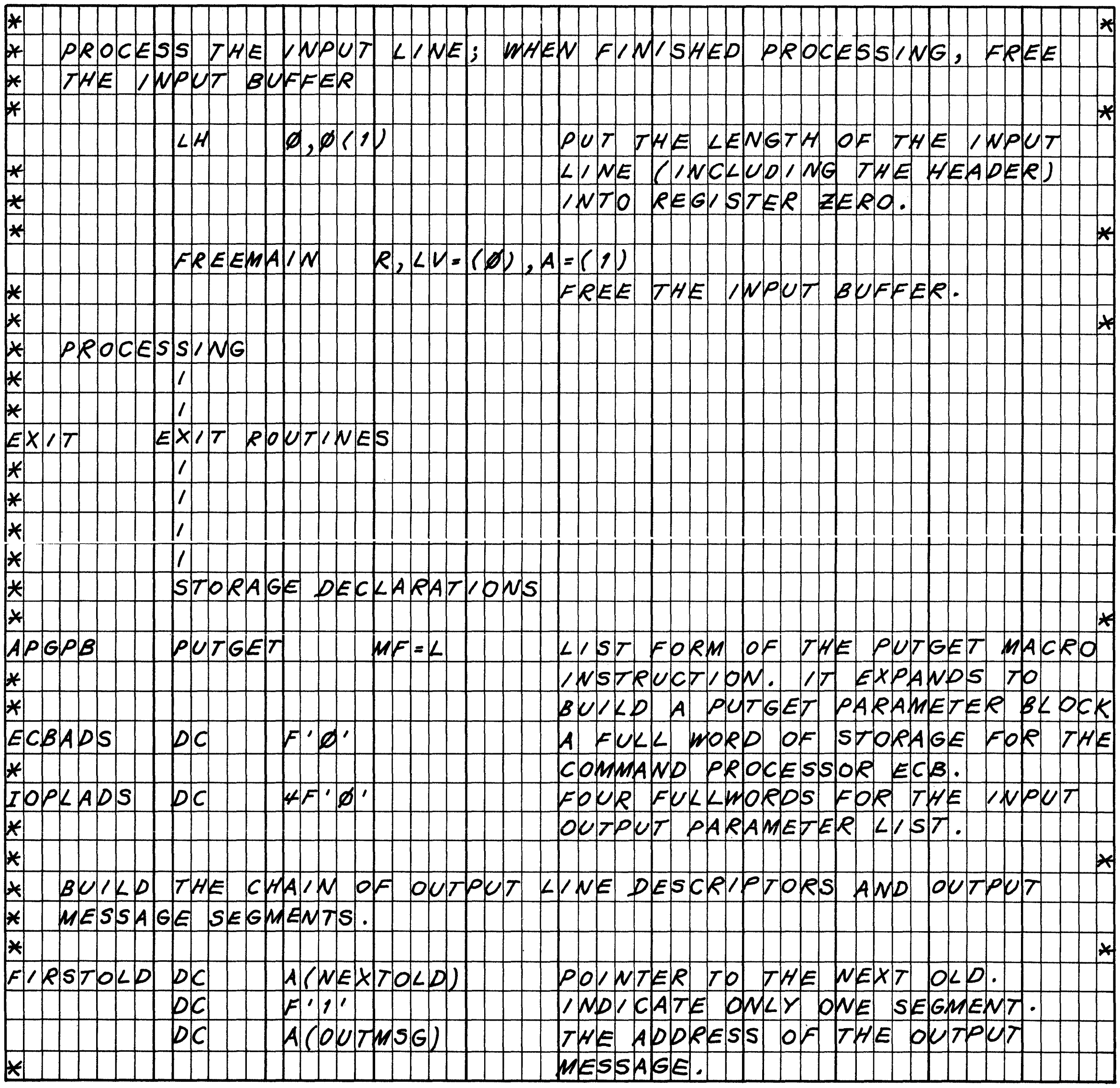
The execute form of the PUTGET macro instruction builds the I/O parameter list, using the addresses of the user profile table and the environment control table supplied in the Command Processor Parameter List. In addition, the I/O parameter list contains the address of an ECB built by the code, and the address of the list form of the PUTGET macro instruction as the PUTGET Parameter Block address.

Note that the TERMPUT, TERMGET, and ENTRY operands are not coded; the default values are used. Note also that this code is effective only if the top element of the input stack indicates a non-procedure as the current source of input.



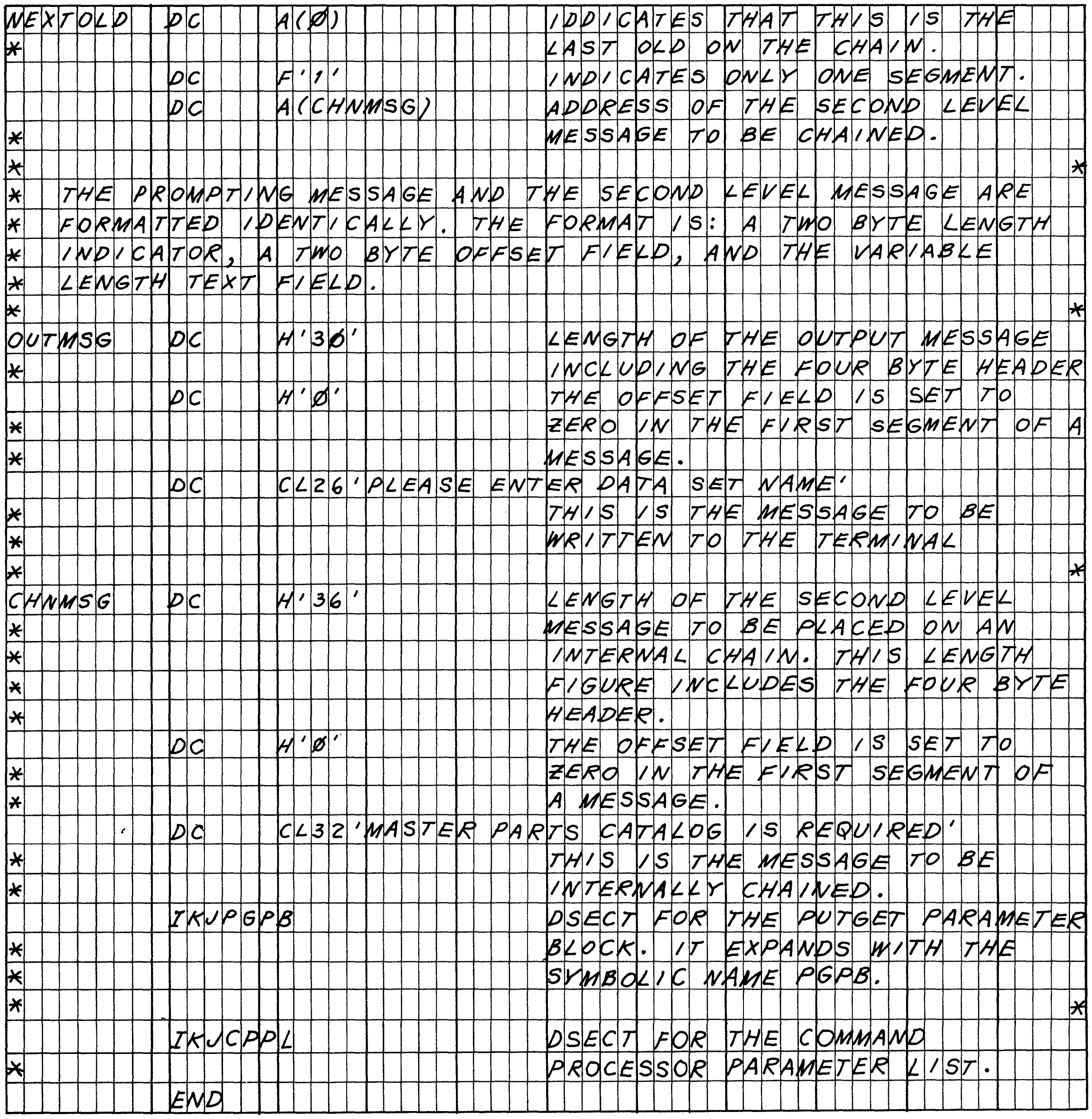
```
*  THIS PIECE OF CODE ASSUMES ENTRY FROM THE TMP.
*  REGISTER ONE CONTAINS THE ADDRESS OF THE COMMAND
*  PROCESSOR  PARAMETER LIST (CPPL).
*                                                          *
*        HOUSEKEEPING
*        ADDRESSABILITY
*        SAVE AREA CHAINING
*
         LR    2,1            SAVE THE ADDRESS OF THE CPPL.
         USING CPPL,2         ADDRESSABILITY FOR THE CPPL.
*                                                          *
         L     3,CPPLUPT      PLACE THE ADDRESS OF THE UPT
*                             INTO A REGISTER.
         L     4,CPPLECT      PLACE THE ADDRESS OF THE ECT
*                             INTO A REGISTER.
*                                                          *
*  ISSUE AN EXECUTE FORM PUTGET MACRO INSTRUCTION. THIS
*  EXECUTION WRITES A PROMPTING MESSAGE TO THE TERMINAL AND
*  CHAINS A SECOND LEVEL MESSAGE.  THIS EXECUTION OF THE
*  PUTGET MACRO INSTRUCTION FILLS IN THE IOPL.
*                                                          *
         PUTGET     PARM=APGPB,UPT=(3),ECT=(4),ECB=ECBADS,
               OUTPUT=(FIRSTOLD,MULTLVL,PROMPT),
               MF=(E,IOPLADS)
*                                                          *
*  TEST THE CODE RETURNED BY THE PUTGET SERVICE ROUTINE
*  A RETURN CODE OF ZERO INDICATES NORMAL COMPLETION.
*                                                          *
         LTR   15,15          IS THE RETURN CODE ZERO?
         BNZ   EXIT           NO - BRANCH TO AN EXIT;
*                             YES- FALL THROUGH AND OBTAIN
*                             THE LINE RETURNED FROM THE
*                             TERMINAL.
*                                                          *
         LA    5,APGPB        SET ADDRESSABILITY FOR
         USING PGPB,5         THE PUTGET PARAMETER BLOCK.
         L     1,PGPBIBUF     GET THE ADDRESS OF THE LINE
*                             RETURNED FROM THE TERMINAL.
```

Figure 71. Coding Example -- PUTGET Multi-Level PROMPT Message (Part 1 of 3)



```
*                                                          *
*  PROCESS THE INPUT LINE; WHEN FINISHED PROCESSING, FREE
*  THE INPUT BUFFER
*                                                          *
         LH    Ø,Ø(1)         PUT THE LENGTH OF THE INPUT
*                             LINE (INCLUDING THE HEADER)
*                             INTO REGISTER ZERO.
*                                                          *
         FREEMAIN   R,LV=(Ø),A=(1)
*                             FREE THE INPUT BUFFER.
*                                                          *
*  PROCESSING
*        /
*        /
EXIT    EXIT ROUTINES
*        /
*        /
*        /
*        /
*        STORAGE DECLARATIONS
*                                                          *
APGPB    PUTGET     MF=L     LIST FORM OF THE PUTGET MACRO
*                             INSTRUCTION. IT EXPANDS TO
*                             BUILD A PUTGET PARAMETER BLOCK
ECBADS   DC    F'Ø'          A FULL WORD OF STORAGE FOR THE
*                             COMMAND PROCESSOR ECB.
IOPLADS  DC    4F'Ø'         FOUR FULLWORDS FOR THE INPUT
*                             OUTPUT PARAMETER LIST.
*                                                          *
*  BUILD THE CHAIN OF OUTPUT LINE DESCRIPTORS AND OUTPUT
*  MESSAGE SEGMENTS.
*                                                          *
FIRSTOLD DC    A(NEXTOLD)     POINTER TO THE NEXT OLD.
         DC    F'1'           INDICATE ONLY ONE SEGMENT.
         DC    A(OUTMSG)      THE ADDRESS OF THE OUTPUT
*                             MESSAGE.
```

Figure 71. Coding Example -- PUTGET Multi-Level PROMPT Message (Part 2 of 3)



```
NEXTOLD  DC    A(Ø)                IDDICATES THAT THIS IS THE
*                                  LAST OLD ON THE CHAIN.
         DC    F'1'                INDICATES ONLY ONE SEGMENT.
         DC    A(CHNMSG)           ADDRESS OF THE SECOND LEVEL
*                                  MESSAGE TO BE CHAINED.
*                                                                     *
*   THE PROMPTING MESSAGE AND THE SECOND LEVEL MESSAGE ARE
*   FORMATTED IDENTICALLY. THE FORMAT IS: A TWO BYTE LENGTH
*   INDICATOR, A TWO BYTE OFFSET FIELD, AND THE VARIABLE
*   LENGTH TEXT FIELD.
*                                                                     *
OUTMSG   DC    H'3Ø'               LENGTH OF THE OUTPUT MESSAGE
*                                  INCLUDING THE FOUR BYTE HEADER
         DC    H'Ø'                THE OFFSET FIELD IS SET TO
*                                  ZERO IN THE FIRST SEGMENT OF A
*                                  MESSAGE.
         DC    CL26'PLEASE ENTER DATA SET NAME'
*                                  THIS IS THE MESSAGE TO BE
*                                  WRITTEN TO THE TERMINAL
*                                                                     *
CHNMSG   DC    H'36'               LENGTH OF THE SECOND LEVEL
*                                  MESSAGE TO BE PLACED ON AN
*                                  INTERNAL CHAIN. THIS LENGTH
*                                  FIGURE INCLUDES THE FOUR BYTE
*                                  HEADER.
         DC    H'Ø'                THE OFFSET FIELD IS SET TO
*                                  ZERO IN THE FIRST SEGMENT OF
*                                  A MESSAGE.
         DC    CL32'MASTER PARTS CATALOG IS REQUIRED'
*                                  THIS IS THE MESSAGE TO BE
*                                  INTERNALLY CHAINED.
         IKJPGPB                   DSECT FOR THE PUTGET PARAMETER
*                                  BLOCK. IT EXPANDS WITH THE
*                                  SYMBOLIC NAME PGPB.
*                                                                     *
         IKJCPPL                   DSECT FOR THE COMMAND
*                                  PROCESSOR PARAMETER LIST.
         END
```

Figure 71. Coding Example -- PUTGET Multi-Level PROMPT Message (Part 3 of 3)

Return Codes From PUTGET

When the PUTGET Service Routine returns control to the program that invoked it, it provides one of the following return codes in general register 15.

CODE decimal — MEANING

0 PUTGET completed normally.
The line obtained came from the terminal.

4 PUTGET completed normally.
The line obtained did not come from the terminal. (MODE messages only)

8 The PUTGET service routine did not complete. An attention interrupt occurred during the execution of PUTGET, and the attention handler turned on the completion bit in the communications ECB.

12 No prompting was allowed on a PROMPT request. Either the user at the terminal requested no prompting with the PROFILE command, or the current source of input is an in-storage list.

12 A line could not be obtained after a MODE request. A chain of second level informational messages exists, and the current stack element is non-terminal, but the terminal user did not request PAUSE processing with the PROFILE command. The messages are therefore not available to him.

16 The NOWAIT option was specified for TPUT and no line was put out or received.

20 The NOWAIT option was specified for TGET and no line was received.

24 Invalid parameters were supplied to the PUTGET service routine.

28 A conditional GETMAIN was issued by PUTGET for output buffers and there was not sufficient space to satisfy the request.

Using the TGET/TPUT SVC for Terminal I/O

A supervisor call routine, SVC 93, reached through the TGET and TPUT macro instructions, provides a route for program I/O to a terminal. The Basic Sequential Access Method, the Queued Sequential Access Method, and the TSO I/O Service Routines all use SVC93 to process terminal I/O. You can use this method in any TSO routines you write, and in any applications programs that run under TSO control. If you do use TGET/TPUT in an applications program however, that program becomes TSO dependent. The TGET and TPUT macro instructions become NOPs in a batch environment.

The TGET and TPUT macro instructions do not require that you build control blocks for their use. The operands you code into each of these macro instructions specify the location and size of the TGET or TPUT buffers, and the SVC functions you want performed. The functions provided by the TGET/TPUT SVC are not as extensive, however, as those provided by the Terminal I/O service routines.

Both the TGET and the TPUT macro instructions have a standard form and a register form.

This section discusses:

- The TPUT Macro Instruction
- The TGET Macro Instruction
- Formatting the TGET/TPUT Parameter Registers
- Examples of TGET and TPUT

The TPUT Macro Instruction - Writing a Line to the Terminal

Use the TPUT macro instruction (SVC 93) to transmit a line of output to the terminal. You can use the TPUT macro instruction in any TSO routines you write, and in any applications programs to be run under TSO. Note however, that TPUT does not provide message ID stripping, text insertion, or second level message chaining. If you require these features, use the PUTLINE macro instruction.

Figure 72 shows the format of the TPUT macro instruction; the figure combines the standard and the register form. Each of the operands is explained following the figure. Appendix B describes the notation used to define macro instructions.

```
[symbol]   TPUT    buffer address,buffer size
                   ┌                                                       ┐
                   │┌,EDIT    ┐┌         ┐┌        ┐┌         ┐            │
                   ││,ASIS    ││,WAIT    ││,NOHOLD ││,NOBREAK │            │
                   │└,CONTROL ┘└,NOWAIT  ┘└,HOLD   ┘└,BREAKIN ┘            │
                   │                                                       │
                   │                         ┌,HIGHP┐ ┌,TJID=id         ┐  │
                   │                         └,LOWP ┘ └,TJIDLOC=address ┘  │
                   │                                                       │
                   └,R                                                     ┘
```

Figure 72. The TPUT Macro Instruction -- Standard and Register Forms

buffer address

Standard form: The address of the buffer that holds your line of output. This can be any address acceptable in an RX instruction, or the address can be placed in one of the general registers 1-12, and that register specified within parenthesis.

Register form: The register which contains the parameters to be passed in register 1 to the TPUT SVC. When the R format is specified, this operand must be in one of the general registers 1-12, and that register specified within parentheses. See the section headed 'Formatting the TGET/TPUT Parameter Registers' for a discussion of the register contents.

buffer size

Standard form: The size of the output buffer in bytes. The allowable range is from 0 through 32,767 bytes. You can specify this buffer size directly as a number, or you can place the buffer size into one of the general registers 0, or 2-12, and specify that register within parentheses.

Register form: The register which contains the parameters to be passed in register 0 to the TPUT SVC. When the R format is specified this operand must be in one of the general registers 0, or 2-12, and that register specified within parentheses. See the section "Formatting the TGET/TPUT Parameter Registers" for a discussion of the register contents.

R

Indicates that this is the register form of the TPUT macro instruction. You must place the parameters you want passed to the TPUT SVC into two registers and specify those registers as the first two operands of the macro instruction. The parameters must be arranged in the registers in the format shown in the section headed 'Formatting the TGET/TPUT Parameter Registers'. The R operand and all other optional operands are mutually exclusive.

If both R and any other optional operands are coded, the macro will not expand.

EDIT

Indicates that in addition to minimal editing (see ASIS), the following TPUT functions are requested:

a. All trailing blanks are removed before the line is written to the terminal. If a blank line is sent, the terminal vertically spaces one line.

b. Control characters are added to the end of the output line to position the carrier to the beginning of the next line.

c. All terminal control characters (for example: bypass, restore, horizontal tab, new line) are replaced with a printable character. "Backspace" is an exception; see (d.) under ASIS.

EDIT is the default value among the EDIT, ASIS, and CONTROL operands.

ASIS

Indicates that minimal editing is to be performed by the TPUT SVC as follows:

a. The line of output is translated from EBCDIC to terminal code. Invalid characters are converted to a printable character to prevent program caused I/O errors. This does not mean that all unprintable characters will be eliminated. "Restore", 'upper case", "lower case", "bypass", and "bell ring", for example, might be valid but nonprinting characters at some terminals. (See CONTROL).

b. Transmission control characters are added.

c. EBCDIC "NL", placed at the end of the message, indicates to the TPUT SVC that the carrier is to be returned at the end of the line. "NL" is replaced with whatever is necessary for that particular terminal type to cause the carrier to return. This "NL" processing occurs only if you specify ASIS, and the "NL" is the last character in your message.

If you specify EDIT, "NL" is handled as described in (c.) under EDIT.

If the "NL" is embedded in your message, a semicolon is substituted for "NL" and sent to the terminal. No idle characters are added (see f. below). This may cause overprinting, particularly on terminals that require a line-feed character to position the carrier on a new line.

d. If you have used "backspace" in your output message, but the "backspace" character does not exist on the terminal type to which the message is being routed, the "backspace" character is removed from the output message.

e. Control characters are added as needed to cause the message to print on several lines if the output line is longer than the terminal line size.

f. A sufficient number of idle characters is added to the end of each output line to prevent the transmission of output to the terminal while the carrier is being returned to the left-hand margin.

CONTROL
Indicates that this line is composed of terminal control characters and will not print or move the carrier on the terminal. This option should be used for transmission of characters such as "bypass", "restore", or "bell ring".

WAIT
Specifies that control will not be returned to the program that issued the TPUT macro instruction until the output line has been placed into a terminal output buffer. If no buffers are available, the issuing program will be placed into a wait state until buffers become available, and the output line is placed into them.
WAIT is the default value for the WAIT and NOWAIT operands.

NOWAIT
Specifies that control should be returned to the program that issued the TPUT macro instruction, whether or not a terminal output buffer is available for the output line. If no buffer is available, the TPUT SVC returns a code of 04 (hex) in register 15.

NOHOLD
Indicates that control is to be returned to the program that issued the TPUT macro instruction as soon as the output line has been placed in terminal output buffers.
NOHOLD is the default value for the NOHOLD and HOLD operands.

HOLD
Specifies that the program that issued the TPUT macro instruction cannot continue its processing until this output line has been written to the terminal or deleted.

NOBREAK
Specifies that if the terminal user has started to enter input, he is not to be interrupted. The output message is placed on the output queue to be printed after the terminal user has completed the line.
NOBREAK is the default value for the NOBREAK and BREAKIN operands.

BREAKIN
Specifies that output has precedence over input. If the user at the terminal has started to enter input, he is interrupted, and this output line is sent. Any data that was received before the interruption is kept and displayed at the terminal following this output line.

HIGHP
Specifies that this message must be sent to the terminal, even though the destination terminal has disallowed messages from other terminals. This operand counters the effect of the interterminal communication bit when set in the terminal status block[1] (TSB). (The HIGHP operand is used by the OPERATOR SEND subcommand and the SEND operator command.) The operand is recognized only if the issuing task is operating under zero protection key. The TJID keyword must also be specified. HIGHP is the default if neither HIGHP nor LOWP is specified, and the issuing program is operating under zero protection key.

----

[1] See the TSO Control Program, Program Logic Manual for a description of the terminal status block (TSB).

LOWP

Specifies that the TPUT with TJID module should test the interterminal communication bit in the terminal status block. If the user of the destination terminal allows interterminal messages, this message will be sent. If such messages are not allowed the message will not be sent, and the return code of '0C' will indicate no message was sent. The LOWP operand is recognized only when TJID is specified. The issuer must be operating under zero protection key.

If LOWP is specified, the issuing program should have an alternate method of transmitting the message to the terminal user. For example, a message data set could be used.

TJID or TJIDLOC

Specifies the TJID (terminal job identifier) of the target terminal, or the address of that TJID. This facility is used for supervisor communication with the terminal, and for inter-user conversation between terminals (the SEND command). If this option is used, NOHOLD is the required option and is defaulted to. If you specify TJID, you must supply a TJID number, or the number of a register containing the TJID number. The register number must be enclosed within parentheses. If TJIDLOC is used, you must supply the address of a halfword containing the TJID.
TJID or TJIDLOC can be specified in registers 2-12, right adjusted. The TJID is located in the 2 byte TJBTJID field of the Terminal Job Block associated by USERID (the TJBUSER field) with the user you wish to send to. See Appendix A for a description of the Terminal Job Block.

Note: If a TPUT without TJID is coded in a background program, the result is a NOP. If however, the TPUT specifies TJID, the message is sent to the target terminal.

RETURN CODES FROM TPUT

When it returns control to the program that invoked it, the TPUT SVC supplies one of the following return codes in general register 15.

Code (hexadecimal)	Meaning
00	TPUT completed successfully.
04	NOWAIT was specified and no terminal output buffer was available.
08	An attention interruption occurred while the TPUT SVC routine was processing.
0C	A TPUT macro instruction with a TJID operand was issued but the user at the terminal indicated by the TJID requested that inter-terminal messages not be printed on his terminal. The message was not sent.
10	Invalid parameters were passed to the TPUT SVC.
14	The terminal has been disconnected and could not be reached.

The TGET Macro Instruction -- Getting a Line From the Terminal

Use the TGET macro instruction to read a line of input from the terminal. A line of input is defined as all the data between the beginning of the input line and a line-end delimiter. A line-end delimiter is any character or combination of characters which causes the carrier to return to the left-hand margin on a new line, or which terminates transmission from the terminal.

You can use the TGET macro instruction in any TSO routines, and in any applications programs to be run under TSO. Note however, that TGET does not provide access to in-storage lists, nor does it perform any type of logical line processing on the returned line. If you require these features, use the GETLINE macro instruction.

Each time TGET returns control to your program, register 1 contains the number of bytes of data actually moved from the terminal to your input buffer. If your buffer is smaller than the line of input entered at the terminal, only as much of the input line as can be contained in the input buffer is moved. Return code 0C indicates that only part of the line was obtained by TGET. You must then issue as many TGET macro instruction as are required to get the rest of the line of input.

Figure 73 shows the format of the TGET macro instruction; it combines the standard and the register form. Each of the operands is explained following the figure. Appendix B describes the notation used to define macro instructions.

```
┌──────────┬──────┬───────────────────────────────────────────────────────┐
│          │      │                           ┌┌     ┐┌       ┐┐           │
│ [symbol] │ TGET │buffer address,buffer size ││,EDIT││,WAIT   ││           │
│          │      │                           ││,ASIS││,NOWAIT ││           │
│          │      │                           │└     ┘└       ┘│           │
│          │      │                           │ ,R             │           │
│          │      │                           └                ┘           │
└──────────┴──────┴───────────────────────────────────────────────────────┘
```

Figure 73. The TGET Macro Instruction -- Standard and Register Forms

buffer address

Standard form: The address of the buffer that is to receive the input line. This can be any address acceptable in an RX instruction, or the address can be placed in one of the general registers 1-12, and that register specified within parentheses.

Register form: The register which contains the parameters to be passed in register 1 to the TGET SVC. When the R format is specified, this operand must be in one of the general registers 1-12, and that register specified within parentheses. See the section headed 'Formatting the TGET/TPUT Parameter Registers' for a discussion of the register contents.

buffer size

Standard form: The size of the input buffer in bytes. The allowable range is from 0 through 32,767 bytes. You can specify this buffer size directly as a number, or you can place the buffer size into one of the general registers 0, or 2-12, and specify that register within parentheses.

Register form: The register which contains the parameters to be passed in register 0 to the TGET SVC. When the R format is specified this operand must be in one of the general registers 0, or 2-12, and that register specified within parentheses. See the topic 'Formatting the TGET/TPUT Parameter Registers' for a discussion of the register contents.

R

Indicates that this is the register form of the TGET macro instruction. You must place the parameters you want passed to the TGET SVC into two registers and specify those registers as the first two operands of the macro instruction. The parameters must be arranged in the registers in the format shown in the section headed 'Formatting the TGET/TPUT Parameter Registers'.
The R operand and all other optional operands are mutually exclusive.
If both R and any other optional operands are coded, the macro will not expand.

EDIT

Specifies that in addition to minimal editing (see ASIS), the following TGET functions are requested:

a. All terminal control characters (that is, nongraphic characters such as bypass, line feed, restore, prefix and the character immediately following it, etc.) are removed from the data.

b. The horizontal tab (HT) character and the backspace (BS) character, when backspace is not used for character deletion, remain in the data.

c. The buffer is filled out with blanks, if the returned input line is shorter than the input buffer length. These blanks are not included in the character count returned in register 1.

EDIT is the default value for the EDIT and ASIS operands.

ASIS

Specifies that minimal editing is done as described below:

a. Transmission control characters are removed.

b. The returned input line is translated from terminal code to EBCDIC. Invalid characters are compressed out of the data.

c. Line deletion and character deletion are performed according to the specifications in the Terminal Status Block.

d. New line (NL), carriage return (CR), and line feed (LF) characters, if present at the end of the line, are not included in the data count returned in register one.

e. After the input message has been received, the carrier is returned to the left-hand margin of the next line before any output to the terminal is allowed.

WAIT

Specifies that control will not be returned to the program that issued the TGET macro instruction until the input line has been placed into your input buffer. If an input line is not available from the terminal, the issuing program is placed into a wait state until a line becomes available and is read into your input buffer.
WAIT is the default value for the WAIT and NOWAIT operands.

NOWAIT

Specifies that control should be returned to the program that issued the TGET macro instruction, whether or not an input line is available from the terminal. If no line is returned, the TGET SVC returns a code of 04 (hex) in register 15.

RETURN CODES FROM TGET

When it returns control to the program that invoked it, the TGET SVC supplies the length of the message moved into your buffer in register 1, and one of the following return codes in general register 15.

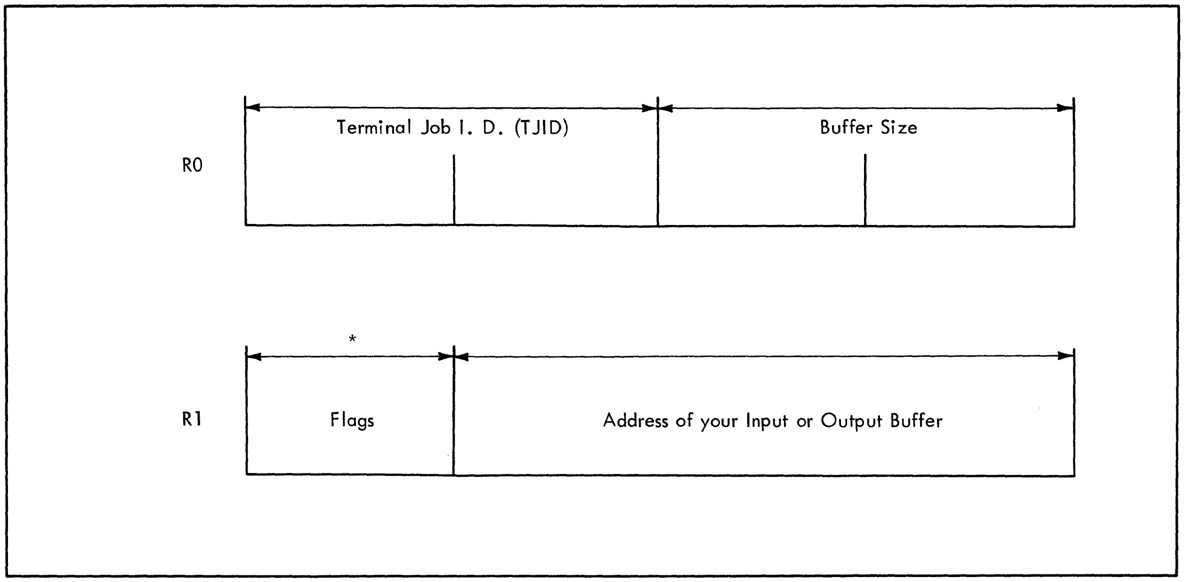
Code (hexadecimal)	Meaning
00	TGET completed successfully. Register 1 contains the length of the input line read into your input buffer.
04	NOWAIT was specified and no input was available to be read into your input buffer.
08	An attention interruption occurred while the TGET SVC routine was processing.
0C	Your input buffer was not large enough to accept the entire line of input entered at the terminal. Subsequent TGET macro instructions will obtain the rest of the input line.
10	Invalid parameters were passed to the TGET SVC.
14	The terminal has been disconnected and could not be reached.

Formatting the TGET/TPUT Parameter Registers

If you use the Register format of the TGET or TPUT macro instruction, you must code the parameters you want passed to the TGET/TPUT SVC into two registers. You specify these two registers enclosed in parentheses as the first two operands of the TGET or TPUT macro instruction, followed by the R operand to indicate that you are executing the register form of the macro instruction.

If the registers you specify as the first and second operand of the macro instruction are register 1 and register 0 respectively, the TGET or TPUT macro instruction expands directly to the TGET/TPUT SVC. If you specify other permissible registers, registers 2-12, the macro expands to load registers one and zero from the registers you specify before issuing the SVC.

The registers must be formatted as shown in Figure 74.



R0
Terminal Job I. D. (TJID)
Buffer Size
R1
*
Flags
Address of your Input or Output Buffer

Figure 74. TGET/TPUT Parameter Registers

* Flags
One Byte

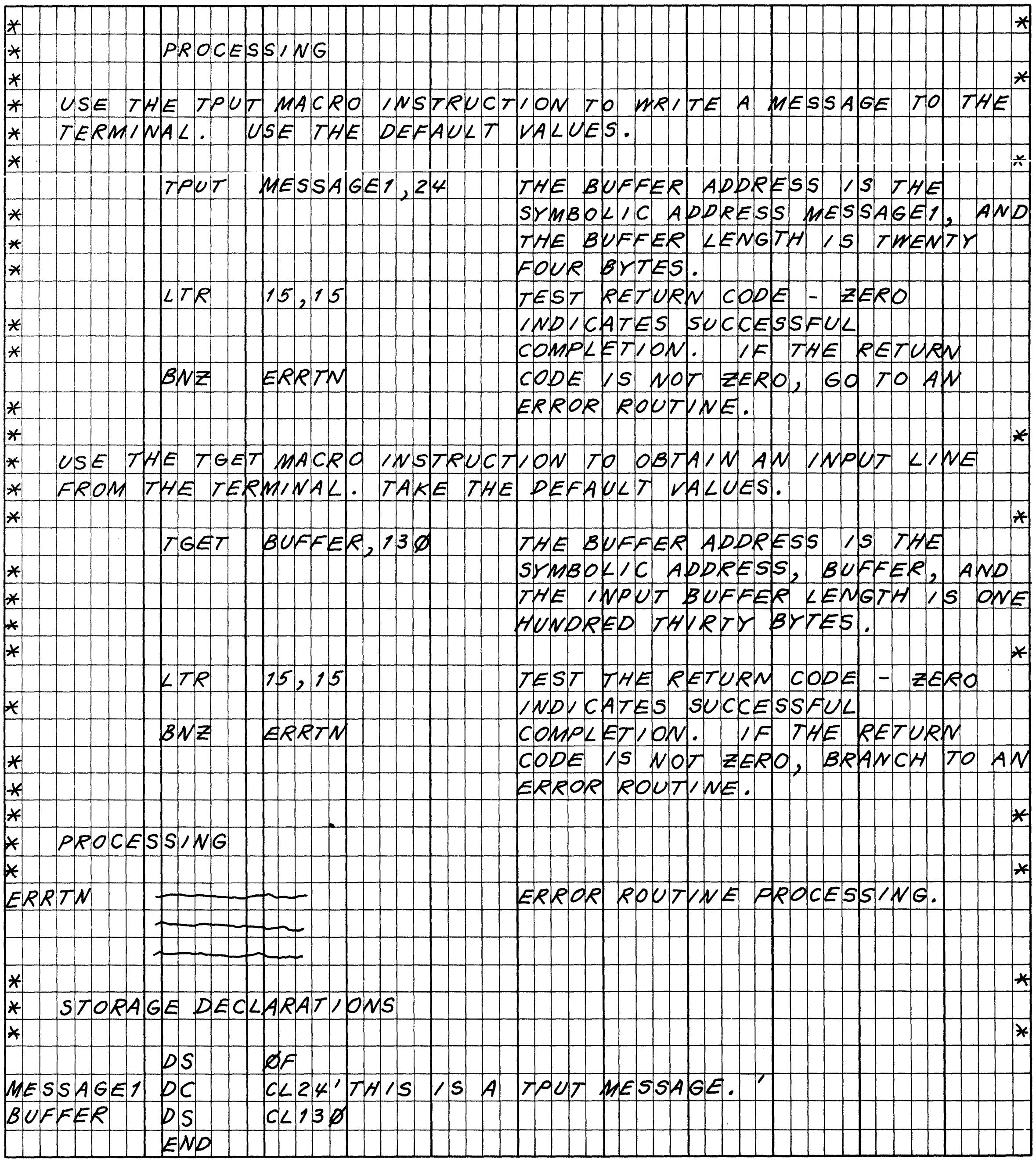
0... ...	Always set to 0 for TPUT.
1...	Always set to 1 for TGET.
.xx.	Reserved bits.
...0	WAIT processing is requested.
...1	NOWAIT processing is requested.
.... 0...	NOHOLD processing is requested.
.... 1...	HOLD processing is requested.
.... .0..	NOBREAK processing is requested.
.... .1..	BREAK processing is requested.
.... ..00	EDIT processing is requested.
.... ..01	ASIS processing is requested.
.... ..10	CONTROL processing is requested.

Coding Examples of TGET and TPUT Macro Instructions

The following coding examples show different ways to use the TGET and TPUT macro instructions.

EXAMPLES OF BOTH TPUT AND TGET USING THE DEFAULT VALUES

Figure 75 shows both a TPUT and a TGET macro instruction. They both take the default values; that is, the TPUT macro instruction defaults to EDIT, WAIT, NOHOLD, and NOBREAK; and the TGET macro instruction defaults to EDIT and WAIT.



```
*                                                          *
*        PROCESSING
*                                                          *
*  USE THE TPUT MACRO INSTRUCTION TO WRITE A MESSAGE TO THE
*  TERMINAL.  USE THE DEFAULT VALUES.
*                                                          *
         TPUT  MESSAGE1,24    THE BUFFER ADDRESS IS THE
*                             SYMBOLIC ADDRESS MESSAGE1, AND
*                             THE BUFFER LENGTH IS TWENTY
*                             FOUR BYTES.
         LTR   15,15          TEST RETURN CODE - ZERO
*                             INDICATES SUCCESSFUL
*                             COMPLETION.  IF THE RETURN
         BNZ   ERRTN          CODE IS NOT ZERO, GO TO AN
*                             ERROR ROUTINE.
*                                                          *
*  USE THE TGET MACRO INSTRUCTION TO OBTAIN AN INPUT LINE
*  FROM THE TERMINAL.  TAKE THE DEFAULT VALUES.
*                                                          *
         TGET  BUFFER,13Ø     THE BUFFER ADDRESS IS THE
*                             SYMBOLIC ADDRESS, BUFFER, AND
*                             THE INPUT BUFFER LENGTH IS ONE
*                             HUNDRED THIRTY BYTES.
*                                                          *
         LTR   15,15          TEST THE RETURN CODE - ZERO
*                             INDICATES SUCCESSFUL
         BNZ   ERRTN          COMPLETION.  IF THE RETURN
*                             CODE IS NOT ZERO, BRANCH TO AN
*                             ERROR ROUTINE.
*                                                          *
*  PROCESSING
*                                                          *
ERRTN                         ERROR ROUTINE PROCESSING.


*                                                          *
*  STORAGE DECLARATIONS
*                                                          *
         DS    ØF
MESSAGE1 DC    CL24'THIS IS A TPUT MESSAGE.'
BUFFER   DS    CL13Ø
         END
```

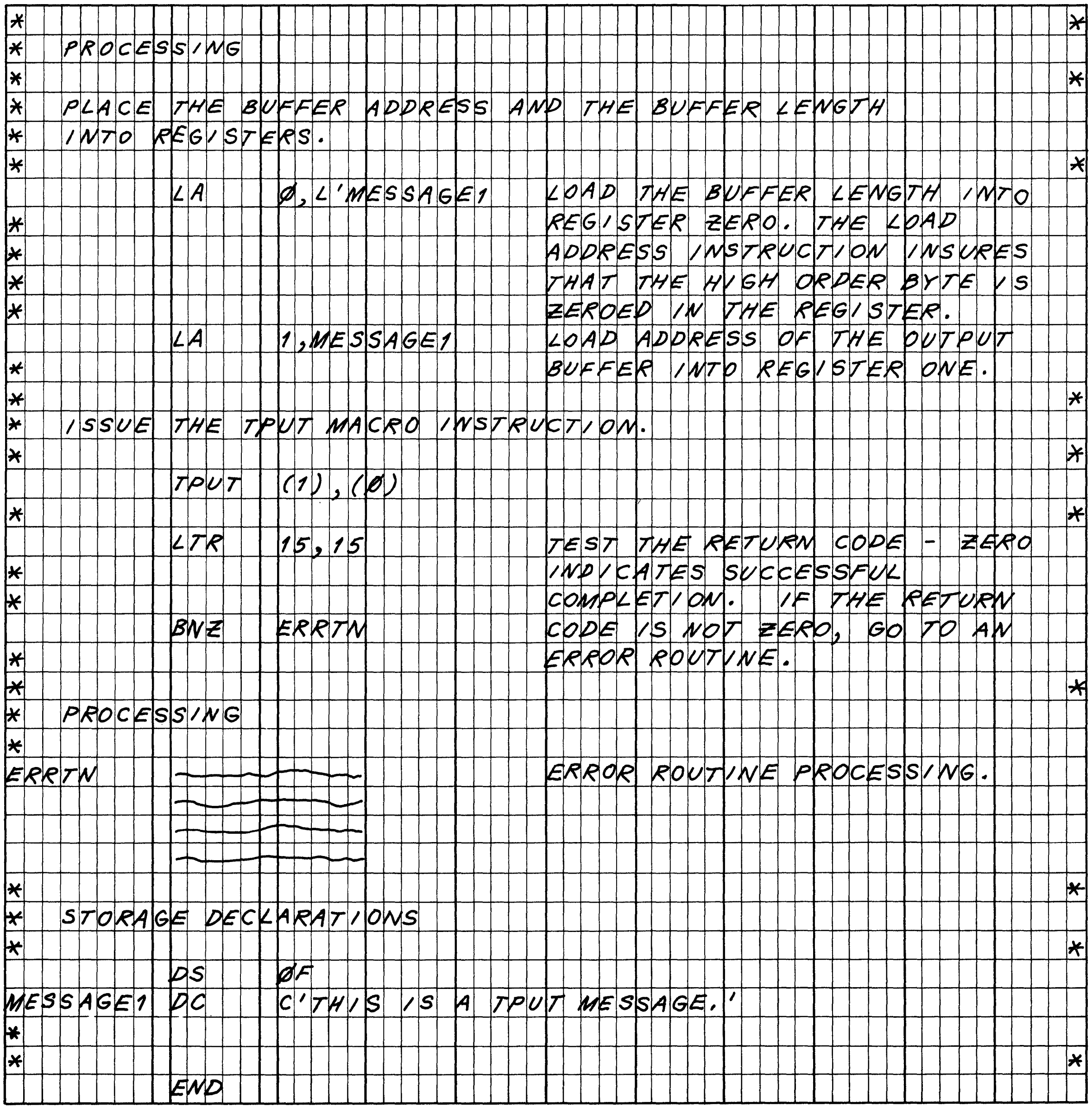
Figure 75. Coding Example -- of TPUT and TGET Macro Instructions Using the Default Values

The program issuing the TGET macro instruction will not be given control until a line of data is returned. The default value is WAIT. The input buffer will be padded with blanks if less than 130 characters were entered; the default is EDIT. Remember that the actual length of the data in the input buffer is returned in register 1.

EXAMPLE OF TPUT MACRO INSTRUCTION -- BUFFER ADDRESS AND BUFFER LENGTH IN REGISTERS.

In the coding example shown in Figure 76, the output message buffer address and length are loaded into registers, and those registers coded as operands in the TPUT macro instruction.

You might want to do this when, for example, the TPUT macro instruction is issued in a subroutine which receives, as parameters, a pointer to the message and the message length.

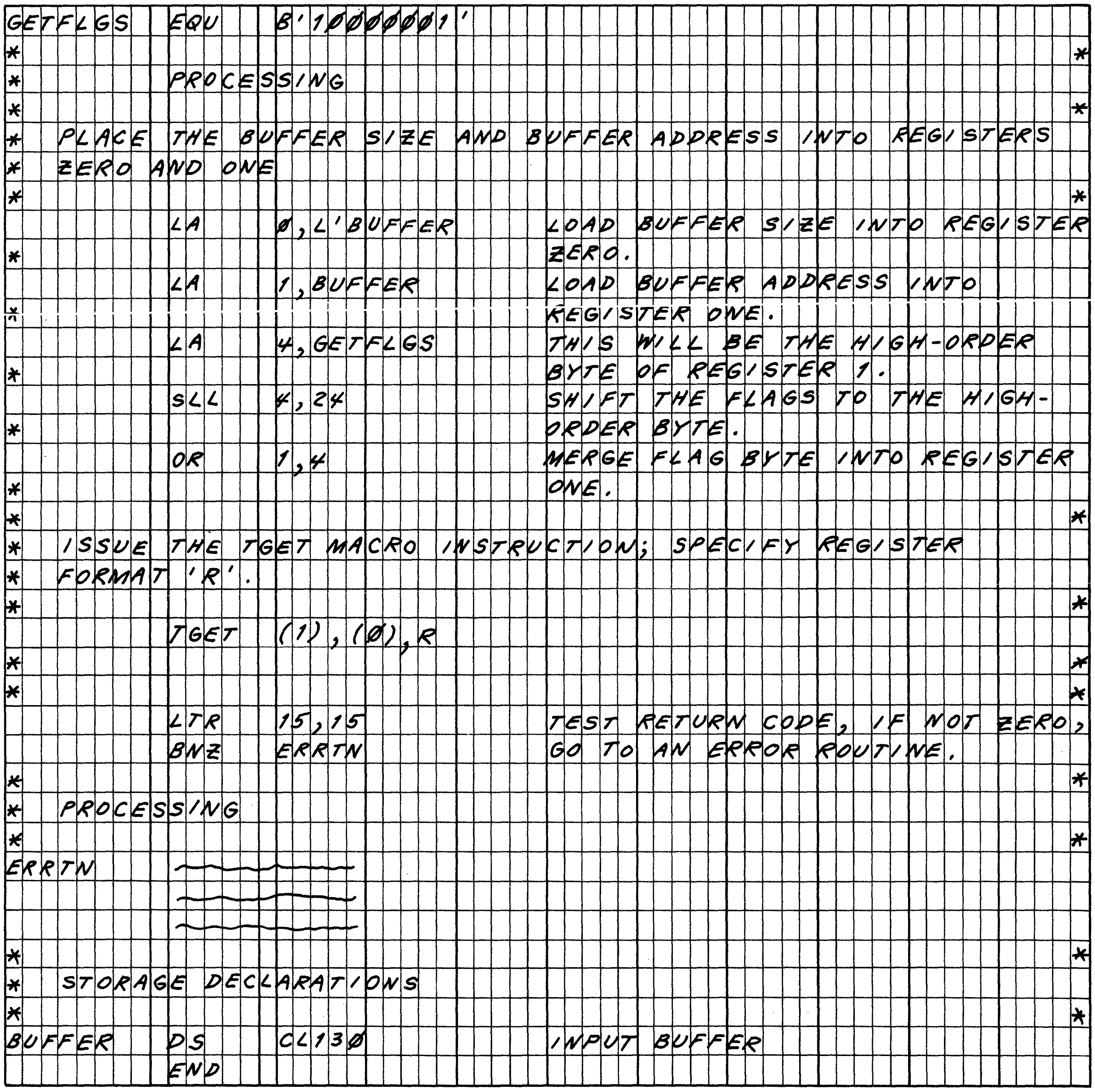


```
*                                                          *
*  PROCESSING
*                                                          *
*  PLACE THE BUFFER ADDRESS AND THE BUFFER LENGTH
*  INTO REGISTERS.
*                                                          *
         LA    Ø,L'MESSAGE1   LOAD THE BUFFER LENGTH INTO
*                             REGISTER ZERO. THE LOAD
*                             ADDRESS INSTRUCTION INSURES
*                             THAT THE HIGH ORDER BYTE IS
*                             ZEROED IN THE REGISTER.
         LA    1,MESSAGE1     LOAD ADDRESS OF THE OUTPUT
*                             BUFFER INTO REGISTER ONE.
*                                                          *
*  ISSUE THE TPUT MACRO INSTRUCTION.
*                                                          *
         TPUT  (1),(Ø)
*                                                          *
         LTR   15,15          TEST THE RETURN CODE - ZERO
*                             INDICATES SUCCESSFUL
*                             COMPLETION.  IF THE RETURN
         BNZ   ERRTN          CODE IS NOT ZERO, GO TO AN
*                             ERROR ROUTINE.
*                                                          *
*  PROCESSING
*
ERRTN                         ERROR ROUTINE PROCESSING.
*                                                          *
*  STORAGE DECLARATIONS
*                                                          *
         DS    ØF
MESSAGE1 DC    C'THIS IS A TPUT MESSAGE.'
*
*                                                          *
         END
```

Figure 76. Coding Example: TPUT Macro Instruction Buffer Address and Buffer Length in Registers

EXAMPLE OF THE TGET MACRO INSTRUCTION -- REGISTER FORMAT

Figure 77 shows the code necessary to issue a register format TGET macro instruction. The buffer length, buffer address, and the option flags are loaded into registers zero and one. Note that the flag byte in register one has been set to binary 10000001, indicating that this is a TGET macro instruction requesting ASIS processing. This means that only minimal editing will be performed on the input line.



```
GETFLGS  EQU   B'10000001'
*                                                          *
*        PROCESSING
*                                                          *
*  PLACE THE BUFFER SIZE AND BUFFER ADDRESS INTO REGISTERS
*  ZERO AND ONE
*                                                          *
         LA    0,L'BUFFER     LOAD BUFFER SIZE INTO REGISTER
*                             ZERO.
         LA    1,BUFFER       LOAD BUFFER ADDRESS INTO
*                             REGISTER ONE.
         LA    4,GETFLGS      THIS WILL BE THE HIGH-ORDER
*                             BYTE OF REGISTER 1.
         SLL   4,24           SHIFT THE FLAGS TO THE HIGH-
*                             ORDER BYTE.
         OR    1,4            MERGE FLAG BYTE INTO REGISTER
*                             ONE.
*                                                          *
*  ISSUE THE TGET MACRO INSTRUCTION; SPECIFY REGISTER
*  FORMAT 'R'.
*                                                          *
         TGET  (1),(0),R
*                                                          *
*                                                          *
         LTR   15,15          TEST RETURN CODE, IF NOT ZERO,
         BNZ   ERRTN          GO TO AN ERROR ROUTINE.
*                                                          *
*  PROCESSING
*                                                          *
ERRTN



*                                                          *
*  STORAGE DECLARATIONS
*                                                          *
BUFFER   DS    CL130          INPUT BUFFER
         END
```

Figure 77. Coding Example: TGET Macro Instruction Register Format

Using Terminal Control Macro Instructions

The following macro instructions allow a command processor to control terminal functions and attributes. (These macro instructions were formerly documented in IBM System/360 Operating System: Supervisor and Data Management Macro Instructions, GC28-6647.) They are listed, then described in detail.

Macro Instruction	Function
GTSIZE	Get Terminal Line Size
RTAUTOPT	Restart Automatic Line Numbering or Character Prompting
SPAUTOPT	Stop Automatic Line Numbering or Character Prompting
STATTN	Set Attention Simulation
STATUS	Change Subtask Status
STAUTOCP	Start Automatic Character Prompting
STAUTOLN	Set Automatic Line Numbering
STBREAK	Set Break
STCC	Specify Line-Deletion and Character-Deletion Characters
STCLEAR	Set Display Clear Character String
STCOM	Set Inter-Terminal Communication
STSIZE	Set Terminal Line Size
STTIMEOU	Set Timeout Feature
TCLEARQ	Clear Buffers

Some of the terminal control macro instructions may be safely coded in a user-written command processor. They are:

GTSIZE
RTAUTOPT
SPAUTOPT
STATUS
STAUTOCP
STAUTOLN
STSIZE
TCLEARQ

The other macro instructions, intended for system use, are not recommended for inclusion in user-written command processors. These macros are used in the IBM-supplied PROFILE and TERMINAL commands. Inappropriate use of the following macros can cause terminal errors:

STATTN
STBREAK
STCC
STCLEAR
STCOM
STTIMEOU

GTSIZE -- Get Terminal Line Size

Use the GTSIZE macro instruction to determine the current logical line size of the user's terminal. If the terminal is a display station, use the GTSIZE macro instruction to determine the size of the display screen.

When the GTSIZE macro instruction is issued in a time sharing environment, the logical line size of the user's terminal (that is, the maximum number of characters per line) is returned in register 1. If the terminal is a display station, the line size is returned in register 1 and the screen length (that is, the maximum number of lines per display) is returned in register 0. If the terminal is not a display station, register 0 will contain all zeros. The GTSIZE macro instruction is ignored if TSO is not active when the macro instruction is issued.

Figure 78 shows the format of the GTSIZE macro instruction.

[symbol]	GTSIZE

Figure 78. The GTSIZE Macro Instruction

When control is returned to the user, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Successful. The contents of registers 0 and 1 are described above.
04	Parameter(s) specified. No parameter(s) should be coded.

RTAUTOPT -- Restart Automatic Line Numbering or Character Prompting

Use the RTAUTOPT macro instruction to restart either the automatic line numbering feature or the automatic character prompting feature. (The feature was suspended when the terminal user caused an attention interruption or entered a null line of input.) Since only one of these features can be used at a time, the restarted feature is the one that was suspended. (See the STAUTOLN macro instruction for a description of the automatic line numbering feature and the STAUTOCP macro instruction for a description of the automatic character prompting feature.)

When this macro instruction is used to restart automatic line numbering, the first line number assigned after line numbering is restarted is the same line number that would have been assigned to the next line of terminal input if automatic line numbering had not been suspended.

If the application program is creating a line numbered data set, use of the STAUTOLN macro to specify the starting number is recommended when restarting automatic line numbering. This will insure that the application's numbers are still in synchronization with the system's.

The RTAUTOPT macro instruction is used only in a time sharing environment. It is ignored if TSO is not active when the macro instruction is issued.

Figure 79 shows the format of the RTAUTOPT macro instruction.

[symbol]	RTAUTOPT

Figure 79. The RTAUTOPT Macro Instruction

When control is returned to the user, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Successful. Either automatic line numbering or automatic character prompting has been restarted.
04	Parameter(s) specified. No parameter(s) should be coded.
08	Invalid request. Either automatic line numbering or automatic character prompting was never started or never suspended, or a SPAUTOPT macro instruction has been issued to stop automatic line numbering or automatic character prompting.

SPAUTOPT -- Stop Automatic Line Numbering or Character Prompting

Use the SPAUTOPT macro instruction to stop either the automatic line numbering feature or the automatic character prompting feature. Since only one of these features can be used at a time, the active feature is the feature that is stopped. (See the STAUTOLN macro instruction for a description of the automatic line numbering feature, and the STAUTOCP macro instruction for a description of the automatic character prompting feature.)

The system can suspend automatic prompting when the terminal user causes an attention interrupt or enters a null line of input. This macro should then be issued by the application program in its attention exit, or as the result of a zero length input line received via TGET. When stopped by the SPAUTOPT macro, prompting cannot be restarted by use of the RTAUTOPT macro. Prompting must be restarted by the STAUTOLN or STAUTOCP macro.

The SPAUTOPT macro instruction is used only in a time sharing environment. It is ignored if TSO is not active when the macro instruction is issued.

Figure 80 shows the format of the SPAUTOPT macro instruction.

[symbol]	SPAUTOPT	

Figure 80. The SPAUTOPT Macro Instruction

When control is returned to the user, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Successful. Either automatic line numbering or automatic character prompting has been stopped.
04	Parameter(s) specified. No parameter(s) should be coded.
08	Invalid request. Either automatic line numbering or automatic character prompting was never started.

STATTN -- Set Attention Simulation

Use the STATTN macro instruction to specify how a terminal user can interrupt the execution of his program without using an Attention key. The TERMINAL command issues the STATTN macro when the terminal user requests that simulated attention be set up.

When the STATTN macro instruction assigns a value to an operand, that value remains in effect until another STATTN macro instruction assigns a new value to the operand, or until the terminal user logs off. Issuing the STATTN macro instruction without specifying any operands results in a NOP instruction.

The STATTN macro instruction is used only in a time sharing environment. It is ignored if TSO is not active when the macro instruction is issued.

Figure 81 shows the format of the STATTN macro instruction. Each of the operands is explained following the figure. If an operand is not specified, its current status is not changed.

```
┌──────────┬─────────┬──────────────────────────────────────────────────┐
│          │         │  ┌       ┌integer┐┐┌        ┌integer┐┐           │
│ [symbol] │ STATTN  │  │LINES= ┤   0   ││││,TENS= ┤   0   ││           │
│          │         │  └       └       ┘┘└        └       ┘┘           │
│          │         │  ┌        ┌address┐┐                             │
│          │         │  │,INPUT= ┤   0   ││                             │
│          │         │  └        └       ┘┘                             │
└──────────┴─────────┴──────────────────────────────────────────────────┘
```

Figure 81. The STATTN Macro Instruction

LINES=
: indicates the output line count (if any) that determines when a terminal user can interrupt the execution of his program.

  integer
  : specifies an integer from 1 through 255. This integer indicates the number of consecutive lines of output that can be directed to the terminal before the keyboard will unlock to let the terminal user interrupt the execution of his program.

  0
  : indicates that output line count will not be used to determine when the terminal user can interrupt the execution of his program.

  If the LINES operand is coded for a display station, it is ignored. However, the display user may cause a simulated attention interruption at the bottom of the screen (i.e., after every 6, 12, or 15 lines of consecutive output, depending on screen size).

TENS=
: indicates whether or not locked keyboard time will be used to determine when a terminal user can interrupt the execution of his program.

  integer
  : specifies an integer from 1 through 255. This integer indicates the tens of seconds (that is, from 10 to 2550 seconds) of locked keyboard time that can elapse before the keyboard will unlock to let the terminal user interrupt the execution of his program.

  0
  : indicates that locked keyboard time will not be used to determine when the terminal user can interrupt the execution of his program.

INPUT=

indicates whether or not a character string will be used to determine when a terminal user can interrupt the execution of his program.

address

specifies the address of a character string from one to four EBCDIC characters long, left-justified and padded to the right with blanks if less than four characters long. When this character string is encountered as the only data in a line, input processing is interrupted to let the program take an attention exit. (See the description of the STAX macro instruction.) This string will not be recognized if it is preceded by any other character(s), including line delete or character delete control characters.

0

indicates that no character string will be used to determine when the terminal user can interrupt the execution of his program.

When control is returned to the user, register 15 will contain the following return code:

Hexadecimal Code	Meaning
00	Successful
04	Invalid request

STATUS -- Change Subtask Status

Use the STATUS macro instruction to change the dispatchability status of one or all of a program's subtasks. One use of the STATUS macro instruction is to restart subtasks that were stopped when an attention exit routine was entered. (See the description of the STAX macro instruction in "Attention Interruption Handling - the STAX Service Routine.")

The STATUS macro instruction is used in both time sharing and non-time sharing environments.

Figure 82 shows the format of the STATUS macro instruction. Each of the operands is explained following the figure.

```
┌──────────┬───────────┬─────────────────────────────────────────────────┐
│ [symbol] │ STATUS    │ ⎧START⎫[,TCB=subtask tcb address]               │
│          │           │ ⎩STOP ⎭                                         │
└──────────┴───────────┴─────────────────────────────────────────────────┘
```

Figure 82. The STATUS Macro Instruction

START

indicates that the STOP/START count in the task control block specified in the TCB operand will be decremented by 1. If the TCB operand is not coded, the STOP/START count is decremented by one in all the subtask control blocks of the originating task.

STOP

indicates that the STOP/START count in the task control block specified in the TCB operand will be incremented by 1. If the TCB operand is not coded, the STOP/START count is incremented by 1 in the task control blocks for all the subtasks of the originating task.

TCB=

is the address of a fullword on a fullword boundary that contains the address of the task control block that is to have its STOP/START count adjusted. If this operand is specified using register notation, the address of the task control block (not the address of the fullword) must have been previously loaded into the specified register. If this operand is not specified, the STOP/START count is adjusted in the task control blocks for all the subtasks of the originating task.

Control is returned to the instruction following the STATUS macro instruction. When control is returned, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Successful
04	The specified task control block does not belong to a subtask of the originating task. The STATUS macro instruction was ignored.

STAUTOCP -- Start Automatic Character Prompting

Use the STAUTOCP macro instruction to start automatic character prompting. Automatic character prompting signals the terminal user when the system is ready to accept input from the terminal. This signal consists of putting out at the terminal either an underscore and a backspace or a period and a carriage return, depending on the type of terminal being used. The STAUTOCP macro has no effect with a 2260 or 2265 display station, since the terminal user is always prompted for input by the "start-of-message" symbol.

This macro instruction can be used to have the system automatically prompt the user for input. It is used, for example, by the INPUT subcommand of the EDIT command.

Once started, automatic prompting is handled as follows: When the system has received a line of input, it immediately sends back to the terminal the next character prompt. If the program should send output while automatic prompting is in effect, the prompt will be repeated after all output has been set to the terminal. For example:

```
line of input
OUTPUT MSG FROM PROGRAM


_
```

Automatic prompting is designed to be used by a program operating in input mode (i.e., issuing successive TGET macros).

The system suspends automatic prompting when the terminal user causes an attention interruption or when he enters a null (nonprinting) line of input. The application program then takes appropriate action in an attention exit routine, or after receiving a zero length input via the TGET macro instruction. The application program can stop the prompting or line numbering function via SPAUTOPT, or restart the function via STAUTOCP.

The STAUTOCP macro instruction is used only in a time sharing environment. It is ignored if TSO is not active when the macro instruction is issued.

Figure 83 shows the format of the STAUTOCP macro instruction.

[symbol]	STAUTOCP	

Figure 83. The STAUTOCP Macro Instruction

When control is returned to the user, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Successful.
04	Parameter(s) specified. No parameter(s) should be coded.

STAUTOLN -- Start Automatic Line Numbering

Use the STAUTOLN macro instruction to start automatic line numbering. Automatic line numbering prints a line number at the beginning of each line.

This macro instruction can be used to have the system automatically prompt the user for input. It is used, for example, by the INPUT subcommand of the EDIT command.

Once started, automatic line numbering is handled as follows: When the system has received a line of input, it immediately sends back to the terminal the next line number. If the program should send output while automatic line numbering is in effect, the line number will be repeated after all output has been set to the terminal. For example:

```
00030 line of input
00040 OUTPUT MSG FROM PROGRAM
00040 _
```

Automatic line numbering is designed to be used by a program operating in input mode (i.e., issuing successive TGET macros).

The system prints a new line number for each line of input received. The current line number maintained by the system is decremented appropriately whenever the input queue is cleared by a TCLEARQ macro or as the result of an attention interruption. The application program is responsible for numbering the lines independently, if it is creating a line numbered data set. The system line number is not available to the application program.

The system suspends automatic line numbering when the terminal user causes an attention interruption or when he enters a null (nonprinting) line of input. The application program then takes appropriate action in an attention exit routine, or after receiving a zero length input via the TGET macro instruction. The application program can stop the line numbering function via SPAUTOPT, or restart the function via STAUTOLN or RTAUTOPT. You should use STAUTOLN rather than RTAUTOPT to restart automatic line numbering, if the application program is numbering the input lines it receives. This choice will insure that the program's numbers are still in synchronization with the system's numbers.

The STAUTOLN macro instruction is used only in a time sharing environment. It is ignored if TSO is not active when the macro instruction is issued.

Figure 84 shows the format of the STAUTOLN macro instruction. Each of the operands is explained following the figure.

[symbol]	STAUTOLN	S=address I=address

Figure 84. The STAUTOLN Macro Instruction

S=

indicates the address of a fullword that contains the number to be assigned to the first line of terminal input. This number can be any integer from 0 through 99,999,999.

I=

indicates the address of a fullword that contains the increment value to be used when assigning line numbers to lines of terminal input. This number can be any integer from 0 through 99,999,999.

When control is returned to the user, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Successful. A line number will be printed out at the beginning of each line of input.
04	Invalid parameter(s) specified.

STBREAK -- Set Break

Use the STBREAK macro instruction to indicate whether the transmit interrupt feature on an IBM 1050 terminal or on an IBM 2741 terminal will be used or suppressed. The transmit interrupt feature lets terminal output processing interrupt terminal input processing.

The TERMINAL command issues this macro when the terminal user specifies the BREAK or NOBREAK operand of the command.

The macro should be issued only when the terminal currently connected is a 1050 or a 2741 which has the transmit interrupt feature. Specifying STBREAK YES for a 1050 or 2741 without the transmit interrupt feature could result in loss of output or permanent error at the terminal.

When the transmit interrupt feature is being used by the system, the terminal user can "type ahead" of his program, entering the next line while the previous one is being processed. All 33/35 teletypes are handled this way. 1050's and 2741's that have been defined in the TSO-TCAM Message Control Program as having the transmit interrupt feature will be handled this way (unless STBREAK NO is specified).

Terminal handling when the feature is in use is as follows. If no output is available for the terminal, and if there are sufficient TSO terminal buffers available, the keyboard will be unlocked to allow the user to enter input. If the user's program generates output (TPUT) before he has started to enter data, the read operation is halted and the break (transmit interrupt) feature can be used to lock the keyboard and condition the communications line to transmit output. If the user has already started to type when the TPUT is issued, the output will not be sent until he has finished that line of input. If, however, the TPUT had specified the BREAKIN option, the output message would interrupt any input in progress. If the application does not issue a TCLEARQ macro to flush the input buffer queue, the interrupted input will be printed out again after the output is sent, to let the user continue to type from the point where he had been interrupted.

When the transmit interrupt feature is not being used by the system, the terminal keyboard is unlocked only after the user's program has issued a TGET request for input. In this mode of operation, the terminal user cannot type ahead of his program. A TPUT with the BREAKIN option cannot interrupt input. The output will not be sent until the terminal user has completed entering his current input line. All 2260 and 2265 display stations are handled in this way. All 1050's and 2741's which have been defined in the TSO-TCAM Message Control Program as not having the transmit interrupt feature will be handled this way.

The STBREAK macro instruction is used only in a time sharing environment. It is ignored if TSO is not active when the macro instruction is issued.

Figure 85 shows the format of the STBREAK macro instruction.

[symbol]	STBREAK	[<u>YES</u> NO]

Figure 85. The STBREAK Macro Instruction

YES
indicates that the transmit interrupt feature will be used. If neither YES nor NO is specified, YES is assumed.

NO
indicates that the transmit interrupt feature not be used.

When control is returned to the user, register 15 will contain one of the following return codes:

Hexadecimal Code	Meaning
00	Successful.
04	Invalid parameter.
08	Invalid terminal type. This macro instruction should be issued only for the IBM 1050 terminal or the IBM 2741 terminal.

STCC -- Specify Terminal Control Characters

Use the STCC macro instruction to specify what control characters will be used to delete a character or a line of terminal input.

The PROFILE command issues this macro when a terminal user requests a new line or character deletion character. The PROFILE command also causes the newly defined characters to be included in the user's profile in the User Attribute Data Set (UADS). Each time the user logs on, the Terminal Monitor Program will issue the STCC macro, specifying the characters in the UADS at the start of the session. If the terminal user does not use the PROFILE command to change the line or character-deletion characters, the system-supplied defaults are always used, as described below.

When the line-delete control character specified in the STCC macro instruction is encountered within a line of terminal input, the line control character and all the preceding characters in that line are deleted. When the character-delete control character specified in the STCC macro instruction is encountered within a line of terminal input, the character delete control character and the character immediately preceding it are deleted from the line.

When the user is logging on, he can delete a line or character by using the system-supplied defaults. The defaults, according to type of terminal, are as follows:

Type of Terminal	Desired Action	Key(s) to be Pressed
2741 and 1050	line deletion or character deletion	Attention key and backspace
33/35 Teletype[1]	line deletion or character deletion	CTRL and X key (hex '18'), back arrow (←), or underscore (_), depending on keyboard. (Either key results in hex '6D'.)

No defaults are defined for the 2260 or 2265 display stations, because the terminal user can use cursor control keys more effectively to delete characters or lines before the input is transmitted to the system.

The STCC macro instruction is used only in a time sharing environment. It is ignored if TSO is not active when the macro instruction is issued.

Figure 86 shows the format of the STCC macro instruction; each of the operands is explained following the figure.

[symbol]	STCC	[ATTN NATN] [,LD={X'n' C'c'}] [,CD={X'n' C'c'}]

Figure 86. The STCC Macro Instruction

ATTN

When this operand is in effect, hitting the Attention key after having typed data will only delete the current line. System response is !D. Automatic prompting is not turned off. The Attention key can then be hit again, without typing any input, to interrupt the program and turn off prompting. When this operand is not in effect, the Attention key will both delete a line of terminal input and interrupt the execution of the user's program. System response is !. or !I.

NATN

indicates that the Attention key will not be used to delete a line of terminal input.

LD=

indicates what character will be used for the line delete control character. (Do not specify both LD= and ATTN.)

X'n', where n is the hexadecimal representation of any EBCDIC character on the terminal keyboard, except the new line (NL) and carriage return (CR) control characters. If X'00' is specified, the previously used line-delete control character is retained. If X'FF' is specified, no character will be used for the line-delete control character. If a character that does not appear on the terminal keyboard is specified, that character is rejected and no character is used to delete a line of terminal input.

C'c' where c is the character representation of any EBCDIC character on the terminal keyboard.

[1] Trademark of the Teletype Corporation.

CD=

indicates what character will be used for the character delete control character.

X'n' where n is the hexadecimal representation of any EBCDIC character on the terminal keyboard except the new line (NL) and carriage return (CR) control characters. If X'00' is specified, the previously used character delete control character is retained. If X'FF' is specified, no character will be used for the character delete control character. If a character that does not appear on the terminal keyboard is specified, that character is rejected and no character is used to delete a character from a line of terminal input.

C'c' where c is the character representation of any EBCDIC character on the terminal keyboard.

When control is returned to the user, the low-order byte of register 0 contains the former line delete control character. If X'FF' appears in the low-order byte of register 0, there is no former line delete control character. If X'80' appears in the high-order byte of register 0, ATTN has been specified for line deletion.

The low-order byte of register 1 contains the former character delete control character. If X'FF' appears in the low-order byte of register 1, there is no former character delete control character.

Register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Successful.
04	Invalid parameters specified.
08	Invalid request. Specified character does not appear on the terminal keyboard or ATTN was specified for a terminal that does not have an attention key.

STCLEAR -- Set Display Clear Character String

Use the STCLEAR macro instruction to specify the character string that will be used to request that a 2260 or 2265 display station screen be erased. The TERMINAL command issues this macro when the user specifies the character string he wants.

The STCLEAR macro instruction is used only in a time sharing environment. It is ignored if TSO is not active when the macro instruction is issued.

Figure 87 shows the format of the STCLEAR macro instruction. Each of the operands is explained following the figure.

[symbol]	STCLEAR	STRING=$\begin{Bmatrix}\text{address}\\ 0\end{Bmatrix}$

Figure 87. The STCLEAR Macro Instruction

STRING=
indicates the address of a one-to four character string that will be used to request that the display station screen be erased. This character string must be left-justified and padded on the right with blanks, if necessary. If 0 is specified, no character string will be used to erase the screen.

When control is returned to the user, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Successful.
04	Invalid parameter.
08	Invalid terminal type. The terminal is not a display station.

STCOM -- Set Inter-Terminal Communication

Use the STCOM macro instruction to specify whether or not a terminal will accept messages from other terminals, or low priority messages from the system operator. High priority operator messages are always sent to the terminal. The PROFILE command issues this macro when the user specifies the INTERCOM or NOINTERCOM operand of the command.

The STCOM macro instruction is used only in a time sharing environment. It is ignored if TSO is not active when the macro instruction is issued.

Figure 88 shows the format of the STCOM macro instruction.

[symbol]	STCOM	YES NO

Figure 88. The STCOM Macro Instruction

YES
indicates that the terminal will accept messages from other terminals. If neither YES nor NO is specified, YES is assumed.

NO
indicates that the terminal will not accept messages from other terminals.

When control is returned to the user, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Successful.
04	Invalid parameter specified.

STSIZE -- Set Terminal Line Size

Use the STSIZE macro instruction to set the logical line size of the time sharing terminal. If the terminal is a display station, the STSIZE macro instruction is used to set the screen size.

The TERMINAL command issues this macro instruction when the user specifies the LINESIZE or SCREEN operands of the command.

The STSIZE macro instruction is used only in a time sharing environment. It is ignored if TSO is not active when the macro instruction is issued.

Figure 89 shows the format of the STSIZE¯macro instruction each of the operands is explained following the figure.

```
┌───────────┬──────────┬──────────────────────────────────────────────────────┐
│           │          │ ⎧SIZE=number     ⎫⎡,LINE=number    ⎤                │
│ [symbol]  │ STSIZE   │ ⎩SIZELOC=address ⎭⎣,LINELOC=address⎦                │
└───────────┴──────────┴──────────────────────────────────────────────────────┘
```

Figure 89. The STSIZE Macro Instruction

SIZE
specify the logical line size of the terminal in characters. If the logical line size requested is greater than the mechanical line size of the terminal, the last character in the line may be repeatedly typed over. Specifying a size greater than 255 will give unpredictable results.

SIZELOC
specify the address of a word containing the logical line size of the terminal in characters.

LINE
specify the number of lines that can appear on the screen of a display station terminal.

LINELOC
specify the address of a word containing the number of lines that can appear on the screen of a display station terminal.

Note: If the terminal is a display station, either the LINE or LINELOC operand must be specified. If the terminal is not a display station, neither operand should be specified.

Defaults by terminal type are as follows:

Terminal Type	Line Size, Number of Lines, or Screen Size
2741	120
1050	120
33/35 Teletype[1]	72
2260,2265	12x80, 12x40, 6x40, 15x64 - as specified by the installation in the TSO-TCAM Message Control Program.

[1] Trademark of the Teletype Corporation.

When control is returned to the user, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Successful.
04	Invalid parameter specified.
08	Invalid LINE, LINELOC, SIZE, or SIZELOC operand, as follows: 1. The LINE or LINELOC operand was specified for any terminal except a display station. (An operand value of zero is not an error, and has the same effect as omitting the operand.) 2. The LINE or LINELOC operand was omitted, or specified as zero, for a display station. 3. The SIZE or SIZELOC operand was omitted, or specified as zero, for any terminal type.
0C	The dimensions specified for a display station do not correspond to known existing screen size. Incorrect screen management can result.

STTIMEOU -- Set Timeout Feature

Use the STTIMEOU macro instruction to specify whether the 1050 terminal has the optional text timeout suppression feature. The macro instruction allows 1050's, with or without the feature, to call in via the same switched line, with any 1050 being handled initially as if it did not have the feature.

A 1050 without the text timeout suppression feature operates as follows: When the PROCEED light is on and the keyboard is unlocked, the terminal will "timeout," that is, the keyboard will lock if the user does not type input for approximately 20 seconds. The system subsequently responds to the timeout by restoring the keyboard so that the user may continue. The user can prevent the timeout by periodically pressing the SHIFT key.

A 1050 with the text timeout suppression feature operates as follows: The keyboard does not lock if the user does not type input within 20 seconds. The system can therefore use the Read Inhibit channel command, which does not timeout within 28 seconds, in contrast to the Read channel command that does timeout. (Note: If the system is directed to use the Read Inhibit channel command for a 1050 that does timeout, the terminal may be locked out of the system.)

Until the STTIMEOU macro instruction is issued, 1050 terminals are handled as per the definition provided in the TSO TCAM Message Control Program. If the currently connected terminal has the text timeout suppression feature, STTIMEOU NO can be issued to direct the system to use Read Inhibit rather than Read channel commands. (STTIMEOU NO should not be issued for a 1050 that does not have the text timeout suppression feature. This specification could cause the terminal to be locked out of the system.)

The TERMINAL command processor issues the STTIMEOU macro instruction when the user specifies the TIMEOUT or NOTIMEOUT operand of the TERMINAL command. The STTIMEOU macro instruction will remain in effect until the user logs off.

The STTIMEOU macro instruction should be issued only when an IBM 1050 terminal is being used. Terminals which are equivalent to the one explicitly supported may also function satisfactorily. The customer is responsible for establishing equivalency. IBM assumes no responsibility for the impact that any changes to the IBM-supplied products or programs may have on such terminals.

The STTIMEOU macro instruction is used only in a time sharing environment. It is ignored if TSO is not active when the macro instruction is issued.

Figure 90 shows the format of the STTIMEOU macro instruction.

[symbol]	STTIMEOU	<u>YES</u> NO

Figure 90. The STTIMEOU Macro Instruction

YES

indicates that IBM 1050 terminal does timeout. It does not have the text timeout suppression feature. If the operand is omitted, the default is YES.

NO

indicates that the IBM 1050 terminal does not timeout. The 1050 does have the text timeout suppression feature.

When control is returned to the user, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Successful.
04	Invalid parameter specified.
08	Invalid terminal type. This macro instruction applies to the IBM 1050 terminal only.

TCLEARQ -- Clear Buffers

TCLEARQ enables the user to throw away "typed ahead" input or unsent output. This clearing of the buffers lets the command processor resynchronize with the terminal user.

For example, when a command processor analyzes the specified operands in a line of input and discovers missing or invalid parameters, it issues a TCLEARQ INPUT before sending a prompting message to the user. This insures that the command processor will receive a line of input entered after the terminal user has seen the prompting message.

When the TCLEARQ macro instruction is issued to clear the input buffers, all the input that has been entered at the terminal but has not yet been processed by the foreground job is purged. To ensure synchronization, the terminal keyboard is locked until the next TGET macro is issued.

When the TCLEARQ macro instruction is issued to clear the output buffers, all the output that has been processed by the foreground job but not yet printed out at the terminal is purged.

The TCLEARQ macro instruction is used only in a time sharing environment. It is ignored if TSO is not active when the macro instruction is issued.

The TCLEARQ macro instruction is written as follows:

Figure 91 shows the format of the TCLEARQ macro instruction; each of the operands is described following the figure.

[symbol]	TCLEARQ	[INPUT OUTPUT]

Figure 91. The TCLEARQ Macro Instruction

INPUT
indicates that all input currently in the terminal's input buffer queue will be lost, including the input line currently being entered, if any. If neither INPUT nor OUTPUT is specified, INPUT is assumed.

OUTPUT
indicates that all the output for this terminal that is currently in the terminal's output buffer queue will be purged, except for output messages that have begun to appear at the terminal, or messages from other terminals or the system operator. (Such messages are sent via the TPUT TJID macro instruction.)

When control is returned to the user, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Successful
04	Invalid parameter(s) specified

Command SCAN and PARSE -- Determining the Validity of Commands

If you write your own command processors to run under TSO, you will need a method of determining whether any command name or subcommand name entering the system is valid, and whether the operands following the command are syntactically correct. Command Scan and Parse are two service routines provided within TSO, which perform those functions.

Command Scan scans the command buffer for commands. Parse scans the command buffer for operands. In general, Command Scan is invoked by a Terminal Monitor Program and Parse is invoked by a command processor. Command Scan may also be invoked by the TEST Program or by any command processors that process subcommands.

Both of these service routines are linked to; their entry points are:

Service Routine	Entry Point
Command Scan	IKJSCAN
Parse	IKJPARS

Sequence of Operations

If you use Command Scan and Parse within a TMP or Command Processor, the sequence of operations is as follows:

1. Your Terminal Monitor Program or Command Processor gets a line of input which may contain a command and its parameters.

2. Your Terminal Monitor Program or Command Processor, links to Command Scan (IKJSCAN) and passes it a parameter list containing, among other things, the address of the command buffer.

3. Command Scan scans the buffer for a command name, syntax checks the command name if you request it, updates the command buffer offset field to point to the command operands (if any), and returns control to the calling program.

4. The calling program receives the address of the command name and gives control to the appropriate command processor or subcommand processor.

5. The command processor links to Parse (IKJPARS) and passes it parameter lists containing, among other things, the syntactical structure of the command operands, and the address of the buffer.

6. Parse scans the buffer for operands, builds a list describing the operands found, and returns control to the calling program.

7. The command processor processes the command according to the operands received.

8. When the command processor terminates, it returns control to the Terminal Monitor Program and the sequence is repeated.

This section discusses:

- Using the Command Scan Service Routine.
- Using the Parse Service Routine.

Using the Command Scan Service Routine (IKJSCAN)

Command Scan scans the command buffer for commanas. In general, Command Scan is linked to by a Terminal Monitor Program, but it may also be invoked by the TEST program or by any command processors that process subcommands.

Command scan scans a command within the command buffer and performs the following functions:

1. It translates all lower case characters within the command name to upper case.
2. It resets the offset pointer in the command buffer to point to the first non-blank character in the operand field, if a valid operand is present. If a valid operand is not present, the offset pointer points to the end of the buffer.
3. It returns a pointer to the command name, the length of the command name, and a code explaining the results of its scan to the calling routine.
4. It optionally, at your request, syntax checks the command name.

This topic discusses:

- Command Name Syntax
- The Parameter List Structure Required by Command Scan.
- The Command Scan Parameter List.
- Flags Passed to Command Scan.
- The Command Scan Output Area.
- The Operation of the Command Scan Service Routine.
- The Results of the Command Scan.
- Return Codes from Command Scan.

COMMAND NAME SYNTAX

If you write your own command processor, and you intend to use the Command Scan Service Routine to check for a valid command name, your name must meet the following syntax requirements:

- The first character must be an alphabetic or a national character.
- The remaining characters must be alphameric.
- The length of the command name must not exceed eight characters.
- The command delimiter must be a separator character.
- The name should include one or more numerals. Since no IBM-Supplied Command Names include numerals, your command name will be unique.

THE PARAMETER LIST STRUCTURE REQUIRED BY COMMAND SCAN

Before you LINK to the Command Scan service routine, you must create the parameter structure shown in Figure 92. You then place the address of the Command Scan Parameter List (CSPL) into general register 1, set the flags in the Flag word, and link to IKJSCAN, the Command Scan service routine.

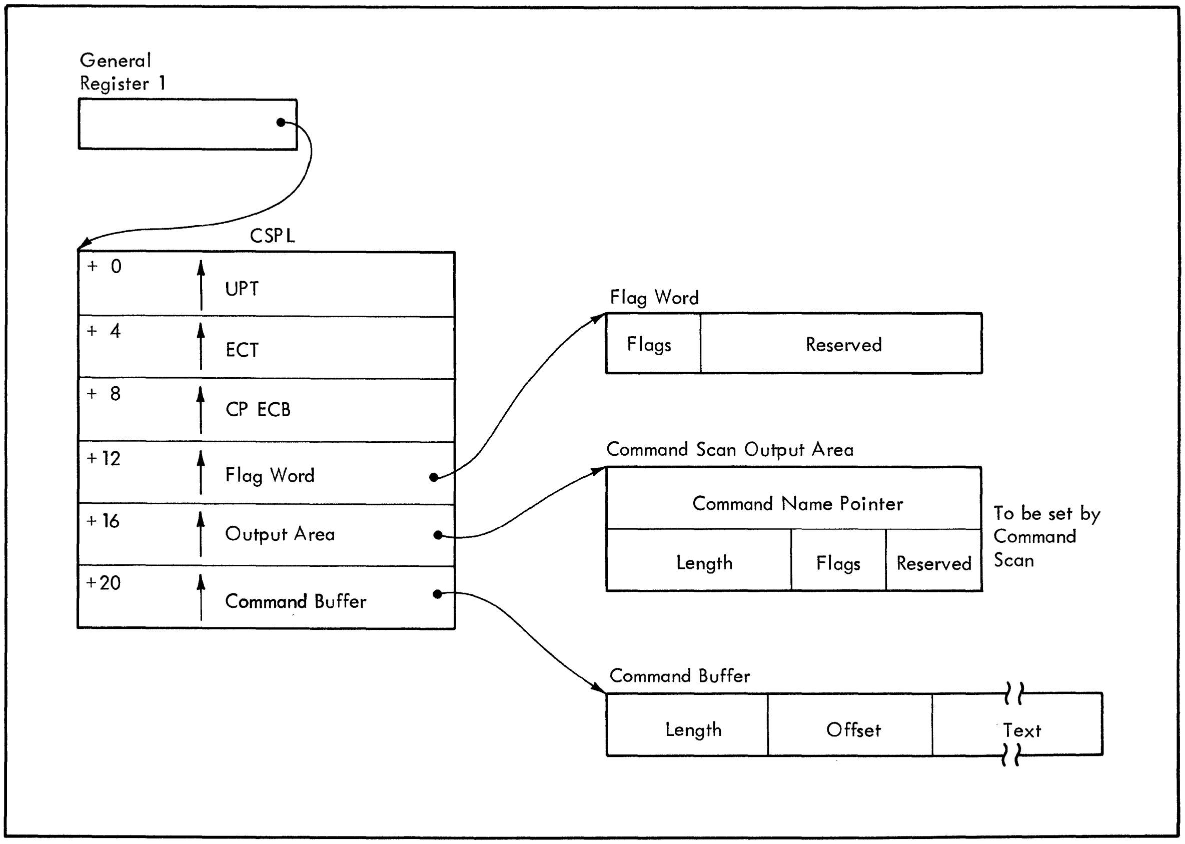


Figure 92. The Parameter List Structure Passed to Command Scan

The Command Scan Parameter List

The Command Scan Parameter List (CSPL) is a six-word parameter list containing addresses required by the Command Scan routine. In order to ensure the reenterability of the calling program, the CSPL should be built in subpool 1 in an area obtained by the calling program with the GETMAIN macro instruction.

The CSPL is defined by the IKJCSPL DSECT. Figure 93 shows the format of the Command Scan Parameter List.

Number of Bytes	Field	Contents or Meaning
4	CSPLUPT	The address of the User Profile Table. (See Appendix A.)
4	CSPLECT	The address of the Environment Control Table. (See Appendix A.)
4	CSPLECB	The address of the Command Processor's Event Control Block. (Required if Command Scan is called by a command processor to scan a subcommand; zeros if Command Scan is called by the TMP.)
4	CSPLFLG	The address of a fullword, obtained via the GETMAIN macro instruction by the routine linking to Command Scan, and located in subpool 1. The first byte of the word pointed to contains flags set by the calling routine; the last three bytes are reserved.
4	CSPLOA	The address of an 8-byte Command Scan Output Area, located in subpool 1. The output area is obtained by the calling routine via a GETMAIN macro instruction. It is filled by the Command Scan service routine before it returns control to the calling routine. (See Figure 92.)
4	CSPLCBUF	The address of the Command buffer.

Figure 93. The Command Scan Parameter List

Flags Passed to Command Scan

The flag word built in subpool 1 and pointed to by the fourth word of the CSPL, is obtained and freed by the calling routine. Only the first byte of the field is used by the Command Scan service routine; the remaining three bytes are reserved. Set the flag byte before linking to the Command Scan routine to indicate whether or not you want the command to be syntax checked. The flag byte has the following meanings:

X'00'	Syntax Check the command name.
X'80'	Do not syntax check the command name.

The Command Scan Output Area

The Command Scan Service routine returns the results of its scan to the calling program by filling in a two word Command Scan Output Area (CSOA). The CSOA must be obtained and freed by the calling program. It must be located in subpool 1 and its address stored into the fifth word of the Command Scan Parameter List before your program links to IKJSCAN.

The CSOA is defined by the IKJCSOA DSECT. Figure 94 shows the format of the Command Scan Output Area.

Number of Bytes	Field	Contents or Meaning
4	CSOACNM	The address of the command name if the command name is present and valid. Zero otherwise.
2	CSOALNM	Length of the command name if the command name is present and valid. Zero otherwise.
1	CSOAFLG	A flag field. Command Scan sets these flags to indicate the results of its scan. See Figure 94 'Return from Command Scan - CSOA and Buffer Setting'.
1		Reserved.

Figure 94. The Command Scan Output Area

THE OPERATION OF THE COMMAND SCAN SERVICE ROUTINE

If you set the flags field in the flag word to X'80' -- do not syntax check the command name -- the command scan service routine merely scans the buffer to determine if it contains a question mark or a command. The first character in the command buffer is checked for a question mark whether or not syntax checking is requested. If it is a question mark, no further scanning is done. If it is not a question mark, the command name is considered to begin at the first non-separator character found, and end at the first command delimiter character found (See Figure 81).

Command Scan translates any lower case letters in the command name to upper case, fills the Command Scan Output Area, updates the command buffer offset field, and returns to the calling program.

If you have requested syntax checking (X'00' in the flag field of the flag word), the command name must meet the syntax requirements, as follows:

- The first character must be an alphabetic or a national character.
- The remaining characters must be alphameric.
- The length of the command name must not exceed eight characters.
- The command delimiter must be a separator character.

Figure 95 shows the various character types recognized by Command Scan.

CHARACTER		CHARACTER TYPE						
		Separator	National	Alphabetic	Numeric	Command Delimiter	Delimiter	Special
Horizontal Tab	HT	x				x		
Blank	ƀ	x				x		
Comma	,	x				x		
Dollar Sign	$		x					
Number Sign	#		x					
At Sign	@		x					
	a - z			x				
	A - z			x				
	0 - 9				x			
New line	NL					x	x	
Period	.					x		x
Left parenthesis	(					x		x
Right parenthesis	)					x		x
Ampersand	&					x		x
Asterisk	*							x
Semicolon	;					x	x	
Minus sign, hyphen	-					x		x
Slash	/					x	x	
Apostrophe	'					x		x
Equal sign	=					x		x
Cent sign	¢		x					x
Less than	<							x
Greater than	>							x
Plus sign	+							x
Logical OR	|							x
Exclamation point	!		x					x
Logical NOT	¬							x
Percent sign	%							x
Dash	-							x
Question mark	?							x
Colon	:							x
Quotation Mark	"		x					x

Figure 95. Character Types Recognized by Command Scan and Parse

RESULTS OF THE COMMAND SCAN

The Command Scan service routine scans the command buffer and returns the results of its scan to the calling routine by filling the Command Scan Output Area, and by updating the offset field in the command buffer. Figure 96 shows the possible CSOA settings and command buffer offset settings upon return from the Command Scan service routine.

Command Scan Output Area			Command Buffer
Flag	Meaning	Length Field	Offset set to:
X'80'	The command name is valid and the remainder of the buffer contains non-separator characters.	Length of command name	The first non-separator following the command name.
X'40'	The command name is valid and there are no non-separator characters remaining.	Length of command name.	The end of the buffer.
X'20'	The command name is a question mark.	Zero	Unchanged.
X'10'	The buffer is empty or contains only separators.	Zero	The end of the buffer.
X'08'	The command name is syntactically invalid.	Zero	Unchanged.

Figure 96. Return from Command Scan - CSOA and Command Buffer Settings

RETURN CODES FROM COMMAND SCAN

The Command Scan service routine returns the following codes in general register 15 to the program that invoked it:

CODE (hex)	Meaning
0	Command Scan completed successfully.
4	Command Scan was passed invalid parameters.

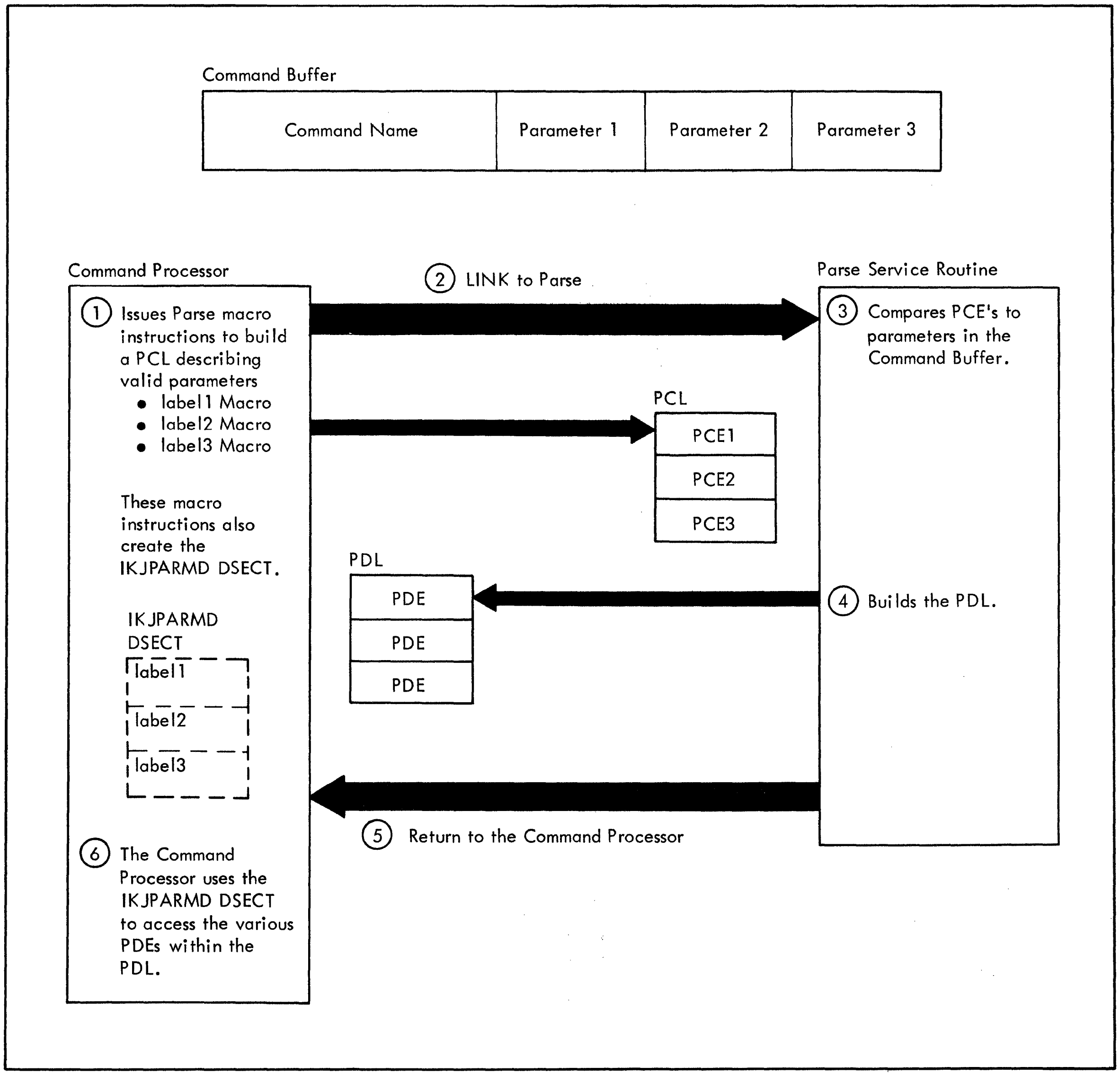
Using the Parse Service Routine (IKJPARS)

The Parse service routine checks the syntax of command operands. To prepare for this, the command processor creates a Parameter Control List (PCL) --- a description of permissible operands, default values, text to be used when prompting, and, if present, the address of a validity checking subroutine.

The command processor links to Parse, which scans and checks each operand against the entries (called PCEs: Parameter Control Entries) in the PCL. In turn, Parse builds and returns results of the scan to the command processor in a Parameter Descriptor List (PDL), whose entries (called PDEs: Parameter Descriptor Entries) contain pointers to data set names, indications of specified options, or pointers to the subfields entered with the command operands.

The command processor uses the IKJPARMD DSECT to refer to the PDL. The command processor specifies the IKJPARMD DSECT at the time it issues the PARSE macro instructions to build the PCL. The labels used by the command processor on the various Parse macro instructions become the symbolic addresses of the fields in the IKJPARMD DSECT.

Figure 97 depicts a command processor's use of the Parse macro instructions, the Parse service routine, and the IKJPARMD DSECT.



Command Buffer
Command Name
Parameter 1
Parameter 2
Parameter 3
Command Processor
① Issues Parse macro instructions to build a PCL describing valid parameters
● label1 Macro
● label2 Macro
● label3 Macro
These macro instructions also create the IKJPARMD DSECT.
IKJPARMD DSECT
label1
label2
label3
⑥ The Command Processor uses the IKJPARMD DSECT to access the various PDEs within the PDL.
② LINK to Parse
PCL
PCE1
PCE2
PCE3
PDL
PDE
PDE
PDE
⑤ Return to the Command Processor
Parse Service Routine
③ Compares PCE's to parameters in the Command Buffer.
④ Builds the PDL.

Figure 97. A Command Processor Using the Parse Service Routine

Parse service routine support consists of the following:

1. The following set of macro instructions:

   IKJPARM Begins the Parameter Control List and establishes a symbolic reference for the Parameter Descriptor List.

   IKJPOSIT Builds a Parameter Control Entry. This PCE describes a positional parameter that contains delimiters recognized by the Parse Service routine; but not including the positional parameters described by the IKJTERM, IKJOPER, or IKJRSVWD macro instructions.

   IKJIDENT Also builds a Parameter Control Entry; however, this PCE describes a positional parameter that does not depend upon a particular delimiter.

IKJKEYWD Builds a Parameter Control Entry that describes a Keyword parameter.

IKJNAME Describes the possible names that may be entered for a keyword or reserved word parameter.

IKJTERM Builds a Parameter Control Entry. This PCE describes a positional parameter that may be a constant, statement number, or variable.

IKJOPER Builds a Parameter Control Entry that describes an expression. An expression consists of three parts; two operands and an operator in the form:

```
    (operand1  operator  operand2)
```

IKJRSVWD Builds a Parameter Control Entry. This PCE may be used with the IKJTERM macro instruction to describe a reserved word constant, with the IKJOPER macro instruction to describe the operator of an expression, or by itself to describe a reserved word parameter.

IKJSUBF Indicates the beginning of a keyword subfield description.

IKJENDP Indicates the end of the PCL.

IKJRLSA Releases any storage (allocated by the Parse service routine) that remains after Parse has returned control to the command processor.

2. A program that checks the syntax of the command operands within the command buffer against the PCL and builds a PDL containing the results of the syntax check.

Parse also provides the following services which may be selected by the calling routine:

- It translates the command operands to upper case.
- It substitutes default values for missing operands.
- It prompts the user at the terminal for missing positional parameters.
- It passes control to an exit, supplied by the calling routine, to do further checking on a positional parameter.
- It inserts implied keywords.
- It appends user-supplied second-level messages to prompting messages.

This section describes:

- Command Parameter Syntax
- Using the Parse Macro Instructions to Define Command Syntax
- The Parse Macro Instructions
- Passing Control to the Parse Service Routine
- Formats of the PDEs Returned by Parse
- Additional Facilities Provided by Parse
- An Example of Using the Parse Service Routine
- Return Codes from the Parse Service Routine

COMMAND PARAMETER SYNTAX

If you write your own command processors, and you intend to use the Parse service routine to determine which operands have been entered following the command name, your command parameters must adhere to the syntactical structure described in this section.

Command parameters must be separated from one another by one or more of the separator characters: blank, tabulation, or comma (See Figure 95). The command parameters end either at the end of a logical line (carriage return), or at a semicolon. If the command parameters end with a semicolon, and other characters are entered after the semicolon but before the end of the logical line, Parse ignores that portion of the line that follows the semicolon. Parse returns no message to indicate this condition.

There are two types of command parameters recognized by the Parse service routine: (1) Positional parameters, or (2) Keyword parameters.

Positional Parameters

Positional parameters must be coded first in the parameter string, and they must be in a specific order.

In general, the Parse service routine considers a positional parameter to be missing, if the first character of the parameter scanned is not the character expected. For instance, if a parameter is supposed to begin with a numeric character and Parse finds an alphabetic character in that position, the numeric parameter is considered missing. Parse then prompts for the missing parameter if it is required, substitutes a default value if one is available, or ignores the missing parameter if the parameter is optional.

For the purpose of syntax checking, positional parameters are divided into parameters that include delimiters as part of their definition (delimiter-dependent parameters), and parameters that do not include delimiters as part of their definition (non-delimiter-dependent parameters).

DELIMITER-DEPENDENT PARAMETERS: Those parameters that include delimiters as part of their definition are called delimiter-dependent parameters. The Parse service routine recognizes the delimiter-dependent parameter syntaxes as shown in Figure 98.

PARAMETER	Macro Instruction Used to Describe Parameter
1. DELIMITER	IKJPOSIT
2. STRING	
3. VALUE	
4. ADDRESS	
5. PSTRING	
6. USERID	
7. DSNAME	
8. DSTHING	
9. QSTRING	
10. SPACE	
11. CONSTANT	IKJTERM
12. VARIABLE	
13. STATEMENT NUMBER	
14. EXPRESSION	IKJOPER
15. RESERVED WORD	IKJRSVWD

Figure 98. Delimiter-Dependent Parameters

1. DELIMITER - It may be any character other than an asterisk, left parenthesis, right parenthesis, semicolon, blank, comma, tab, carriage return, or digit. A self-defining delimiter character is represented in this discussion by the symbol Δ . The delimiter parameter is used only in conjunction with the string parameter.

2. STRING - A string is the group of characters between two alike self-defining delimiter characters, such as

   ΔstringΔ

   or, the group of characters between a self-defining delimiter character and the end of a logical line, such as

   Δstring

   The same self-defining delimiter character can be used to delimit two contiguous strings, such as

   ΔstringΔstringΔ

   or

   ΔstringΔstring

A null string, which indicates that a positional parameter has not been entered, is defined as two contiguous delimiters or a delimiter and the end of the logical line. If the missing string is a required parameter, the null string must be entered as two contiguous delimiters. Note that a string received from a prompt or a default must not include the delimiters.

3. VALUE - A value consists of a character followed by a string enclosed in apostrophes, such as

   X'string'

   The character must be an alphabetic or national character. The string may be of any length and may consist of any combination of enterable characters. If the ending apostrophe is left off the string, Parse assumes that the string ends at the end of the logical line. If Parse encounteres two successive apostrophes, it assumes them to be part of the string and continues to scan for a single ending apostrophe. The Parse service routine always raises the character preceding the first apostrophe to upper case. The value is considered missing if the first character is not an alphabetic or national character, or if the second character is not an apostrophe.

4. ADDRESS - There are several forms of the address parameter.

   Absolute address - An absolute address consists of from one to six hexadecimal digits followed by a period.

   Relative address - A relative address consists of from one to six hexadecimal digits preceded by a plus sign.

General register address - A general register address consists of a decimal integer in the range 0 to 15 followed by the letter R. R can be entered in either upper or lower case.

Floating-point register address - A floating-point register address consists of an even decimal integer in the range 0 to 6 followed by the letter D (for double precision) or E (for single precision). The letter E or D can be entered in either upper or lower case.

Symbolic address - A symbolic address consists of any combination, up to 31 characters in length, of the alphameric characters and the break character. The first character must be either an alphabetic or a national character.

Qualified address - A qualified address has the following format:

```
loadname.entryname.symbolic address
                         or
                  .relative address
```

- loadname - any combination of alphameric characters up to eight characters in length, of which the first character is an alphabetic or a national character.
- entryname - has the same syntax as a loadname, but it must be preceded by a period as illustrated in the example.
- symbolic address - as defined above, but must be preceded by a period as illustrated in the example.
- relative address - as defined above, but must be preceded by a period as illustrated in the example.

Indirect address - An indirect address is an absolute, relative, symbolic, or general register address followed by from 1 to 255 percent signs, such as:

```
+A%
```

The number of percent signs following the address indicate the number of levels of indirect addressing. In this example (+A%), the data is pointed to by the location pointed to by +A.

Address expression - An address expression has the following format:

```
addr[%...]±expression value[%...][±expression value[%...]]
```

addr - represents an absolute, relative, symbolic, or general register address. If a general register address is used, then it must have indirect address notation, that is, it must be followed by at least one percent sign.

expression value - consists of from one to six hexadecimal digits or one to six decimal digits followed by the letter N. The N can be in either upper or lower case. The expression values can be indirect. There is no limit to the number of expression values in the address expression.

Note: Blanks are not allowed within any form of the address parameter.

5. PSTRING - A parenthesized string is a string of characters enclosed within a set of parentheses, such as:

   (string)

   The string may consists of any combination of characters of any length, with one restriction; if it includes parentheses, they must be balanced. The enclosing right parenthesis of a PSTRING can be omitted if the string ends at the end of a logical line.

   A null PSTRING is defined as a left parenthesis followed by either a right parenthesis or the end of a logical line.

6. USERID - A userid consists of an identification optionally followed by a slash and a password. The format is:

   identification [/password]

   identification - can be any combination of alphameric characters up to seven characters in length, the first of which must be an alphabetic or national character.

   password - can be any combination of alphameric characters up to eight characters in length, the first of which must be an alphabetic or national character.

   Blanks may be inserted between the identification and the slash, and between the slash and the password.

   If just the identification is entered, Parse does not prompt for the password. If the identification is entered followed by a slash and no password, Parse prompts for the password by executing a PUTGET macro instruction specifying bypass mode, that is, the terminal user's reply will not print at the terminal. The terminal user can reply to a prompt for password by entering either a password or a null line. If the user enters a null line, PARSE builds the PDE and leaves the password field blank.

7. DSNAME - The data set name parameter has three possible formats:

   dsname [(membername)] [/password]
   [dsname] (membername) [/password]
   'dsname [(membername)] ' [/password]

   dsname - may be either a qualified or an unqualified name.

   An unqualified name is any combination of alphameric characters up to eight characters in length, the first character of which must be an alphabetic or national character.
   A qualified name is made up of several unqualified names, each unqualified name separated by a period. A qualified name, including the periods, may be up to 44 characters in length.

   membername - one to eight alphameric characters, the first of which must be an alphabetic or a national character.

Note: PARSE considers the entire DSNAME parameter missing if the first character scanned is not an apostrophe, an alphabetic character, a national character, or a left parenthesis.

If the slash and the password are not entered, Parse does not prompt for the password. If the slash is entered and not the password, Parse prompts for the password by executing a PUTGET macro instruction specifying bypass mode, i.e., the terminal user's reply will not print at the terminal.

8. DSTHING - A DSTHING is a dsname parameter as previously defined except that an asterisk can be substituted for an unqualified name or for each qualifier of a qualified name. PARSE processes the asterisk as if it were a DSNAME. The asterisk is used to indicate that all data sets at that particular level are considered.

9. QSTRING - A quoted string is a string of characters enclosed within apostrophes, such as:

   'string'

   The string can consist of any length combination of characters, with one restriction: if the user wishes to enter apostrpohes within the string, two successive apostrophes must be entered for each single apostrophe desired; one of the apostrophes is removed during the parse.

   The ending apostrophe is not required if the string ends at the end of the logical line.

   A null quoted string is defined as two contiguous apostrophes or an apostrophe at the end of the logical line.

10. SPACE - Space is a special purpose parameter; it allows a string parameter that directly follows a command name to be entered without a preceding self-defining delimiter character. The space parameter must always be followed by a string parameter. If the delimiter of the command name is a tab, the tab is the first character of the string. The string always ends at the end of the logical line.

11. CONSTANT - There are several forms of the constant parameter.

    Fixed-point numeric literal - Consists of a string of digits (0 through 9) preceded optionally by a sign (+ or -), such as:

    +1234.43

    This literal may contain a decimal point anywhere in the string except as the rightmost character. The total number of digits cannot exceed 18. Embedded blanks are not allowed.

    Floating-point numeric literal - Takes the following form:

    +1234.56E+10

    This literal is a string of digits (0 through 9) preceded optionally by a sign (+ or -) and must contain a decimal point. This is immediately followed by the letter E and then a string of digits (0 through 9) preceded optionally by a sign (+ or -). Embedded blanks are not allowed. The string of digits preceding the letter E cannot be greater than 16 and the string following E cannot be greater than 2.

Non-numeric literal - Consists of a string of characters from the EBCDIC character set excluding the apostrophe and enclosed in apostrophes such as:

```
'Numbers (1234567890) and letters are OK'
```

The length of the string excluding apostrophes may be from 1 to 120 characters in length.

Figurative constant - Is one of a set of reserved words supplied by the caller of the Parse routine such as:

```
test123
```

A figurative constant consists of a string of characters up to 255 in length. Embedded blanks are not allowed. All characters of the EBCDIC character set are allowed except the blank, comma, tab, semicolon, and carriage return.

12. VARIABLE - The following is the form of the variable parameter.

```
[program-id.]data-name ┌{OF} qualification┐
                       │{IN}              │
                       └(subscript)       ┘
```

Data-name - consists of a maximum of 30 characters of the set:

A through Z (Alphabetic)
0 through 9 (Numeric)
- (hyphen)

such as:

```
My-dataset-123
```

The data-name cannot begin or end with a hyphen and must contain at least one alphabetic character.

Program-id - Consists of the first eight characters of a program identifier followed by a period. The first character must be alphabetic (A through Z) and the remaining characters must be alphameric (A through Z or 0 through 9) such as:

```
Here55.My-dataset
```

Qualification - Is applied by placing after a data-name one or more data-name(s) preceded by the qualifiers IN or OF, such as:

```
My-dataset-123 OF Your-dataset-456
```

The number of qualifiers that can be entered for a data-name is limited to 255.

Subscript - Consists of a data-name with subscripts enclosed in parentheses following the data-name such as:

```
Your-dataset-456 (My-dataset-123)
```

A separator between the data-name and the subscript is optional. Subscripts are a list of constants or variables.

The number of subscripts that can be entered for a data-name is limited to 3, such as:

```
Here55 (ABC def H15)
```

A separator character between subscripts is required.

13. STATEMENT NUMBER - The following is the form of a statement number.

    [program id.]line number[.verb number]

    An example is:

    ```
    Here.23.7
    ```

    Where:

    Program id - consists of the first eight characters of a program identifier followed by a period. The first character must be alphabetic (A through Z) and the remaining characters must be alphameric (A through Z or 0 through 9).

    Line number - consists of a string of digits (0 through 9) and that cannot exceed a length of 6 digits.

    Verb number - consists of one digit (0 through 9) that is preceded by a period.

    Embedded blanks are not allowed in a statement number.

14. EXPRESSION - An expression takes the form:

    (operand1 operator operand2)

    The operator in the expression shows a relationship between the operands, such as:

    ```
    (ABC  equals 123)
    ```

    An expression must be enclosed in parentheses. An expression is defined by the IKJOPER macro. The operands are defined by the IKJTERM macro, and the operator by the IKJRSVWD macro instruction.

15. RESERVED WORD - Has three uses depending on the presence or absence of operands on the IKJRSVWD macro instruction. The uses are:

    - When used with the RSVWD keyword of the IKJTERM macro instruction, the IKJRSVWD macro identifies the beginning of a list of reserved words anyone of which can be entered as a constant.

    - When used with the RSVWD keyword of the IKJOPER macro instruction, the IKJRSVWD macro identifies the beginning of a list of reserved words anyone of which can be an operator in an expression.

    - When used by itself, the IKJRSVWD macro instruction defines a positional reserved word parameter.

    Note: The IKJRSVWD macro instruction is followed by a list of IKJNAME macros that contain all of the possible reserved words used as figurative constants or operators.

POSITIONAL PARAMETERS NOT DEPENDENT ON DELIMITERS: A positional parameter that is not dependent on delimiters is parsed as a character string with restrictions on the beginning character, additional characters, and length. These restrictions are passed to the Parse service routine as operands on the IKJIDENT macro instruction.

The Parse service routine recognizes the following character types as the beginning character and additional characters of a non-delimiter-dependent positional parameter:

ALPHA - Indicates an alphabetic or national character.
NUMERIC - Indicates a number, (0-9).
ALPHANUM - Indicates an alphabetic or national character or a number.
ANY - Indicates that the character to be expected can be any character other than a blank, comma, tab, semicolon, or carriage return. Right parenthesis must, however, be balanced by left parenthesis.

An asterisk can be entered in place of any positional parameter that is not dependent on delimiters.

ENTERING POSITIONAL PARAMETERS AS LISTS OR RANGES: You may want to have some positional parameters of your command entered in the form of a list, a range, or a list of ranges. The macro instructions that describe positional parameters to the Parse service routine, IKJPOSIT, IKJTERM and IKJIDENT, provide a LIST and a RANGE operand. If coded in the macro instruction, they indicate that the positional parameters expected can be in the form of a list or a range.

LIST

Indicates to the Parse service routine that one or more of the same type of positional parameters may be entered enclosed in parentheses as follows:

```
(positional-parameter positional-parameter ...)
```

If one or more of the items contained in the list are to be entered enclosed in parentheses, both the left and the right parenthesis must be included for each of those items.
The following positional parameter types may be used in the form of a list:

- VALUE
- ADDRESS
- USERID
- DSNAME
- DSTHING
- CONSTANT
- STATEMENT NUMBER
- VARIABLE
- Any positional parameter that are not dependent upon delimiters.

RANGE

Indicates to the Parse service routine that two positional parameters are to be entered separated by a colon as follows:

```
positional-parameter:positional-parameter
```

The following positional parameter types may be used in the form of a range or a list of ranges:

- ADDRESS
- VALUE
- CONSTANT
- STATEMENT NUMBER
- VARIABLE
- Any positional parameter that is not dependent upon delimiters.

If the user at the terminal wants to enter a parameter that begins with a left parentheses, and you have specified in either the IKJPOSIT or IKJIDENT macro instruction that the parameter can be entered as a list or a range, the user must enclose the parameter in an extra set of parentheses to obtain the correct result.

For instance, if you have specified via the IKJPOSIT macro instruction that the DSNAME operand may be entered as a list, and the terminal user wishes to enter a dsname of the form:

```
(membername)/password
```

He must enter it as:

```
((membername)/password)
```

Keyword Parameters

Keyword parameters can be entered anywhere in the command as long as they follow all positional parameters. They may consists of any combination of alphameric characters up to 31 characters long, the first of which must be an alphabetic character.

You describe keyword parameters to the Parse service routine with the IKJKEYWD, IKJNAME and IKJSUBF macro instructions.

Keyword parameters can have other parameters associated with them. These parameters, known as subfields, must be enclosed in parentheses directly following the keyword. A subfield may contain positional as well as keyword parameters. In the following example posn1 and kywd2 are parameters in the subfield of keyword 1:

```
keyword1(posn1 kywd2)
```

The same syntax rules that apply to commands, apply within keyword subfields.

- Keyword parameters must follow positional parameters.

- Enclosing right parenthesis may be eliminated if the subfield ends at the end of a logical line.

- The subfield may not contain unbalanced right parentheses.

If a keyword, with a subfield in which there is a required parameter, is entered without the subfield, Parse prompts for the required parameter. The terminal user must not include the subfield parentheses when he enters the required parameter.

If a subfield has a positional parameter, that can be entered as a list, and if this is the only parameter in the subfield, the list must be enclosed by the same parentheses that enclose the subfield, such as

```
keyword(item1 item2 item3)
```

where item1, item2, and item3 are members of a list.

If a subfield has, as its first parameter, a positional parameter that may be entered as a list, and there are additional parameters in the subfield, a separate set of parentheses is required to enclose the list, such as

```
keyword((item1 item2 item3) param)
```

where item1, item2, and item3 are members of a list, and param is a parameter not included in the list.

USING THE PARSE MACRO INSTRUCTIONS TO DEFINE COMMAND SYNTAX

A Command Processor using the Parse service routine must build a Parameter Control List (PCL) defining the syntax of acceptable command parameters. Each acceptable command parameter is described by a Parameter Control Entry (PCE) within the PCL. The Parse service routine compares the command parameters within the command buffer against the PCL to determine if valid command parameters have been entered.

Parse returns the results of this comparison to the Command Processor in a Parameter Descriptor List (PDL). The PDL is composed of separate entries (PDEs) for each of the Command Parameters found in the command buffer.

The Command Processor builds the PCL and the PCEs within it using the Parse macro instructions. These macro instructions generate the PCL and establish symbolic references for the PDL returned by the Parse service routine.

There are eleven Parse macro instructions. They are:

- IKJPARM
- IKJPOSIT
- IKJTERM
- IKJOPER
- IKJRSVWD
- IKJIDENT
- IKJKEYWD
- IKJNAME
- IKJSUBF
- IKJENDP
- IKJRLSA

These macro instruction perform the following functions:

1. The IKJPARM macro instructions begins the PCL CSECT and the PDL DSECT, and provides symbolic addresses for both.

2. The IKJPOSIT, IKJTERM, IKJOPER, IKJRSVWD, IKJIDENT, IKJKEYWD, IKJNAME, and IKJSUBF macro instructions describe the positional and keyword parameters valid for the command processor. These macro instructions expand into the PCEs required by the Parse service routine during its scan of the command buffer. The label fields of these macro instructions are used as labels within the DSECT that maps the PDL returned by the Parse service routine.

3. The IKJENDP macro instruction ends the PCL CSECT.

4. The IKJRLSA macro instruction releases the storage obtained by the Parse service routine for the PDL.

IKJPARM - Beginning the PCL and the PDL

Code the IKJPARM macro instruction to begin the Parameter Control List and to provide a symbolic address for the beginning of the Parameter Descriptor List returned by the Parse service routine. The PCL is constructed in a CSECT named by the label field of the macro instruction; the PDL will be mapped by the DSECT named in the DSECT operand of the macro instruction.

Figure 99 shows the format of the IKJPARM macro instruction. Each of the operands is explained following the figure. Appendix B describes the notation used to define macro instructions.

label	IKJPARM	DSECT= {dsectname
		IKJPARMD}

Figure 99. The IKJPARM Macro Instruction

label
    The name you provide is used as the name of the CSECT in which the PCL is constructed.

DSECT=
    Provides a name for the DSECT created to map the Parameter Descriptor List. This may be any name; the default is IKJPARMD.

THE PARAMETER CONTROL ENTRY BUILT BY IKJPARM: The IKJPARM macro instruction generates the Parameter Control Entry (PCE) shown in Figure 100. This PCE begins the Parameter Control List.

Number of Bytes	Field Name	Contents or Meaning
2		Length of the Parameter Control List. This field contains a hexadecimal number representing the number of bytes in this PCL.
2		Length of the Parameter Descriptor List. This field contains a hexadecimal number representing the number of bytes in the Parameter Descriptor List returned by the Parse service routine.
2		This field contains a hexadecimal number representing the offset within the PCL to the first IKJKEYWD PCE or to an end-of-field indicator if there are no keywords. An end-of-field indicator may be an IKJSUBF or an IKJENDP PCE.

Figure 100. The Parameter Control Entry Built by IKJPARM

IKJPOSIT - Describing a Delimiter-Dependent Positional Parameter

Code the IKJPOSIT macro instruction to describe delimiter-dependent positional parameters.

The order in which you code the macros for positional parameters is the order in which the Parse service routine expects to find the positional parameters in the command string.

Figure 101 shows the format of the IKJPOSIT macro instruction. Each of the operands is explained following the figure. Appendix B describes the notation used to define macro instructions.

```
label      IKJPOSIT   { ,SPACE     }
                      { ,DELIMITER }
                      { ,STRING    }
                      { ,VALUE     }
                      { ,ADDRESS   }              [,LIST][,RANGE]
                      { ,PSTRING   }
                      { ,USERID    }
                      { ,DSNAME    }
                      { ,DSTHING   }
                      { ,QSTRING   }

                       [,SQSTRING]

                      [,UPPERCASE] [,PROMPT='prompt data'   ]
                      [,ASIS     ] [,DEFAULT='default value']

                      [,HELP=('help data','help data',...)]

                      [,VALIDCK=symbolic-address]
```

Figure 101. The IKJPOSIT Macro Instruction

label
: This name is used as the symbolic address within the PDL DSECT of the Parameter Descriptor Entry for the parameter described by this IKJPOSIT macro instruction.

SPACE DELIMITER STRING VALUE ADDRESS PSTRING USERID DSNAME DSTHING QSTRING	These are the positional parameter types recognized by the Parse service routine. A syntactic definition of each is contained under the heading, "Delimiter-Dependent Parameters".

SQSTRING

The command operand is processed either as a string or as a quoted string. If the delimiter is an apostrophe, the command operand is processed as a quoted string. If the delimiter is any of the other acceptable delimiter characters, the command operand is processed as a string. The SQSTRING option can only be specified if STRING is specified for the parameter type. As an example, if SQSTRING is coded in the IKJPOSIT macro instruction, a terminal user entering a command could specify either

```
/string/string...  or 'string' 'string' ...
```

for this positional parameter.

LIST
The command operands may be entered by the terminal user as a list, that is, in the form:

```
Command Name (parameter,parameter, ...)
```

This list option may be used with the following delimiter-dependent positional parameters:

USERID, DSNAME, DSTHING, ADDRESS, and VALUE.

RANGE
The command operands may be entered by the terminal user as a range, that is, in the form:

```
Command Name parameter:parameter
```

This range option may be used with the following delimiter-dependent positional parameters:

ADDRESS and VALUE.

Note: The following options (UPPERCASE, ASIS, PROMPT, DEFAULT, HELP, and VALIDCK) may be used with all delimiter-dependent positional parameters except SPACE and DELIMITER.

UPPERCASE
The parameter is to be translated to uppercase.

ASIS
The parameter is to be left as it was entered by the terminal user.

PROMPT='prompt data'
The parameter described by this IKJPOSIT macro instruction is required; the prompting data is the message to be issued if the parameter is not entered by the terminal user. If prompting is necessary and the terminal is in prompt mode, Parse adds a message-identifying number (message ID) and the word "ENTER" to the beginning of this message before writing it to the terminal.
If prompting is necessary but the terminal is in no-prompt mode, Parse adds a message ID and the word "MISSING" to the beginning of this message before writing it to the terminal.

DEFAULT='default value'
The parameter described by this IKJPOSIT macro instruction is required, but the terminal user need not enter it. If the parameter is not entered, the value specified as the default value is used.

Note: If neither PROMPT nor DEFAULT is specified, the parameter is optional. The Parse service routine takes no action if the parameter specified by this IKJPOSIT macro instruction is not present in the command buffer.

HELP=('help data','help data'...)
You can provide up to 255 second-level messages. Enclose each message in apostrophes and separate the messages by single commas. These messages are issued one at a time after each question mark entered by the terminal user in response to a prompting message from the Parse service routine. These messages are not sent to the user when the prompt is for a password on a DSNAME or USERID parameter.
Parse adds a message ID and the word "ENTER" (in prompt mode) or "MISSING" (in no-prompt mode) to the beginning of each message before writing it to the terminal.

VALIDCK=symbolic-address

Supply the symbolic address of a validity checking subroutine if you want to perform additional validity checking on this parameter. Parse calls this routine after first determining that the parameter is syntactically correct.

<u>THE PARAMETER CONTROL ENTRY BUILT BY IKJPOSIT</u>: The IKJPOSIT macro instruction generates the variable length Parameter Control Entry (PCE) shown in Figure 102.

Number of Bytes	Field	Contents or Meaning
2		Flags. These flags are set to indicate which options were coded in the IKJPOSIT macro instruction.
	Byte 1	
	001.	This is an IKJPOSIT PCE.
	...1	PROMPT
	.... 1...	DEFAULT
	.... .0..	Reserved
	.... ..1.	HELP
	1	VALIDCK
	Byte 2	
	1...	LIST
	.1..	ASIS
	..1.	RANGE
	.... 1...	SQSTRING
	...0 .000	Reserved
2		Length of the Parameter Control Entry. This field contains a hexadecimal number representing the number of bytes in this IKJPOSIT PCE.
2		Contains a hexadecimal offset from the beginning of the Parameter Descriptor List to the related Parameter Descriptor Entry built by the PARSE service routine.
1		This field contains a hexadecimal number indicating the type of positional parameter described by this PCE. These numbers have the following meaning:
	<u>HEX</u>	
	1	DELIMITER
	2	STRING
	3	VALUE
	4	ADDRESS
	5	PSTRING
	6	USERID
	7	DSNAME
	8	DSTHING
	9	QSTRING
	A	SPACE
	B to FF	Not used.

Figure 102. The Parameter Control Entry Built by IKJPOSIT (Part 1 of 2)

Number of Bytes	Field	Contents or Meaning
1		Contains the length minus one of the default or prompting information supplied on the IKJPOSIT macro instruction. This field and the next are present only if DEFAULT or PROMPT were specified on the IKJPOSIT macro instruction.
VARIABLE		This field contains the prompting or default information supplied on the IKJPOSIT macro instruction.
2		This field contains a hexadecimal figure representing the length in bytes of all the PCE fields used for second-level messages. The figure includes the length of this field. The fields are present only if HELP is specified on the IKJPOSIT macro instruction.
1		This field contains a hexadecimal number representing the number of second-level messages specified by HELP on this IKJPOSIT PCE.
2		This field contains a hexadecimal number representing the length of this HELP segment. The length figure includes the length of this field, the message segment offset field, and the length of the information. These fields are repeated for each second-level message specified by HELP on the IKJPOSIT macro instruction.
2		This field contains the message segment offset. It is set to X'0000'.
Variable		This field contains one second-level message supplied on the IKJPOSIT macro instruction specified by HELP. This field and the two preceding ones are repeated for each second-level message supplied on the IKJPOSIT macro instruction. These fields do not appear if second-level message data was not supplied.
3		The address of a validity checking routine. This field is present only if a validity checking address was included in the IKJPOSIT macro instruction.

Figure 102. The Parameter Control Entry Built by IKJPOSIT (Part 2 of 2)

IKJTERM - Describing a Delimiter-Dependent Positional Parameter

Code the IKJTERM macro instruction to describe a positional parameter that is one of the following:

Statement Number
Constant
Variable
Constant or Variable

The order in which you code the macros for positional parameters is the order in which the Parse service routine expects to find the parameters in the command string.

Figure 103 shows the format of the IKJTERM macro instruction. Each of the operands is explained following the figure. Appendix B describes the notation used to define macro instructions.

```
label      IKJTERM     'parameter-type'[,LIST][,RANGE]

                       [,UPPERCASE] [       {STMT}]
                       [,ASIS     ] [,TYPE= {CNST}]
                                    [       {VAR }]
                                    [       {ANY }]

                       [,SBSCRPT[=label-PCE]] [,PROMPT='prompt data'   ]
                                              [,DEFAULT='default value']

                       [,HELP=('help data','help data',...)]

                       [,VALIDCK=symbolic-address][,RSVWD=label-PCE]
```

Figure 103. The IKJTERM Macro Instruction

label
This name is used as the symbolic address within the PDL DSECT of the Parameter Descriptor Entry for the parameter described by this IKJTERM macro instruction.

Note: The label is not used when the IKJTERM macro instruction is describing a subscript of a data-name.

'parameter-type'
This field is required so that the parameter can be identified when an error message is necessary. This field differs from the PROMPT field in that the PROMPT field is not required and if supplied is used only for a required parameter that is not entered by the terminal user. Blanks within the apostrophes are allowed.

LIST
The command operands may be entered by the terminal user as a list, in the form:

```
Command Name  (parameter,parameter,...)
```

The LIST option may be used with any of the TYPE= positional parameters.

RANGE
The command operands may be entered by the terminal user as a range, in the form:

```
Command Name  parameter:parameter
```

The range option may be used with any of the TYPE= positional parameters.

Note: The LIST and RANGE options can not be used when the IKJTERM macro instruction is describing a subscript of a data-name.

UPPERCASE
The parameter is to be translated to uppercase.

ASIS
The parameter is to be left as it was entered by the terminal user.

TYPE=
Describes the type of the parameter as one of:

• STMT	Statement Number
• CNST	Constant
• VAR	Variable
• Any	Constant or Variable

Note: A syntactical definition of these parameters is contained under the heading "Delimiter-Dependent Parameters".

SBSCRPT[=label-PCE]
Specifies one of two conditions:

1. If SBSCRPT is entered with a label-PCE then the data-name described by the IKJTERM macro may be subscripted. Supply the name of the label of an IKJTERM macro instruction that describes the subscript. Only TYPE=VAR or TYPE=ANY parameters can be subscripted.
2. If SBSCRPT is entered without a label-PCE then the IKJTERM macro is describing the subscript of a data-name. All TYPE= parameters may be used on a subscript except TYPE=STMT. The LIST and RANGE options can not be used on an IKJTERM macro that is describing a subscript.

Note: Two IKJTERM macros are coded to describe a subscripted data-name. The first IKJTERM macro describes the data-name and specifies the SBSCRPT option with the label of the second IKJTERM macro. The second IKJTERM macro describes the subscript of the data-name and specifies SBSCRPT without a label-PCE. The second macro must immediately follow the first.

PROMPT='prompt data'
The parameter described by this IKJTERM macro instruction is required. The prompting data is the message to be issued if the parameter is not entered by the terminal user. If prompting is necessary and the terminal is in prompt mode, Parse adds a message-identifying number (message ID) and the word "ENTER" to the beginning of the message before writing it to the terminal.

If prompting is necessary but the terminal is in no-prompt mode, Parse adds a message ID and the word "MISSING" to the beginning of the message before writing it to the terminal. If a subscripted data-name requires prompting, the terminal user is prompted for the entire name including the subscript.

DEFAULT='default value'
The parameter described by this IKJTERM macro instruction is required, but the terminal user need not enter it. If the parameter is not entered, the value specified as the default value is used.

Note: If neither PROMPT nor DEFAULT is specified, the parameter is optional. The Parse service routine takes no action if the parameter is not present.

HELP=('help data','help data',...)
You can provide up to 255 second-level messages. Enclose each message in apostrophes and separate the messages by single commas. These messages are issued one at a time after each question mark entered by the terminal user in response to a prompting message from the Parse routine.
Parse adds a message ID and the word "ENTER" (in prompt mode) or "MISSING" (in no-prompt mode) to the beginning of each message before writing it to the terminal.

VALIDCK=symbolic-address
Supply the symbolic address of a validity checking subroutine if you want to perform additional checking on this parameter. Parse calls this routine after first determining that the parameter is syntactically correct.

RSVWD=label-PCE
This parameter is used when TYPE=CNST or TYPE=ANY is specified. This option indicates that this parameter can be a figurative constant. Supply the address of the PCE (label on a IKJRSVWD macro instruction) that begins the list of reserved words that can be entered as a figurative constant.
This list of reserved words is defined by a series of IKJNAME macros that contain all possible names and immediately follow the IKJRSVWD macro.

Note: The IKJRSVWD macro can be coded anywhere in the list of macros that build the PCL except following an IKJSUBF macro instruction. This permits other IKJTERM macro instructions to refer to the same list.

THE PARAMETER CONTROL ENTRY BUILT BY IKJTERM: The IKJTERM macro instruction generates the variable Parameter Control Entry (PCE) shown in Figure 104.

Number of Bytes	Field	Contents or Meaning
2		Flags. These flags are set to indicate options on the IKJTERM macro instruction.
	Byte 1	
	110.	This is an IKJTERM PCE.
	...1	PROMPT
	.... 1...	DEFAULT
	.... .0..	Reserved
	.... ..1.	HELP
	1	VALIDCK
	Byte 2	
	1...	LIST
	.1..	ASIS
	..1.	RANGE
	...1	This term may be SUBSCRIPTED.
	.... 1...	A reserved word PCE is chained from this term.
	.... .000	Reserved
2		The hexadecimal length of this PCE.
2		Contains a hexadecimal offset from the beginning of the Parameter Descriptor List to the Parameter Descriptor Entry built by the Parse routine.

Figure 104. The Parameter Control Entry Built by IKJTERM (Part 1 of 2)

Number of Bytes	Field	Contents or Meaning
1		This field indicates the type of positional parameter described by this PCE.
	1...	STATEMENT NUMBER
	.1..	VARIABLE
	..1.	CONSTANT
	...1	ANY (Constant or Variable)
	.... 1...	This term is a SUBSCRIPT term.
	.... .000	Reserved
4	Byte 1-2	Contains the hexadecimal length of the parameter-type field.
	Byte 3-4	Contains the offset of the parameter-type field. It is set to X'0012'.
Variable		Contains the parameter-type field.
1		Contains the length of the default or prompting information supplied on the macro instruction.
Variable		Contains the default or prompting information supplied on the macro instruction.
2		If a subscript is specified on the macro, this field contains the offset into the Parameter Control List of the subscript PCE.
2		If a reserved word PCE is specified on the macro, this field contains the offset into the Parameter Control List of the reserved word PCE.
2		Contains the length (including this field) of all the PCE fields used for second-level messages if HELP is specified on the macro.
1		The number of second-level messages specified on the macro instruction by the HELP parameter.
2		Contains the length of this segment including this field, the message offset field and second-level message. Note: This field and the following two are repeated for each second-level message specified by HELP on the macro.
2		This field contains the message segment offset.
Variable		This field contains one second-level message specified by HELP on the macro instruction. This field and the two preceding fields are repeated for each second-level message specified.
3		Contains the address of the validity checking routine if it is spacified on the macro with the VALIDCK keyword.

Figure 104. The Parameter Control Entry Built by IKJTERM (Part 2 of 2)

IKJOPER - Describing a Delimiter-Dependent Positional Parameter

Code the IKJOPER macro instruction to provide a Parameter Control Entry (PCE) that describes an expression. An expression consists of three parts; two operands and one operator in the form:

(operand1 operator operand2)

such as: (ABC eq 123)

The parts of an expression are described by PCEs that are chained to the IKJOPER PCE. The IKJTERM macro instruction is used to identify the operands, and the IKJRSVWD macro instruction is used to identify the operator.

Figure 105 shows the format of the IKJOPER macro instruction. Each of the operands is explained following the figure. Appendix B describes the notation used to define macro instructions.

```
 label     IKJOPER    'parameter-type'[,PROMPT='prompt data'    ]
                                      [,DEFAULT='default value' ]

                      [,HELP=('help data','help data',...)]

                      [,VALIDCK=symbolic-address],OPERND1=label1

                      ,OPERND2=label2,RSVWD=label3

                      [,CHAIN=label4]
```

Figure 105. The IKJOPER Macro Instruction

label
: This name is used as the symbolic address within the PDL DSECT of the Parameter Descriptor Entry for the parameter described by this IKJOPER macro instruction.

'parameter-type'
: This field is required so that the parameter can be identified when an error message is necessary. This field differs from the PROMPT field in that the PROMPT field is not required and if supplied is used only for a required parameter that is not entered by the terminal user. Blanks within the apostrophes are allowed.

Note: This field is used only with error messages for the complete expression. The IKJTERM and IKJRSVWD PCEs are used with an error message for missing operands or operator. If a validity check routine specifies an invalid expression, then the entire expression is prompted for.

PROMPT='prompt data'
: The parameter described by this IKJOPER macro instruction is required. The prompting data is the message to be issued if the parameter is not entered by the terminal user. If prompting is necessary and the terminal is in prompt mode, Parse adds a message-identifying number (message ID) and the word "ENTER" to the beginning of the message before writing it to the terminal.
If prompting is necessary but the terminal is in no-prompt mode, Parse adds a message ID and the word "MISSING" to the beginning of the message before writing it to the terminal.

DEFAULT='default value'
The parameter described by this IKJOPER macro instruction is required, but the terminal user need not enter it. If the parameter is not entered the value specified as the default value is used.

Note: If neither PROMPT nor DEFAULT is specified, the parameter is optional. The Parse service routine takes no action if the parameter is not present.

HELP=('help data','help data',...)
You can provide up to 255 second-level messages. Enclose each message in apostrophes and separate the messages by single commas. These messages are issued one at a time after each question mark entered by the terminal user in response to a prompting message from the Parse routine.
Parse adds a message ID and the word "ENTER" (in prompt mode) or "MISSING" (in no-prompt mode) to the beginning of each message before writing it to the terminal.

VALIDCK=symbolic-address
Supply the symbolic address of a validity checking subroutine if you want to perform additional checking on this expression. Parse calls this routine after first determining that the expression is syntactically correct.

OPERND1=label1
Supply the name of the label field of the IKJTERM macro instruction that is used to describle the first operand in the expression. This IKJTERM macro instruction should be coded immediately following the IKJOPER macro instruction that describes the expression.

OPERND2=label2
Supply the name of the label field of the IKJTERM macro instruction that is used to describe the second operand in the expression. This IKJTERM macro instruction should be coded immediately following the IKJNAME macro instructions that describe the operator in the expression under the associated IKJRSVWD macro instruction.

RSVWD=label3
Supply the name of the label field of the IKJRSVWD macro instruction that begins the list of reserved words that are used to describe the possible operators to be entered for the expression. The IKJRSVWD and associated IKJNAME macro instructions should be coded immediately following the IKJTERM macro that describes the first operand, and immediately preceding the IKJTERM macro that describes the second operand.

CHAIN=label4
Indicates that this parameter described by the IKJOPER macro instruction may be entered as an expression or as a variable. Supply the name of the label field of an IKJTERM macro instruction that describes the variable term. The LIST and RANGE options are not permitted on this IKJTERM macro instruction. Code this IKJTERM macro instruction immediately following the IKJTERM macro that describes the second operand.

Note: The Parse routine first determines if the parameter is entered as an expression. If the parameter is an expression, that is, enclosed in parentheses, then it is processed as an expression. If it is not an expression, then it is processed using the chained IKJTERM PCE to control the scan of the parameter.

THE PARAMETER CONTROL ENTRY BUILT BY IKJOPER: The IKJOPER macro instruction generates the variable Parameter Control Entry (PCE) shown in Figure 106.

Number of Bytes	Field	Contents or Meaning
2		Flags. These flags are set to indicate options on the IKJOPER macro instruction.
	Byte 1	
	111.	This is an IKJOPER PCE.
	...1	PROMPT
	.... 1...	DEFAULT
	.... .0..	Reserved
	.... ..1.	HELP
	1	VALIDCK
	Byte 2	
	0000 0000	Reserved
2		The hexadecimal length of this PCE.
2		Contains a hexadecimal offset from the beginning of the Parameter Descriptor List to the Parameter Descriptor Entry built by the Parse routine.
4	Byte 1-2	Contains the hexadecimal length of the parameter-type field.
	Byte 3-4	Contains the offset of the parameter-type field (X'0012').
Variable		Contains the parameter-type field.
2		If a reserved word PCE is specified on the macro, this field contains the offset into the Parameter Control List of the reserved word PCE.
2		Contains the offset into the Parameter Control List of the OPERND1 PCE.
2		Contains the offset into the Parameter Control List of the OPERND2 PCE.
2		Contains the offset into the Parameter Control List of the chained term PCE if present. Zero if not present.
1		Contains the length of the default or prompting information supplied on the macro instruction.
Variable		Contains the default or prompting information supplied on the macro instruction.
2		Contains the length (including this field) of all the PCE fields used for second-level messages if HELP is specified on the macro.

Figure 106. The Parameter Control Entry Built by IKJOPER (Part 1 of 2)

Number of Bytes	Field	Contents or Meaning
1		The number of second-level messages specified on the macro instruction by the HELP= parameter.
2		Contains the length of this segment including this field, the message offset field and second-level message. Note: This field and the following two are repeated for each second-level message specified by HELP on the macro.
2		This field contains the message segment offset.
Variable		This field contains one second-level message specified by HELP on the macro instruction. This field and the two preceding fields are repeated for each second-level message specified.
3		Contains the address of the validity checking routine if it is specified on the macro with the VALIDCK keyword.

Figure 106. The Parameter Control Entry Built by IKJOPER (Part 2 of 2)

IKJRSVWD - Describing a Delimiter-Dependent Positional Parameter

Code the IKJRSVWD macro instruction with at least the 'parameter-type' operand when you use it:

- With the RSVWD keyword of the IKJOPER macro instruction to define the beginning of a list of the possible reserved words that can be an operator in an expression. The possible reserved words that can be operators in an expression are identified by a list of IKJNAME macro instructions that immediately follow the IKJRSVWD macro instruction.

- By itself to define a positional reserved word.

Code the IKJRSVWD macro instruction without operands when you use it:

- With the RSVWD keyword of the IKJTERM macro instruction to define the beginning of a list of possible reserved words that can be used as a figurative constant. The possible figurative constants are defined by a list of IKJNAME macros that immediately follow the IKJRSVWD macro instruction.

  In this case, simply code the IKJRSVWD macro instruction as:

```
| label  | IKJRSVWD |                                              |
```

The order in which you code the macros for positional parameters is the order in which the Parse service routine expects to find the parameters in the command string.

Figure 107 shows the format of the IKJRSVWD macro instruction. Each of the operands is explained following the figure. Appendix B describes the notation used to define macro instructions.

```
| label  | IKJRSVWD |  'parameter-type' [,PROMPT='prompt data'    ] |
|        |          |                   [,DEFAULT='default value' ] |
|        |          |                                               |
|        |          | [,HELP=('help data','help data',...)]         |
```

Figure 107. The IKJRSVWD Macro Instruction

label
: This name is used as the symbolic address within the PDL DSECT of the Parameter Descriptor Entry for the parameter described by this IKJRSVWD macro instruction.

  Note: The following operands are not coded on the IKJRSVWD macro when you use it with the RSVWD keyword of the IKJTERM macro instruction.

'parameter-type'
: This field is required so that the parameter can be identified when an error message is necessary. This field differs from the PROMPT field in that the PROMPT field is not required and if supplied is used only for a required parameter that is not entered by the terminal user. Blanks within the apostrophes are allowed.

PROMPT='prompt data'
: The parameter described by this IKJRSVWD macro instruction is required. The prompting data is the message to be issued if the parameter is not entered by the terminal user. If prompting is necessary and the terminal is in prompt mode, Parse adds a

message-identifying number (message ID) and the word "ENTER" to the beginning of the message before writing it to the terminal.
If prompting is necessary but the terminal is in no-prompt mode, Parse adds a message ID and the word "MISSING" to the beginning of the message before writing it to the terminal.

DEFAULT='default value'
The parameter described by this IKJRSVWD macro instruction is required, but the terminal user need not enter it. If the parameter is not entered, the value specified as the default value is used.

Note: If neither PROMPT nor DEFAULT is specified, the parameter is optional. The Parse service routine takes no action if the parameter is not present.

HELP=('help data','help data',...)
You can provide up to 255 second-level messages. Enclose each message in apostrophes and separate the messages by single commas. These messages are issued one at a time after each question mark entered by the terminal user in response to a prompting message from the Parse routine.
Parse adds a message ID and the word "ENTER" (in prompt mode) or "MISSING" (in no-prompt mode) to the beginning of each message before writing it to the terminal.

THE PARAMETER CONTROL ENTRY BUILT BY IKJRSVWD: The IKJRSVWD macro instruction generates the variable Parameter Control Entry (PCE) shown in Figure 108.

Number of Bytes	Field	Contents or Meaning
2		Flags. These flags are set to indicate options on the IKJRSVWD macro instruction.
	Byte 1	
	101.	This is an IKJRSVWD PCE.
	...1	PROMPT
	.... 1...	DEFAULT
	.... .0..	Reserved
	.... ..1.	HELP
	0	Reserved
	Byte 2	
	1...	This PCE is used with the IKJTERM macro as a figurative constant.
	0...	This PCE is not used with the IKJTERM macro as a figurative constant.
	.000 0000	Reserved
2		The hexadecimal length of this PCE.
2		Contains a hexadecimal offset from the beginning of the Parameter Descriptor List to the Parameter Descriptor Entry built by the Parse routine.

Note: The following fields are omitted if this PCE is used with the IKJTERM macro to describe a figurative constant.

Figure 108. The Parameter Control Entry Built by IKJRSVWD (Part 1 of 2)

Number of Bytes	Field	Contents or Meaning
4	Byte 1-2 Byte 3-4	Contains the hexadecimal length of the parameter-type field. Contains the offset of the parameter-type field (X'0012').
Variable		Contains the parameter-type field.
1		Contains the length of the default or prompting information supplied on the macro instruction.
Variable		Contains the default or prompting information supplied on the macro instruction.
2		Contains the length (including this field) of all the PCE fields used for second-level messages if HELP is specified on the macro.
1		The number of second-level messages specified on the macro instruction by the HELP= parameter.
2		Contains the length of this segment including this field, the message offset field and second-level message. Note: This field and the following two are repeated for each second-level message specified by HELP on the macro.
2		This field contains the message segment offset.
Variable		This field contains one second-level message specified by HELP on the macro instruction. This field and the two preceding fields are repeated for each second-level message specified.

Figure 108. The Parameter Control Entry Built by IKJRSVWD (Part 2 of 2)

IKJIDENT - Describing a Non-Delimiter Dependent Positional Parameter

Execute the IKJIDENT macro instruction to describe a positional parameter that does not depend upon a particular delimiter for its syntactical definition -- those parameters discussed under the heading "Positional Parameters Not Dependent on Delimiters."

These positioned parameters must be in the form of a character string, with restrictions on the beginning character, additional characters, and length.

The order in which you code the macro instructions for positional parameters is the order in which the Parse service routine expects to find the positional parameters in the command string.

Figure 109 shows the format of the IKJIDENT macro instruction. Each of the operands is explained following the figure. Appendix B describes the notation used to define macro instructions.

```
┌──────────┬──────────┬──────────────────────────────────────────────────┐
│  label   │ IKJIDENT │ 'parameter-type'  [,LIST][,RANGE][,PTBYPS]        │
│          │          │                                                  │
│          │          │ [,ASTERISK]┌,UPPERCASE┐[,MAXLNTH=number]          │
│          │          │            └,ASIS     ┘                           │
│          │          │                                                  │
│          │          │ ┌        ⎧ALPHA    ⎫┐┌        ⎧ALPHA    ⎫┐        │
│          │          │ │,FIRST= ⎪NUMERIC  ⎪││,OTHER= ⎪NUMERIC  ⎪│        │
│          │          │ │        ⎨ALPHANUM ⎬││        ⎨ALPHANUM ⎬│        │
│          │          │ │        ⎪ANY      ⎪││        ⎪ANY      ⎪│        │
│          │          │ │        ⎪NONATABC ⎪││        ⎪NONATABC ⎪│        │
│          │          │ └        ⎩NONATNUM ⎭┘└        ⎩NONATNUM ⎭┘        │
│          │          │                                                  │
│          │          │ ┌,PROMPT='prompt data'    ┐                        │
│          │          │ └,DEFAULT='default value' ┘                        │
│          │          │                                                  │
│          │          │ [VALIDCK=symbolic-address]                       │
│          │          │                                                  │
│          │          │ [,HELP=('help data','help data',...)]            │
└──────────┴──────────┴──────────────────────────────────────────────────┘
```

Figure 109. The IKJIDENT Macro Instruction

label
: This name is used within the PDL DSECT as the symbolic address of the Parameter Descriptor Entry for this positional parameter.

'parameter-type'
: This field is required so that the parameter can be identified when an error message is necessary. This field differs from the PROMPT field in that the PROMPT field is not required and if supplied is used only for a required parameter that is not entered by the terminal user. Blanks within the apostrophes are allowed.

LIST
: This positional parameter may be entered by the terminal user as a list, that is, in the form:

      Command Name (parameter,parameter,...)

RANGE
: This positional parameter may be entered by the terminal user as a range, that is, in the form:

      Command Name parameter:parameter

PTBYPS
: All prompting for the parameter is to be done in print inhibit mode. This option may be specified only when the PROMPT option is specified.

ASTERISK
An asterisk may be substituted for this positional parameter.

UPPERCASE
The parameter is to be translated to uppercase.

ASIS
The parameter is to be left as it was entered.

MAXLNTH=number
The maximum number of characters the string may contain. If you do not code the MAXLNTH operand, the Parse service routine accepts a character string of any length.

FIRST=
Specify the character type restriction on the first character of the string.

OTHER=
Specify the character type restriction on the characters of the string other than the first character.

Note: The restrictions on the characters of the string are specified by coding one of the following character types after the FIRST= and the OTHER= operands:

ALPHA
An alphabetic or national character. ALPHA is the default value for both the FIRST and the OTHER operands.

NUMERIC
A digit, 0 - 9.

ALPHANUM
An alphabetic, numeric, or national character.

ANY
Any character other than a blank, comma, tab, or semicolon. Parentheses must be balanced.

NONATABC
An alphabetic character only. National characters and numerics are excluded.

NONATNUM
An alphabetic or numeric character. National characters are excluded.

PROMPT='prompt data'
The parameter is required; the prompting data is the message to be issued if the parameter is not entered by the terminal user. If prompting is necessary and the terminal is in prompt mode, Parse adds a message-identifying number (message ID) and the word "ENTER" to the beginning of this message before writing it to the terminal.

If prompting is necessary but the terminal is in no-prompt mode, Parse adds a message ID and the word "MISSING" to the beginning of this message before writing it to the terminal.

DEFAULT='default value'
The parameter is required, but a default value may be used. If the parameter is not entered by the terminal user, the value specified as the default value is used.

Note: The parameter is optional if neither PROMPT nor DEFAULT is specified. The Parse service routine takes no action if the parameter specified by this IKJIDENT macro instruction is not present in the command buffer.

VALIDCK=symbolic-address

Supply the symbolic address of a validity checking subroutine if you want to perform additional validity checking on this parameter. The Parse service routine calls the addressed routine after first determining that the parameter is syntactically correct.

HELP=('help data','help data'...)

You can provide up to 255 second-level messages. Enclose each message in apostrophes and separate the messages by single commas. These messages are issued one at a time after each question mark entered by the terminal user in response to a prompting message from the Parse service routine. These messages are not sent to the user when the prompt is for a password on a DSNAME or USERID parameter.

Parse adds a message ID and the word "ENTER" (in prompt mode) or "MISSING" (in no-prompt mode) to the beginning of each message before writing it to the terminal.

THE PARAMETER CONTROL ENTRY BUILT BY IKJIDENT: The IKJIDENT macro instruction generates the variable length Parameter Control Entry (PCE) shown in Figure 110.

Number of Bytes	Field	Contents or Meaning
2		Flags. These flags are set to indicate which options were coded in the IKJIDENT macro instruction.
	Byte 1	
	100.	This is an IKJIDENT PCE.
	...1	PROMPT
	.... 1...	DEFAULT
	.... .0..	Reserved
	.... ..1.	HELP
	1	VALIDCK
	Byte 2	
	1...	LIST
	.1..	ASIS
	..1.	RANGE
	...0 0000	Reserved
2		Length of the Parameter Control Entry. This field contains a hexadecimal number representing the number of bytes in this IKJIDENT PCE.
2		Contains a hexadecimal offset from the beginning of the Parameter Descriptor List to the related Parameter Descriptor Entry built by the PARSE service routine.
1		A flag field indicating the options coded on the IKJIDENT macro instruction.
	1...	ASTERISK
	.1..	MAXLNTH
	..1.	PTBYPS
	...0 0000	Reserved

Figure 110. The Parameter Control Entry Built by IKJIDENT (Part 1 of 3)

Number of Bytes	Field	Contents or Meaning
1		This field contains a hexadecimal number indicating the character type restriction on the first character of the character string described by the IKJIDENT macro instruction.
	HEX	Acceptable Characters:
	0	Any (except blank, comma, tab, semicolon).
	1	Alphabetic or national.
	2	Numeric.
	3	Alphabetic, national, or numeric.
	4	Alphabetic.
	5	Alphabetic or numeric.
	6 to FF	Not used.
1		This field contains a hexadecimal number indicating the character type restriction on the other characters of the character string described by the IKJIDENT macro instruction.
	HEX	Acceptable Characters:
	0	Any (except blank, comma, tab, semicolon).
	1	Alphabetic or national.
	2	Numeric.
	3	Alphabetic, national, or numeric.
	4	Alphabetic.
	5	Alphabetic or numeric.
	6 to FF	Not used.
2		This field contains a hexadecimal number representing the length of the parameter type segment. This figure includes the length of this field, the length of the message segment offset field, and the length of the Parameter type field supplied on the IKJIDENT macro instruction.
2		This field contains the message segment offset. It is set to X'0012'.
Variable		This field contains the field supplied as the parameter type operand of the IKJIDENT macro instruction.
1		This field contains a hexadecimal number representing the mexaimum number of characters the string may contain. This field is present only if the MAXLNTH operand was coded on the IKJIDENT macro instruction.
1		This field contains the length minus one of the default or prompting information supplied on the IKJIDENT macro instruction. This field and the next are present only if DEFAULT or PROMPT were specified on the IKJIDENT macro instruction.
Variable		This field contains the prompting or default information supplied on the IKJIDENT macro instruction.

Figure 110. The Parameter Control Entry Built by IKJIDENT (Part 2 of 3)

Number of Bytes	Field	Contents or Meaning
2		This field contains a hexadecimal figure representing the length in bytes of all the PCE fields used for second-level messages. The figure includes the length of this field. The fields are present only if HELP is specified on the IKJIDENT macro instruction.
1		This field contains a hexadecimal number representing the number of second-level messages specified by HELP on this IKJIDENT PCE.
2		This field contains a hexadecimal number representing the length of this HELP segment. The figure includes the length of this field, the message segment offset field, and the length of the information. These fields are repeated for each second-level message specified by HELP on the IKJIDENT macro instruction.
2		This field contains the message segment offset. It is set to X'0000'.
Variable		This field contains one second-level message supplied on the IKJIDENT macro instruction specified by HELP. This field and the two preceding ones are repeated for each second-level message supplied on the IKJIDENT macro instruction; these fields do not appear if no second-level message data was supplied.
3		The address of a validity checking routine. This field is present only if a validity checking address was included in the IKJPOSIT macro instruction.

Figure 110. The Parameter Control Entry Built by IKJIDENT (Part 3 of 3)

IKJKEYWD - Describing a Keyword Parameter

Execute the IKJKEYWD macro instruction to describe a keyword parameter. Execute a series of IKJNAME macro instructions to indicate the possible names for the keyword parameter. Keyword parameters may appear in any order in the command but must follow all positional parameters. A user is never required to enter a keyword parameter; if he does not, the default value you supply, if you choose to supply one, is used. Keywords may consist of any combination of alphameric characters up to 31 characters in length, the first of which must be an alphabetic character.

Figure 111 shows the format of the IKJKEYWD macro instruction. Each of the operands is explained following the figure. Appendix B describes the notation used to define macro instructions.

label	IKJKEYWD	[DEFAULT='default-value']

Figure 111. The IKJKEYWD Macro Instruction

label
This name is used within the PDL DSECT as the symbolic address of the Parameter Descriptor Entry for this parameter.

DEFAULT='default-value'
The default value you specify is the value that is used if this keyword is not present in the command buffer. Specify the valid keyword names with IKJNAME macro instructions following this IKJKEYWD macro instruction.

THE PARAMETER CONTROL ENTRY BUILT BY IKJKEYWD: The IKJKEYWD macro instruction generates the variable length parameter Control Entry (PCE) shown in Figure 112.

Number of Bytes	Field	Contents or Meaning
2		Flags. These flags are set to indicate which options were coded in the IKJKEYWD macro instruction.
	Byte 1	
	010.	This is an IKJKEYWD PCE
	...0	Reserved
	.... 1...	DEFAULT
	.... .000	Reserved.
	Byte 2	
	0000 0000	Reserved.
2		Length of the Parameter Control Entry. This field contains a hexadecimal number representing the number of bytes in this IKJEKYWD PCE.

Figure 112. The Parameter Control Entry Built by IKJKEYWD (Part 1 of 2)

Number of Bytes	Field	Contents or Meaning
2		This field contains a hexadecimal offset from the beginning of the Parameter Descriptor List to the related Parameter Descriptor Entry built by the PARSE service routine.
1		This field contains the length minus one of the default information supplied on the IKJKEYWD macro instruction. This field and the next are present only if DEFAULT was specified on the IKJKEYWD macro instruction.
Variable		This field contains the default value supplied on the IKJKEYWD macro instruction.

Figure 112. The Parameter Control Entry Built by IKJKEYWD (Part 2 of 2)

IKJNAME - Listing the Keyword or Reserved Word Parameter Names

The IKJNAME macro instruction may be coded with the following two macro instructions.

1. With the IKJKEYWD macro instruction to define keyword parameter names.

2. With the IKJRSVWD macro instruction to define reserved word parameter names.

A description and format of the IKJNAME macro instruction for both methods of coding follows.

1. Code a series of IKJNAME macro instructions to indicate the possible names for a keyword parameter. One IKJNAME macro instruction is needed for each possible keyword name. Code the IKJNAME macro instructions immediately following the IKJKEYWD macro instruction to which they pertain.

Figure 113 shows the format of the IKJNAME macro instruction. Each of the operands is explained following the figure. Appendix B describes the notation used to define macro instructions.

	IKJNAME	'keyword-name'[,SUBFLD=subfield-name] [,INSERT='keyword-string']

Figure 113. The IKJNAME Macro Instruction (When used with the IKJKEYWD Macro Instruction)

keyword-name
: One of the valid keyword parameters for the IKJKEYWD macro instruction that precedes this IKJNAME macro instruction.

SUBFLD=subfield-name
: This option indicates that this keyword name has other parameters associated with it. Use the subfield-name as the label field of the IKJSUBF macro instruction that begins the description of the possible parameters in the subfield.

INSERT='keyword-string'

The use of some keyword parameters may imply that other keyword parameters are required. The Parse service routine inserts the keyword string specified into the command string just as if it had been entered as part of the original command string. The command buffer is not altered.

2. Code a series of IKJNAME macro instructions to indicate the possible names for reserved words. One IKJNAME macro instruction is needed for each possible reserved word name. Code the IKJNAME macro instructions immediately following the IKJRSVWD macro instruction to which they apply.

Figure 114 shows the format of the IKJNAME macro instruction. Each of the operands is explained following the figure. Appendix B describes the notation used to define macro instructions.

	IKJNAME	'reserved-word name'

Figure 114. The IKJNAME Macro Instruction (when used with the IKJRSVWD macro)

reserved-word name

One of the valid reserved word parameters for the IKJRSVWD macro instruction that precedes the IKJNAME macro instructions.

Note: The IKJNAME macro instruction has two uses when coded with the IKJRSVWD macro instruction. The reserved-words identified on the IKJNAME macros may be figurative constants when the IKJRSVWD macro is chained from an IKJTERM macro, or operators in an expression when the IKJRSVWD macro is chained from the IKJOPER macro.

THE PARAMETER CONTROL ENTRY BUILT BY IKJNAME: The IKJNAME macro instruction generates the variable length Parameter Control Entry (PCE) shown in Figure 115.

Note: Only the first four fields are valid when the IKJNAME macro instruction is coded with the IKJRSVWD macro instruction.

Number of Bytes	Field	Contents or Meaning
2		Flags. These flags are set to indicate which options were coded in the IKJNAME macro instruction.
	Byte 1	
	011.	This is an IKJNAME PCE.
	...0 0...	Reserved.
	.... .1..	SUBFLD
	00	Reserved.
	Byte 2	
	000.	Reserved.
	...1	INSERT
	 0000	Reserved.
2		Length of the Parameter Control Entry. This field contains a hexadecimal number representing the number of bytes in this IKJNAME PCE.
1		This field contains the length minus one of the keyword or reserved word name specified on the IKJNAME macro instruction.
Variable		This field contains the keyword or reserved word name specified on the IKJNAME macro instruction.
2		This field contains a hexadecimal offset, plus one, from the beginning of the Parameter Control List to the beginning of a subfield PCE. This field is present only if the SUBFLD operand was specified in the IKJNAME macro instruction.
1		This field contains the length minus one of the keyword string included as the INSERT operand in the IKJNAME macro instruction. This field and the next are not present if INSERT was not specified.
Variable		This field contains the keyword string specified as the INSERT operand of the IKJNAME macro instruction.

Figure 115. The Parameter Control Entry Built by IKJNAME

IKJSUBF - Describing a Keyword Subfield

Keyword parameters may have subfields associated with them. A subfield consists of a parenthesized list of parameters directly following the keyword.

Execute the IKJSUBF macro instruction to indicate the beginning of a subfield description. The IKJSUBF macro instruction ends the main part of the Parameter Control List or the previous subfield description, and begins a new subfield description.

Note that the IKJSUBF macro instruction is used only to begin the subfield description; the subfield is described using the IKJPOSIT, IKJIDENT, and IKJKEYWD macro instructions, depending upon the type of parameters within the subfield.

You must use the name you have coded as the SUBFLD operand of the IKJNAME macro instruction for the label of this macro instruction.

Figure 116 shows the format of the IKJSUBF macro instruction. Appendix B describes the notation used to define macro instructions.

label	IKJSUBF	

Figure 116. The IKJSUBF Macro Instruction

label
: The name you supply as the label of this macro instruction must be the same name you have coded as the SUBFLD operand of the IKJNAME macro instruction describing the keyword name that takes this subfield.

THE PARAMETER CONTROL ENTRY BUILT BY IKJSUBF: The IKJSUBF macro instruction generates the Parameter Control Entry (PCE) shown in Figure 117.

Number of Bytes	Field	Contents or Meaning
1		Flags. These flags indicate which type of PCE this is.
	000.	This PCE indicates an end-of-field. These end-of-field indicators are present in IKJSUBF and IKJENDP PCEs; they indicate the end of a previous subfield or of the PCL itself.
	...0 0000	Reserved.
2		This field contains a hexadecimal number representing the offset within the PCL to the first IKJKEYWD PCE or to the next end-of-field indicator if there are no keywords in this subfield.

Figure 117. The Parameter Control Entry Built by IKJSUBF

IKJENDP - Ending the Parameter Control List

Execute the IKJENDP macro instruction to inform the Parse service routine that it has reached the end of the Parameter Control List built for this command.

Figure 118 shows the format of the IKJENDP macro instruction. Appendix B describes the notation used to define macro instructions.

	IKJENDP	

Figure 118. The IKJENDP Macro Instruction

THE PARAMETER CONTROL ENTRY BUILT BY IKJENDP: The IKJENDP macro instruction generates the Parameter Control Entry (PCE) shown in Figure 119. It is merely an end-of-field indicator.

Number of Bytes	Field	Contents or Meaning
1		Flags. These flags are set to indicate end-of-field.
	000.	End-of-field indicator. Indicates the end of the PCL.
	...0 0000	Reserved.

Figure 119. The Parameter Control Entry Built by IKJENDP

IKJRLSA - Releasing Storage Allocated by Parse

Execute the IKJRLSA macro instruction to release storage allocated by the Parse service routine and not previously released by Parse. This storage consists of the Parameter Descriptor List (PDL) returned by the Parse service routine and any storage obtained for new data received by Parse as a result of a prompt.

If the return code from the Parse service routine is non-zero, all storage allocated by Parse has been freed by Parse. In that case, this macro instruction need not be issued, but will not cause an error if it is issued.

Figure 120 shows the format of the IKJRLSA macro instruction. Each of the operands is explained following the figure. Appendix B describes the notation used to define macro instructions.

label	IKJRLSA	{address of the answer place (1-12)}

Figure 120. The IKJRLSA Macro Instruction

address of the answer place
: The address of the word within the Parse Parameter List in which Parse placed a pointer to the PDL when control was returned to the command processor. This address may be loaded into one of the general registers 1 through 12, right adjusted with the unused high order bits set to zero. See the section headed "Passing Control to the Parse Service Routine" for a description of the Parse Parameter List.

PASSING CONTROL TO THE PARSE SERVICE ROUTINE

You pass control to the Parse service routine by issuing a LINK macro instruction specifying IKJPARS as the entry point. Before you LINK to the Parse service routine however, you must build a Parse Parameter List (PPL), and place its address into register 1. This PPL must remain intact until Parse returns control to the calling routine. Figure 121 shows this flow of control between a Command Processor and the Parse service routine.

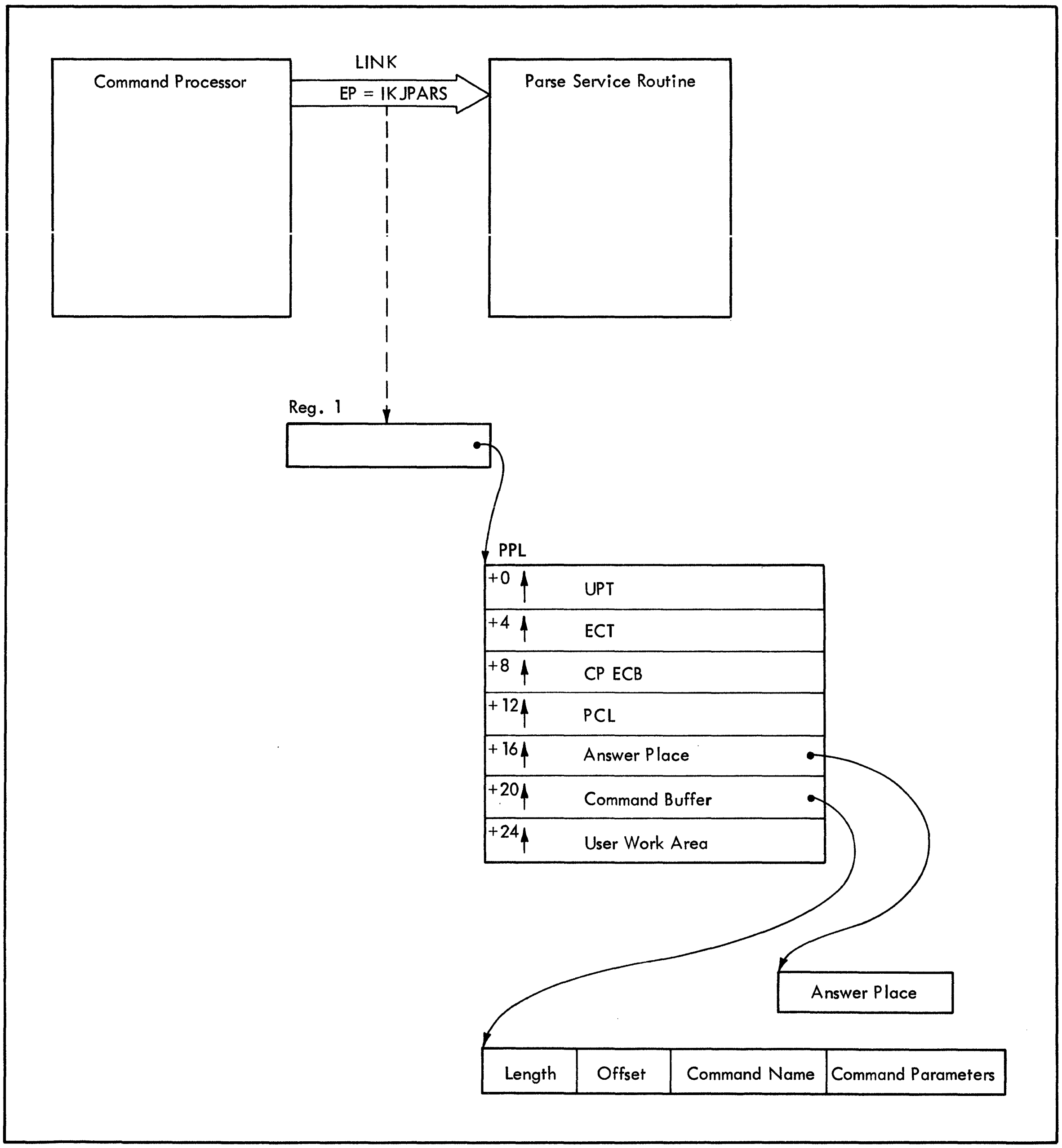


Figure 121. Control Flow Between Command Processor and Parse

The Parse Parameter List

The Parse Parameter List (PPL) is a seven-word parameter list containing addresses required by the Parse service routine.

The PPL is defined by the IKJPPL DSECT. Figure 122 shows the format of the Parse Parameter List.

Number of Bytes	Field	Contents or Meaning
4	PPLUPT	The address of the User Profile Table.
4	PPLECT	The address of the Environment Control Table.
4	PPLECB	The address of the Command Processor's Event Control Block. The ECB is one word of storage, declared and initialized to zero by the Command Processor.
4	PPLPCL	The address of the Parameter Control List created by the Command Processor using the Parse macro instructions. Use the label on the IKJPARM macro instruction as the symbolic address of the PCL.
4	PPLANS	The address of a fullword of storage, supplied by the calling routine, in which the Parse service routine places a pointer to the Parameter Descriptor List (PDL). If the parse of the command buffer is unsuccessful, Parse sets the pointer to the PDL to FF000000.
4	PPLCBUF	The address of the command buffer.
4	PPLUWA	The address of a user supplied work area. This field can contain anything that the calling routine wishes passed to a validity checking routine.

Figure 122. The Parse Parameter List

FORMATS OF THE PDES RETURNED BY PARSE

Parse returns the results of the scan of the command buffer to the command processor in a Parameter Descriptor List (PDL). The PDL, built by Parse, consists of Parameter Descriptor Entries (PDE), which contain pointers to the parameters, indicators of the options specified, and pointers to the subfield parameters entered with the command operands.

Use the IKJPARMD DSECT to map the PDL and each of the PDEs. Base the IKJPARMD DSECT on the PDL address returned by the Parse service routine. The PPLANS field of the Parse Parameter List points to a fullword of storage that contains the address of the PDL. Then use the labels you used on the Parse macro instructions to access the corresponding PDEs.

The format of the PDE depends upon the type of parameter parsed. For a discussion of parameter types, see the topic "Command Parameter Syntax." The following description of the possible PDEs within a PDL shows each of the PDE formats and the type of parameters they describe.

The PDL Header

The PDL begins with a two word header. The DSECT= operand of the IKJPARM macro instruction provides a name for the DSECT created to map the PDL. Use this name as the symbolic address of the beginning of the PDL header.

+0 A pointer to the next block of storage	
+4 Subpool number	+6 LENGTH

Pointer to the next block of storage:
The Parse service routine gets storage for the PDL and for any data received as the result of a prompt. Each block of storage obtained begins with another PDL header. The blocks of storage are forward chained by this field. A forward chain pointer of FF000000 in this field indicates that this is the last storage element obtained.

Subpool number:
This field will always indicate subpool 1. All storage allocated by the Parse service routine for the PDL and for data received from a prompt is allocated from subpool 1.

Length:
This field contains a hexadecimal number indicating the length of this block of storage (this PDL); the length includes the header.

PDEs Created for Positional Parameters

The labels you use to name the macro instructions provide access to the corresponding PDEs. The positional parameters described by the IKJPOSIT, IKJTERM, IKJOPER, IKJRSVWD and the IKJIDENT macro instructions have the following PDE formats.

SPACE, DELIMITER: The Parse service routine does not build a PDE for either a SPACE or a DELIMITER parameter.

STRING, PSTRING, AND QSTRING: PARSE uses the IKJPOSIT macro to build a two-word PDE to describe a STRING, PSTRING, or a QSTRING parameter; the PDE has the following format:

+0 A pointer to the character string		
+4 Length	+6 Flags	+7 Reserved

Pointer to the character string:
Contains a pointer to the beginning of the character string, or a zero if the parameter was omitted.

Length:
Contains the length of the string. Any punctuation around the character string is not included in this length figure.
The length is zero if the string is omitted or if the string is null.

Flags:

0...	The parameter is not present.
1...	The parameter is present.
.xxx xxxx	Reserved bits.

Note: If the string is null, the pointer is set, the length is zero, and the flag bit is 1.

VALUE: Parse uses the IKJPOSIT macro to build a two word PDE to describe a VALUE parameter; the PDE has the following format:

+0 A pointer to the character string		
+4 Length	+6 Flags	+7 Type-char.

Pointer to the character string:
Contains a pointer to the beginning of the character string, that is, the first character after the quote. Contains a zero if the VALUE parameter is not present.

Length:
Contains the length of the character string excluding the quotes.

Flags:

0...	The parameter is not present.
1...	The parameter is present.
.xxx xxxx	Reserved bits.

Type-character:
Contains the letter that precedes the quoted string.

DSNAME, DSTHING: Parse uses the IKJPOSIT macro to build a six-word PDE to describe a DSNAME or a DSTHING parameter. The PDE has the following format:

+0 A pointer to the dsname		
+4 Length1	+6 Flags1	+7 Reserved
+8 A pointer to the member name		
+12 Length2	+14 Flags2	+15 Reserved
+16 A pointer to the password		
+20 Length3	+22 Flags3	+23 Reserved

Pointer to the dsname:
Contains a pointer to the beginning of the data set name. Contains zero if the data set name was omitted.

Length1:
Contains the length of the data set name. If the data set name is contained in quotes, this length figure does not include the quotes.

Flags1:

0...	The data set name is not present.
1...	The data set name is present.
.0..	The data set name is not contained within quotes.
.1..	The data set name is contained within quotes.
..xx xxxx	Reserved bits.

Pointer to the member name:
Contains a pointer to the beginning of the member name. Contains zero if the member name was omitted.

Length2:
Contains the length of the member name. This length figure does not include the parentheses around the member name.

Flags2:

0...	The member name is not present.
1...	The member name is present.
.xxx xxxx	Reserved bits.

Pointer to the password:
Contains a pointer to the beginning of the password. Contains zero if the password was omitted.

Length3:
Contains the length of the password.

Flags3:

0...	The password is not present.
1...	The password is present.
.xxx xxxx	Reserved bits.

ADDRESS: Parse uses the IKJPOSIT macro to build a nine-word PDE to describe an ADDRESS parameter. The PDE has the following format:

<table>
<tr><td colspan="4">+0
A pointer to the load name</td></tr>
<tr><td colspan="2">+4
Length1</td><td>+6
Flags1</td><td>+7
Reserved</td></tr>
<tr><td colspan="4">+8
A pointer to the entry name</td></tr>
<tr><td colspan="2">+12
Length2</td><td>+14
Flags2</td><td>+15
Reserved</td></tr>
<tr><td colspan="4">+16
A pointer to the address string</td></tr>
<tr><td colspan="2">+20
Length3</td><td>+22
Flags3</td><td>+23
Reserved</td></tr>
<tr><td>+24
Flags4</td><td>+25
Sign</td><td colspan="2">+26
Indirect count</td></tr>
<tr><td colspan="4">+28
A pointer to the first expression value PDE</td></tr>
<tr><td colspan="4">+32
Reserved for use by user validity check rtn.</td></tr>
</table>

Pointer to the load name:
Contains a pointer to the beginning of the load module name. Contains zero if no load module name was specified.

Length1:
Contains the length of the load module name, excluding the period.

Flags1:

0...	The load module name is not present.
1...	The load module name is present.
.xxx xxxx	Reserved bits.

Pointer to the entry name:
Contains a pointer to the name of the CSECT, zero if the CSECT name is not specified.

Length2:
Contains the length of the entryname, excluding the period.

Flags2:

0...	The entry name is not present.
1...	The entry name is present.
.xxx xxxx	Reserved bits.

Pointer to the address string:
Contains a pointer to the address string portion of a qualified address. Contains a zero if the address string was not specified.

Length3:

Contains the length of the address string portion of a qualified address. This length count excludes the following characters for the following address types:

1. Relative address - excludes the plus sign.
2. Register address - excludes letters.
3. Absolute address - excludes period.

Flags3:

0...	The address string is not present.
1...	The address string is present.
.xxx xxxx	Reserved bits.

Flags4:

The bits set in this one byte flag field indicate the type of address found by the Parse service routine.

Bit Setting	Hex	Meaning
0000 0000	00	Absolute address.
1000 0000	80	Symbolic address.
0100 0000	40	Relative address.
0010 0000	20	General register.
0001 0000	10	Double precision floating point register.
0000 1000	08	Single precision floating point register.

Sign:

Contains the arithmetic sign character used before an expression value. Contains a zero if the address is not an address expression.

Indirect count:

Contains a number representing the number of levels of indirect addressing.

Pointer to the first expression value PDE:

If the address is in the form of an address expression, this is a pointer to the PDE for the first expression value. Contains hexadecimal FF000000 if the address is not an address expression.

User word for validity checking routine:

A word provided for use by the user-written validity checking routine.

EXPRESSION VALUE: If an ADDRESS parameter is found to be in the form of an address expression, the Parse service routine builds an expression value PDE for each expression value within the address expression. These expression value PDEs are chained together, beginning at the eighth word of the address PDE built by Parse to describe the address parameter. The last expression value PDE is indicated by hexadecimal FF000000 in its fourth word, the forward chaining field.

Parse uses the IKJPOSIT macro to build a four word PDE to describe an expression value, it has the following format:

<table>
<tr><td colspan="3">+0
A pointer to the address string</td></tr>
<tr><td colspan="2">+4
Length3</td><td>+6
Reserved</td></tr>
<tr><td>+8
Flags5</td><td>+9
Sign</td><td>+10
Indirect count</td></tr>
<tr><td colspan="3">+12
A pointer to the next expression value</td></tr>
</table>

Pointer to the address string:
Contains a pointer to the expression value address string.

Length3:
Contains the length of the expression value address string. The N is not included in this length value.

Flags5:
The Parse service routine sets these flags to indicate the type of expression value:

Bit Setting	Hex	Meaning
0000 0100	04	This is a decimal expression value.
0000 0010	02	This is a hexadecimal expression value.

Sign:
Contains the arithmetic sign character used before an expression value.

Indirect count:
Contains a number representing the number of levels of indirect addressing within this particular address expression.

Pointer to the next expression value PDE:
Contains a pointer to the next expression value PDE if one is present; contains hexadecimal FF000000 if this is the last expression value PDE.

USERID: Parse uses the IKJPOSIT macro to build a four-word PDE to describe a USERID parameter; it has the following format:

+0 A pointer to the userid		
+4 Length1	+6 Flags1	+7 Reserved
+8 A pointer to the password		
+12 Length2	+14 Flags2	+15 Reserved

Pointer to the userid:
Contains a pointer to the beginning of the userid. Contains zero if the userid was omitted.

Length1:
Contains the length of the userid.

Flags1:

0...	The userid is not present.
1...	The userid is present.
.xxx xxxx	Reserved bits.

Pointer to the password:
Contains a pointer to the beginning of the password. Contains zero if the password is omitted.

Length2:
Contains the length of the password, excluding the slash.

Flags2:

0...	The password is not present.
1...	The password is present.
xxxx xxxx	Reserved bits.

CONSTANT: Parse uses the IKJERM macro to build a five word PDE to describe a CONSTANT parameter. The PDE has the following format:

+0 Length1	+1 Length2	+2 Reserved
+4 Reserved Word Number		+6 Flags
+8 A pointer to the string of digits		
+12 A pointer to the exponent		
+16 A pointer to the decimal point		

Length1

Contains the length of term entered, depending on the type of parameter entered as follows:

- For a fixed-point numeric literal, the length includes the digits but not the sign or decimal point.

- For a floating-point numeric literal, the length includes the mantissa (string of digits preceding the letter E) but not the sign or decimal point.

- For a non-numeric literal, the length includes the string of characters but not the apostrophes.

Length2

For a floating-point numeric literal, length2 contains the length of the string of digits following the letter E but not the sign.

Reserved Word Number

The reserved word number contains the number of the IKJNAME macro that corresponds to the entered name.

Note: The possible names of reserved words are given by coding a list of IKJNAME macros following an IKJRSVWD macro. One IKJNAME macro is needed for each possible name. If the name entered does not correspond to one of the names in the IKJNAME macro list then this field is set to zero.

Flags

Byte 1

0...	The parameter is missing.
1...	The parameter is present.
.1..	Constant.
..1.	Variable.
...1	Statement Number.
.... 1...	Fixed-point numeric literal.
.... .1..	Non-numeric literal.
.... ..1.	Figurative constant.
....1	Floating-point numeric literal.

Byte 2

0...	Sign on constant is either plus or omitted.
1...	Sign on constant is minus.
.0..	Sign on exponent of floating-point numeric literal is either plus or omitted.
.1..	Sign on exponent of floating-point numeric literal is munus.
..1.	Decimal point is present.
...x xxxx	Reserved bits.

Pointer to the string of digits:

Contains a pointer to the string of digits, not including the sign if entered. Contains zero if a constant type of parameter is not entered.

Pointer to the exponent:

Contains a pointer to the string of digits in a floating-point numeric literal following the letter E, not including the sign if entered.

Pointer to the decimal point:

Contains a pointer to the decimal point in a fixed-point or floating-point numeric literal. If a decimal point is not entered, this field is zero.

STATEMENT NUMBER: Parse uses the IKJTERM macro to build a five word PDE to describe a STATEMENT NUMBER parameter. The PDE has the following format:

<table>
<tr><td>+0
Length1</td><td>+1
Length2</td><td>+2
Length3</td><td>+3
Reserved</td></tr>
<tr><td colspan="2">+4
Reserved</td><td colspan="2">+6
Flags</td></tr>
<tr><td colspan="4">+8
A pointer to the program-id</td></tr>
<tr><td colspan="4">+12
A pointer to the line number</td></tr>
<tr><td colspan="4">+16
A pointer to the verb number</td></tr>
</table>

Length1
Contains the length of the program-id specified but not including the following period. Contains zero if the program-id is not present.

Length2
Contains the length of the line number entered but not including the delimiting periods. Contains zero if the line number is not present.

Length3
Contains the length of the verb number entered but not including the preceding period. Contains zero if the verb number is not present.

Flags

Byte 1	
0...	The parameter is missing.
1...	The parameter is present.
.1..	Constant.
..1.	Variable.
...1	Statement Number.
.... xxxx	Reserved.
Byte 2	
xxxx xxxx	Reserved.

Pointer to the program-id:
Contains a pointer to the program-id, if entered. Contains zero if not present.

Pointer to the line number:
Contains a pointer to the line number, if entered. Contains zero if not present.

Pointer to the verb number:
Contains a pointer to the verb number, if entered. Contains zero if not present.

VARIABLE: Parse builds a five word PDE (when using the IKJTERM macro) to describe a VARIABLE parameter. The PDE has the following format:

+0 A pointer to the data-name			
+4 Length1	+5 Reserved	+6 Flags	+7 Reserved
+8 A pointer to the PDE for the first qualifier.			
+12 A pointer to the program-id name.			
+16 Length2	+17 Number of Qualifiers	+18 Number of Subscripts	+19 Reserved

Pointer to the data-name:
Contains a pointer to the data-name. If a program-id qualifier precedes the data-name, this pointer points to the first character after the period of the program-id qualifier.

Length1
Contains the length of the data-name.

Flags
Byte 1

0...	The parameter is missing.
1...	The parameter is present.
.1..	Constant.
..1.	Variable.
...1	Statement Number.
.... xxxx	Reserved.

Pointer to the PDE for the first qualifier:
Contains a pointer to the PDE describing the first qualifier of the data-name, if any.
This field contains X'FF000000' if no qualifiers are entered.

Note: The format of the PDE for a data-name qualifier follows this description.

Pointer to the program-id name:
Contains a pointer to the program-id name, if entered. This field contains zero if the optional program-id name is not present.

Length2
Contains the length of the program-id name, if entered. Contains zero if the optional program-id name is not present.

Number of Qualifiers
Contains the number of qualifiers entered for this data-name. (For example: if data-name A of B is entered, this field would contain 1.)

Number of Subscripts
Contains the number of subscripts entered for this data-name. (For example: if data-name A(1,2) is entered this field would contain 2.)

The format of a DATA-NAME QUALIFIER is:

+0 A pointer to the data-name qualifier.			
+4 Length	+5 Reserved	+6 Flags	+7 Reserved
+8 A pointer to the PDE for the next qualifier.			

Pointer to the data-name qualifier.
Contains a pointer to the data-name qualifier.

Length:
Contains the length of the data-name qualifier.

Flags

xxxx xxxx	Reserved

Pointer to the PDE for the next qualifier:
Contains a pointer to the PDE describing the next qualifier, if any. This field contains X'FF000000' for the last qualifier.

RESERVED WORD: Parse uses the IKJRSVWD macro to build a two word PDE (using the IKJRSVWD macro instruction) to describe a RESERVED WORD parameter. The PDE has the following format:

Note: This PDE is not used when the IKJRSVWD macro instruction is chained from an IKJTERM macro instruction. In this case, the reserved-word number is returned in the CONSTANT parameter PDE built by the IKJTERM macro instruction.

+0 Reserved	+2 Reserved-word number	
+4 Reserved	+6 Flags	+7 Reserved

Reserved-word number:
The reserved-word number contains the number of the IKJNAME macro instruction that corresponds to the entered name.

Note: The possible names of reserved words are given by coding a list of IKJNAME macros following an IKJRSVWD macro. One IKJNAME macro is needed for each possible name. If the name entered does not correspond to one of the names in the IKJNAME macro list then this field is set to zero.

Flags
Byte 1

0...	The parameter is missing.
1...	The parameter is present.
.xxx xxxx	Reserved.

EXPRESSION: Parse uses the IKJOPER macro to build a two word PDE to describe an EXPRESSION parameter. The PDE has the following format:

<table>
<tr><td colspan="3">+0
Reserved</td></tr>
<tr><td>+4
Reserved</td><td>+6
Flags</td><td>+7
Reserved</td></tr>
</table>

Flags
 0... .... The entire parameter (expression) is missing.
 1... .... The entire parameter (expression) is present.
 .xxx xxxx Reserved.

IKJIDENT PDE: Parse uses the IKJIDENT macro instruction to build a two-word PDE to describe a non-delimiter dependent positional parameter; it has the following format:

<table>
<tr><td colspan="3">+0
A pointer to the character string</td></tr>
<tr><td>+4
Length</td><td>+6
Flags</td><td>+7
Reserved</td></tr>
</table>

Pointer to the character string:
 Contains a pointer to the beginning of the character string.
 Contains zero if the character string is omitted.

Length:
 Contains the length of the character string.

Flags:
 0... .... The parameter is not present.
 1... .... The parameter is present.
 .xxx xxxx Reserved bits.

Affect of List and Range Options on PDE Formats

The formats of the IKJPARMD mapping DSECT and of the PDEs built by the Parse service routine are affected by the options you specify in the Parse macro instructions, as well as by the type of parameter specified. If you specify the LIST or the RANGE options in the Parse macro instructions describing positional parameters, the IKJPARMD DSECT and the PDEs returned by the Parse service routine are modified to reflect these options.

LIST: The LIST option may be used with the following positional parameter types:

- USERID
- DSNAME
- DSTHING
- ADDRESS
- VALUE
- CONSTANT
- VARIABLE
- STATEMENT NUMBER
- Any non-delimiter dependent positional parameter.

If you specify the LIST option in the Parse macro instructions describing the above listed positional parameter types, the Parse service routine allocates an additional word for the PDE created to describe the positional parameter. This word is allocated even though a list may not actually be entered by the terminal user. If a list is not entered, this word is set to hexadecimal FF000000. If a list is entered, the additional word will be used to chain the PDEs created for each element found in the list. Each additional PDE has a format identical to the one described for that parameter type within the IKJPARMD DSECT. Since the number of elements in a list is variable, the number of PDEs created by the Parse service routine is also variable. The chain word of the PDE created for the last element of the list is set to hexadecimal FF000000.

Figure 123 shows the PDL returned by the Parse service routine after three positional parameters have been entered. In this case, the first two parameters, a USERID and a STRING parameter, had been defined as not accepting lists. The third parameter, a VALUE parameter, had the LIST option coded in the IKJPOSIT macro instruction that defined the parameter syntax. The VALUE parameter was entered as a two element list.

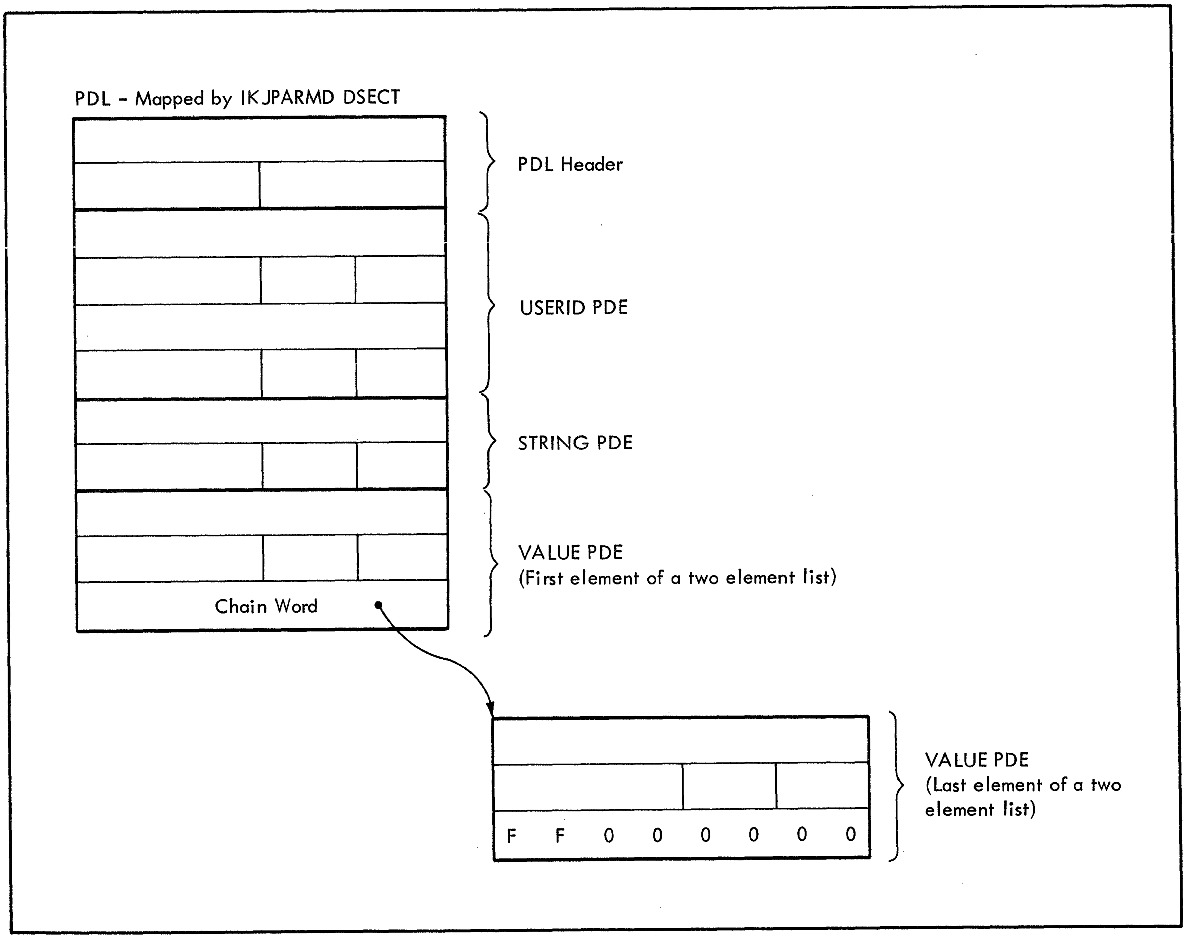

PDL - Mapped by IKJPARMD DSECT
PDL Header
USERID PDE
STRING PDE
VALUE PDE
(First element of a two element list)
Chain Word
F F 0 0 0 0 0 0
VALUE PDE
(Last element of a two element list)

Figure 123. A PDL Showing PDEs Describing a List

RANGE: The RANGE option may be used with the following positional parameter types:

- ADDRESS
- VALUE
- CONSTANT
- VARIABLE
- STATEMENT NUMBER
- Any non-delimiter dependent positional parameter.

If you specify the RANGE option in the Parse macro instructions describing the above listed positional parameter types, the Parse service routine builds two identical, sequential PDEs within the PDL returned to the calling routine. Space is allocated for the second PDE even though a range may not actually be supplied by the terminal user. If a range is not supplied, the second PDE is set to zero. The flag bit which is normally set for a missing parameter will also be zero in the second PDE.

Figure 124 shows the PDL returned by the Parse service routine after two positional parameters have been entered. In this case, the first parameter is a USERID parameter and the second parameter is a VALUE parameter that had the RANGE option coded in the IKJPOSIT macro instruction that defined the parameter syntax. For this example, the VALUE parameter was not entered as a range, and, consequently, the second PDE is set to zero.

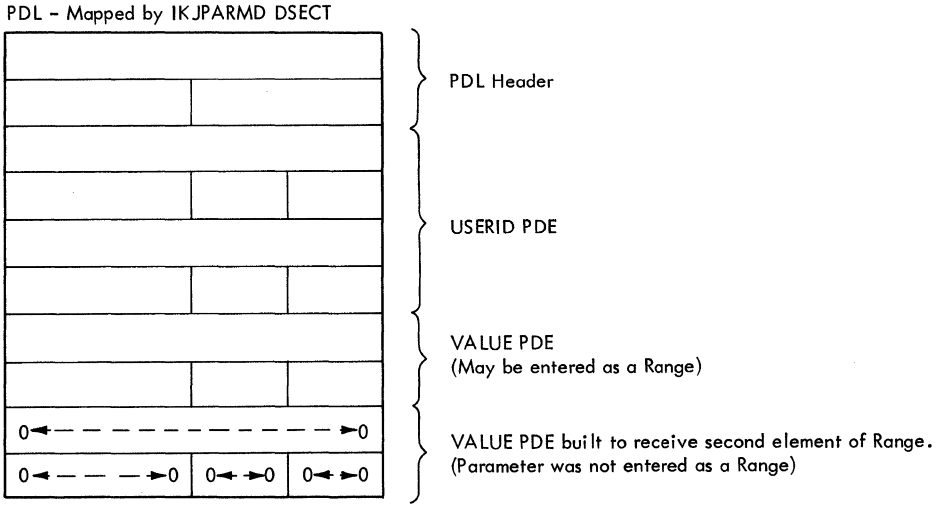

PDL - Mapped by IKJPARMD DSECT
PDL Header
USERID PDE
VALUE PDE
(May be entered as a Range)
0 ← — — — — — — — — — — — → 0
0 ← — — — → 0
0 ← → 0
0 ← → 0
VALUE PDE built to receive second element of Range.
(Parameter was not entered as a Range)

Figure 124. A PDL Showing PDEs Describing a Range

COMBINING THE LIST AND RANGE OPTIONS: If you specify both the LIST and RANGE options in a Parse macro instruction describing a positional parameter, the Parse service routine builds two identical PDEs within the PDL returned to the calling routine. Both of these PDEs are formatted according to the type of positional parameter described. These two PDEs describe the RANGE. An additional word is appended to the second PDE for the purpose of chaining any additional PDEs built to describe the LIST.

Figure 125 shows this general format.

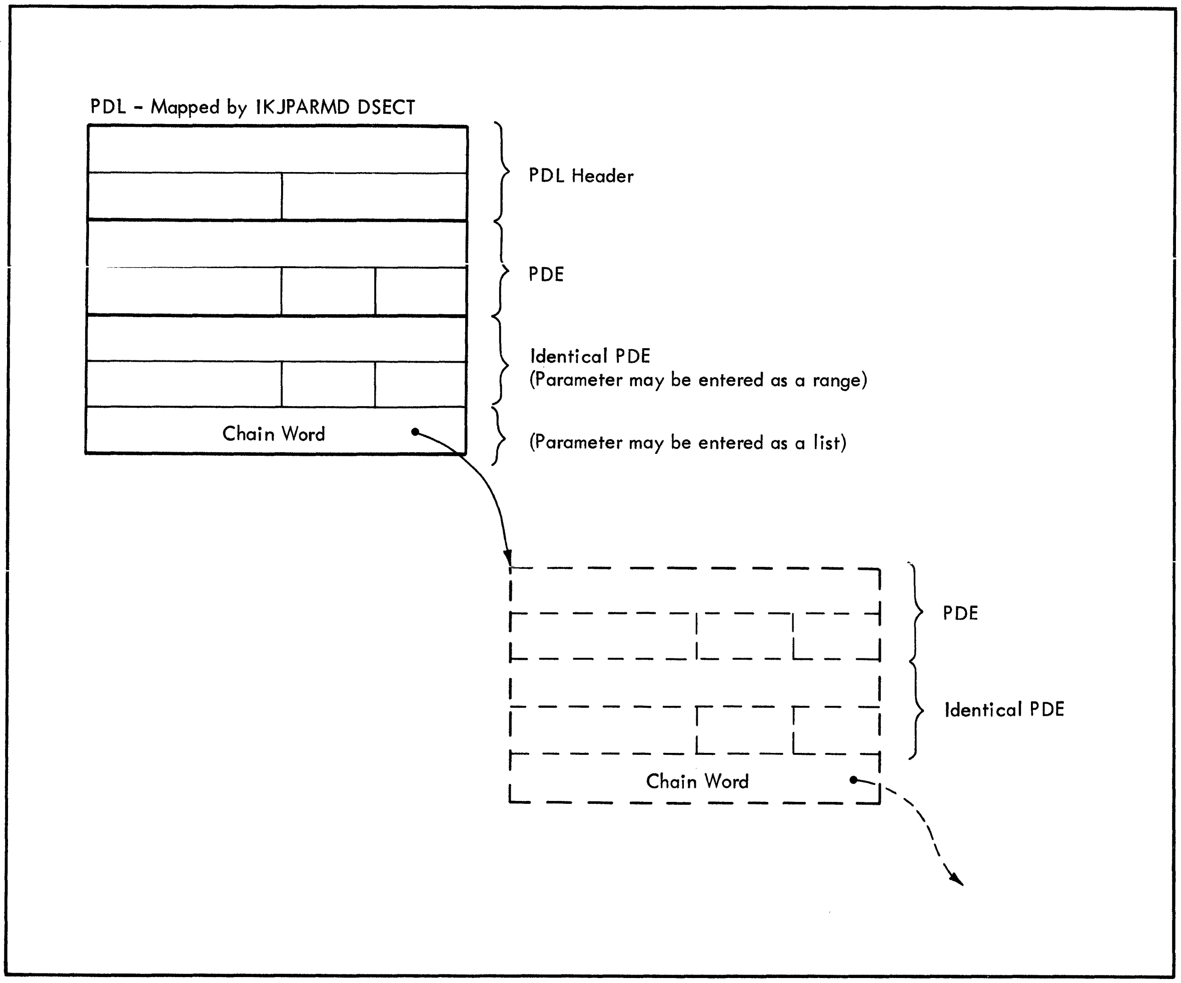

PDL - Mapped by IKJPARMD DSECT
PDL Header
PDE
Identical PDE
(Parameter may be entered as a range)
Chain Word
(Parameter may be entered as a list)
PDE
Identical PDE
Chain Word

Figure 125. PDL Showing PDEs Describing LIST and RANGE Options

If you have specified both the LIST and the RANGE options in the Parse macro instruction describing a positional parameter, the user at the terminal has the option of supplying a single parameter, a single range, a list of parameters, or a list of ranges. The construction of the PDL returned by the Parse service routine can reflect each of these conditions.

Figure 126 shows the PDL returned by the Parse service routine if the user enters a single parameter.

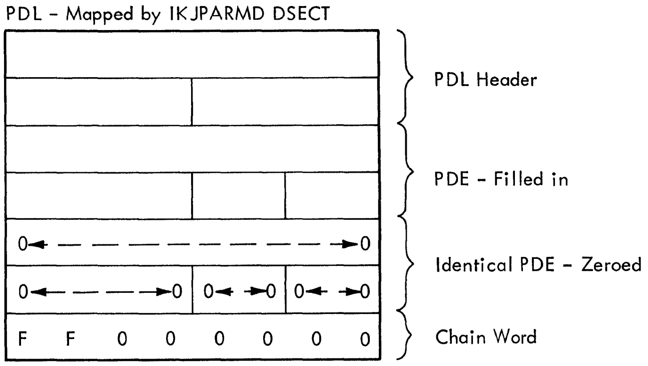


Figure 126. PDL - LIST and RANGE Acceptable, Single Parameter Entered

As Figure 126 shows, the second PDE and the chain word are both set to zero by the Parse service routine, if the LIST and RANGE options were coded in the macro instruction describing the parameter, but the user entered a single parameter.

Figure 127 shows the PDL returned by the Parse service routine if the user enters a single range of the form:

parameter:parameter

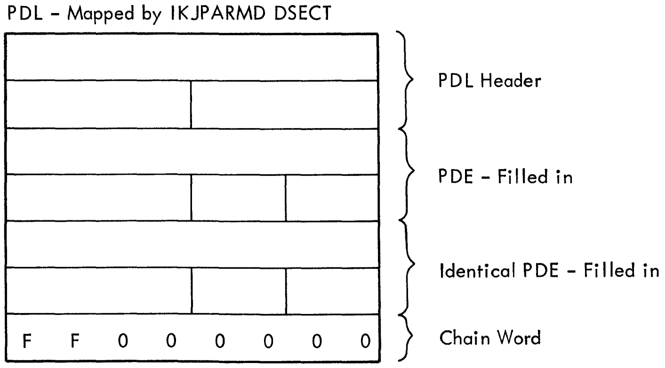


Figure 127. PDL - LIST and RANGE Acceptable, Single Range Entered

As Figure 127 shows, both PDEs are filled in to describe the single RANGE parameter entered by the user. The chain word is set to hexadecimal FF000000 to indicate that there are no elements chained onto this one; that is, the parmaeter was not entered in the form of a LIST.

Figure 128 shows the format of the PDL returned by the Parse service routine if the user enters a list of parameters in the form:

(parameter,parameter,...)

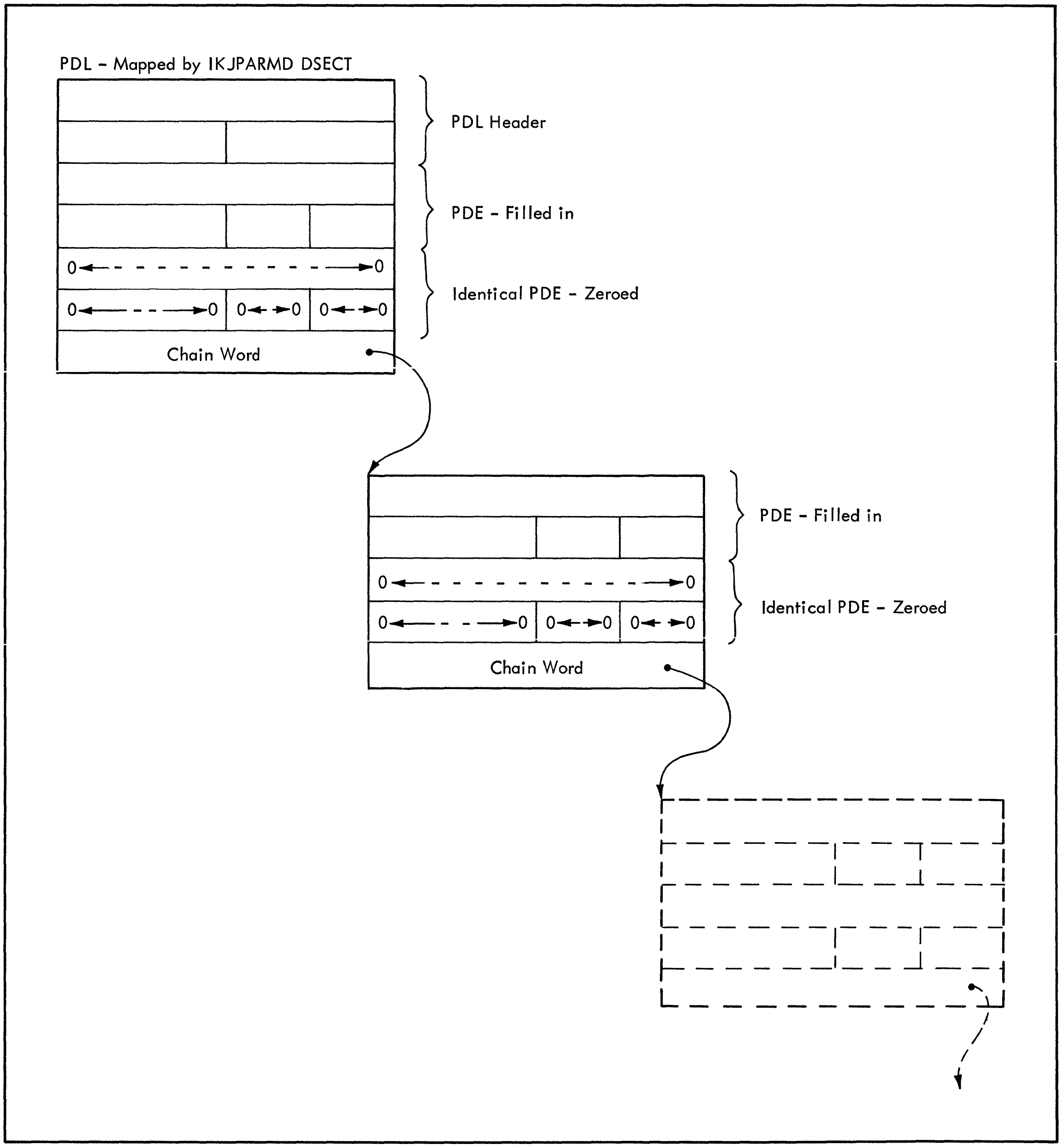


Figure 128. PDL - LIST and RANGE Acceptable, LIST Entered

As Figure 128 shows, each of the first PDEs and the chain word pointers are filled in by the Parse service routine to describe the list of parameters entered by the user. The second, identical PDEs are zeroed to indicate that the parameter was not entered in the form of a range.

The last set of PDEs on the chain will contain hexadecimal FF000000 in the chain word to indicate that there are no more PDEs on that particular chain.

The PDL created by the Parse service routine to describe a parameter entered as a list of ranges is similar to the one created to describe a list. The difference is that the second, identical PDEs are also filled in by Parse to describe the ranges entered.

Figure 129 shows the format of the PDL returned by the Parse service routine if the user enters a list of ranges in the form:

```
(parameter:parameter, parameter:parameter, ...)
```

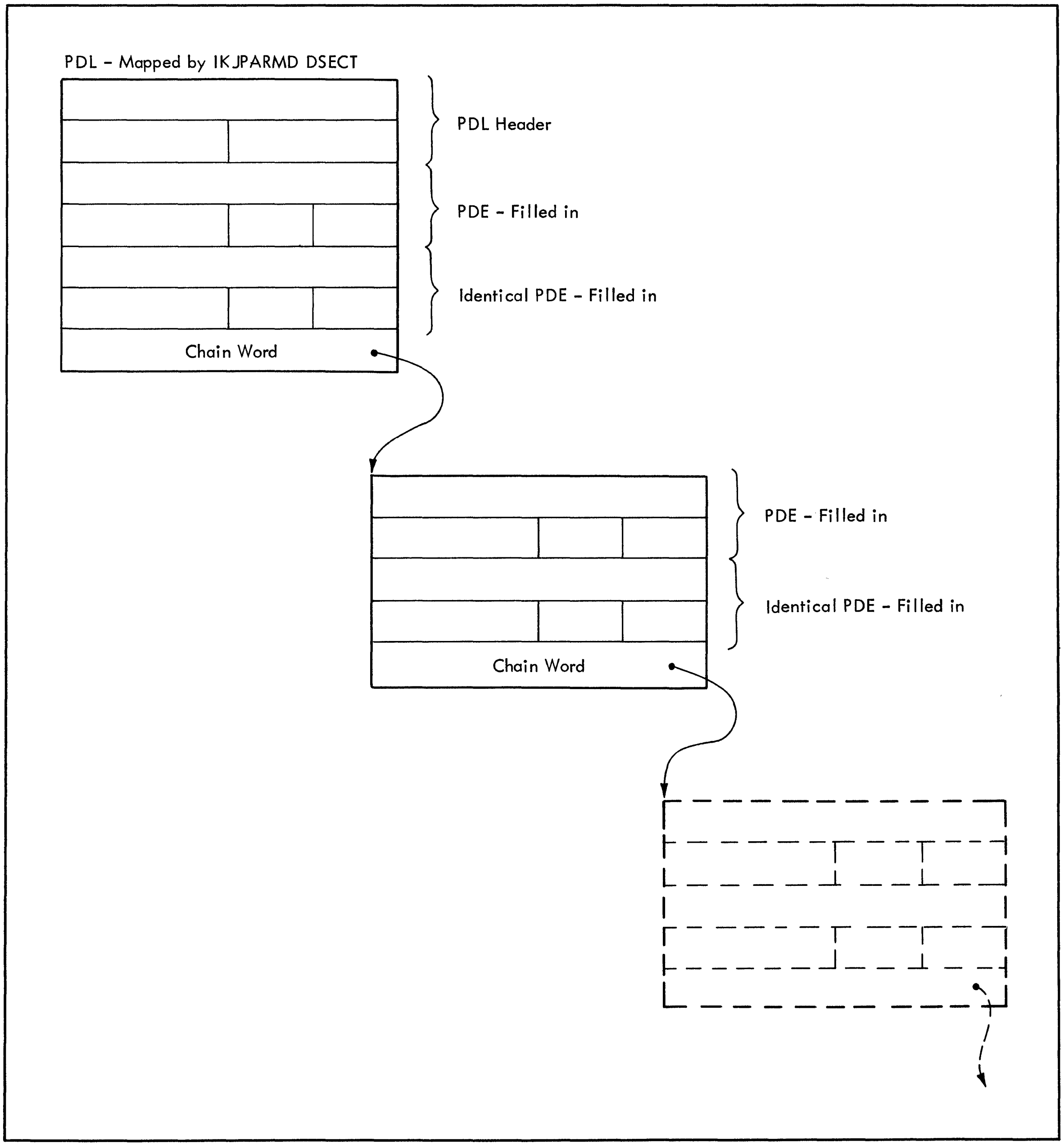

PDL - Mapped by IKJPARMD DSECT
PDL Header
PDE - Filled in
Identical PDE - Filled in
Chain Word
PDE - Filled in
Identical PDE - Filled in
Chain Word

Figure 129. PDL - LIST and RANGE Acceptable, A LIST of Ranges Entered

As Figure 129 shows, each of the PDEs and each of the second, identical PDEs are filled in by the Parse service routine to describe the ranges entered. The chain words are also filled in to point down through the list of parameters entered.

The last set of PDEs on the chain will contain hexadecimal FF000000 in the chain word to indicate that there are no more PDEs on that particular chain.

The PDE Created for a Keyword Parameter

Parse builds a halfword (2 byte) PDE to describe a KEYWORD parameter; it has the following format:

```
+0
  ┌─────────┐
  |Number   |
  |       +2|
  └─────────┘
```

Number:

You describe the possible names for a KEYWORD parameter to the Parse service routine by coding a list of IKJNAME macro instructions directly following the IKJKEYWD macro instruction. One IKJNAME macro instruction must be executed for each possible name.

The Parse service routine places the number of the IKJNAME macro instruction, that corresponds to the keyword name entered, into the PDE.

If the keyword is not entered, and you did not specify a default in the IKJKEYWD macro instruction, the Parse service routine places a zero into the PDE.

ADDITIONAL FACILITIES PROVIDED BY PARSE

The Parse service routine, in addition to determining if command parameters are syntactically correct, provides the following services which may be selected by the calling routine.

Translation to Upper Case

Positional parameters are ordinarily translated to uppercase unless the calling routine specifies ASIS in the IKJPOSIT or IKJIDENT macro instructions. The first character of a value parameter, the type-character, is always translated to uppercase, however. The string that follows the type character is translated to uppercase, unless ASIS is coded in the describing macro instructions.

Parse always translates keyword parameters to upper case.

Insertion of Default Values

Positional parameters (except delimiter and space) and keyword parameters may have default values. These default values are indicated to the Parse service routine through the DEFAULT= operand of the IKJPOSIT, IKJTERM, IKJOPER, IKJRSVWD, IKJIDENT, and IKJKEYWD macro instructions. When a positional or a keyword parameter is omitted, for which a default value has been specified, Parse inserts the default value. Parse also inserts the default value you specified if a parameter is invalid and the terminal user enters a null line in response to a prompt.

Passing Control to a Validity Checking Routine

You can provide a validity checking routine to do additional checking on a positional parameter. Each positional parameter can have a unique validity checking routine. Indicate the presence of a validity checking routine by coding the entry point address of the routine as the VALIDCK= operand in the IKJPOSIT, IKJTERM, IKJOPER or IKJIDENT macro instructions.

Parse can call validity checking routines for the following types of positional parameters:

- STRING
- VALUE
- ADDRESS
- QSTRING
- USERID
- DSNAME
- DSTHING
- CONSTANT
- VARIABLE
- STATEMENT NUMBER
- EXPRESSION
- And any non-delimiter dependent parameters.

The validity check exit is taken after the Parse service routine has determined that the parameter is syntactically correct. If a DSNAME or USERID parameter is entered with a password, Parse takes the validity check exit after first parsing both the userid or dsname and the password. If the terminal user enters a list, the validity check routine is called as each element in the list is parsed. If a range is entered, Parse calls the validity check routine only after both items of the range are parsed.

When control is passed from Parse to a validity checking routine, Parse uses standard linkage conventions. The validity check routine must save Parse's registers and restore them before returning control to Parse. The Parse service routine builds a three word parameter list and places the address of this list into register 1 before branching to a validity checking routine. This three-word parameter list has the format shown in Figure 130.

Number of Bytes	Field	Contents or Meaning
4	PDEADR	The address of the Parameter Descriptor Entry (PDE) built by parse for this syntactically correct parameter.
4	USERWORD	The address of the user work area. This is the same address you supplied to the Parse Service routine in the Parse Parameter List.
4	VALMSG	Initialized to HEX FF000000 by PARSE. A user provided validity checking routine can place the address of a second level message in this field.

Figure 130. Format of the Validity Check Parameter List

Your validity checking routines must return a code in general register 15 to the Parse service routine. These codes inform Parse of the results of the validity check and determine the action that Parse should take. Figure 131 shows the return codes, their meaning, and the action taken by the Parse service routine.

Return Code	Meaning	Action Taken by Parse
0	The parameter is valid.	No additional processing is performed on this parameter by the Parse service routine.
4	The parameter is invalid.	Parse writes an error message to the terminal and prompts for a valid parameter.
8	The parameter is invalid.	The validity checking routine has issued an error message; Parse prompts for a valid parameter.
12	The parameter is invalid; the processor cannot continue.	Parse stops all further syntax checking and returns to the calling routine.

Figure 131. Return Codes from a Validity Checking Routine

If Parse receives a return code of 4 or 8, the new data entered in response to the prompt is parsed as if it were the original data and control is again passed to the validity check routine. This cycle continues until a valid parameter is obtained.

Insertion of Keywords

Some keyword parameters may imply other keyword parameters. You may specify that other keywords are to be inserted into the parameter string when a certain keyword is entered. Use the INSERT operand of the IKJNAME macro instruction to indicate that a keyword or a list of keywords is to be inserted following the named keyword. The inserted keywords are processed as if they were entered from the terminal.

Issuing Second Level Messages

You may supply second-level messages to be chained to any prompt message issued for a positional parameter - (keyword parameters are never required). Use the HELP operand of the IKJPOSIT, IKJTERM, IKJOPER, IKJRSVWD or IKJIDENT macro instructions to supply these second level messages to the Parse service routine. You can supply up to 255 second-level messages for each positional parameter. One second-level message is issued each time a question mark is entered from the terminal. If a question mark is entered and no second-level messages were provided, or they have all been issued in response to previous question marks, the terminal user is notified that no help is available.

If a user provided validity checking routine returns the address of a second-level message to the Parse service routine, that second-level message or chain will be written out in response to question marks entered from the terminal. The original second-level chain, if one was present, is deleted.

Prompting

The Parse service routine prompts the terminal user if the command parameters found are incorrect or if required parameters are missing. It allows the terminal user to enter a missing parameter or correct an incorrect one without having to reenter the entire command. Parse prompts, and the terminal user must respond, in the following situations:

1. A userid or dsname was entered with a slash but without a password.

2. A parameter is syntactically invalid.

3. A keyword is ambiguous, that is, it is not clear to the Parse service routine which keyword of several similar ones is being entered.

4. A required positional parameter is missing. The requirement for a particular positional parameter and the prompting message to be issued if that parameter is not present, are specified to the Parse service routine through the PROMPT operand of the IKJPOSIT, IKJTERM, IKJOPER, IKJRSVWD, and IKJIDENT macro instructions. Parse puts out the prompting message supplied in the macro instruction.

5. A validity check exit indicates that a parameter is invalid.

There are a number of rules that govern the processing of responses entered from the terminal after a prompt.

1. All of the new data entered is parsed before the scan of the original command is resumed.

2. Unless otherwise stated in the command syntax definition, the new parameter is entered as it is entered in the original command. See the section on Command Parameter Syntax for exceptions to this rule.

3. In general, additional parameters may be entered along with the data prompted for. It must be kept in mind, however, that all of the new data entered is parsed before the scan of the material in the original command buffer is resumed. A problem could occur in a situation where a command is entered followed by two positional parameters and a keyword, and the first positional parameter is invalid. Parse issues a prompt for the first positional parameter. When the user at the terminal reenters that first positional parameter, it would be invalid to enter additional keywords along with it. The additional keywords would be scanned before the second positional parameter and an error condition would result when parse returned to the original command buffer and found a positional parameter.

   Keep in mind also, that if the parameter prompted for is within a subfield, only parameters valid within that subfield may be entered along with the parameter prompted for.

4. In general, a null response is acceptable only for optional parameters. However, if a null response is entered for an optional parameter that has a default, Parse inserts the default. If a prompt for a required parameter is answered by a null response from the terminal, Parse reissues the prompt message. Parse continues prompting until a correct parameter is entered. The terminal user can request termination by entering an attention.

Parse will always accept a null response to a prompt for a password, whether or not the dsname or userid parameters are required. It is the responsibility of the routine using the Parse service routine to insure that the correct password was entered if one was required, by checking the password pointed to by the PDE returned by Parse.

5. If a required parameter which may be entered in the form of a list is missing, or if it was entered as a single parameter (not as a list), and that single parameter is incorrect, Parse will not accept a list after the prompt. The user at the terminal must enter a single parameter.

   If however, the item was entered as a list but an item within the list is invalid, Parse accepts one or more parameters after the prompt. Parse considers these newly entered parameters to be part of the original list. No parameters not valid in the list may be entered from the terminal in response to this prompt.

   If the last item in a list is found to be invalid, Parse only accepts one parameter after a prompt.

6. If Parse determines that a parameter is invalid, the invalid portion of the parameter is indicated in the error message. The remainder of the parameter is not yet parsed. The user must reenter as much of the invalid parameter as was indicated in the error message. This situation often occurs if a dsname parameter or userid parameter is entered with blanks between the dsname or userid and the password. The dsname or userid may be invalid but the password is still good and will be parsed after a new dsname or userid is entered in response to the prompt.

Parse always attempts to obtain syntactically correct parameters before returning to the calling routine. However, this is not always possible. The terminal user may have requested that no prompt messages be sent to the terminal, or the command being parsed may have come from a procedure. In these cases, an error message is issued and a code is returned to the calling routine indicating that a correct command could not be obtained. Any second level messages that would ordinarily be appended to the request for new data are appended to the error message.

EXAMPLES OF USING THE PARSE SERVICE ROUTINE

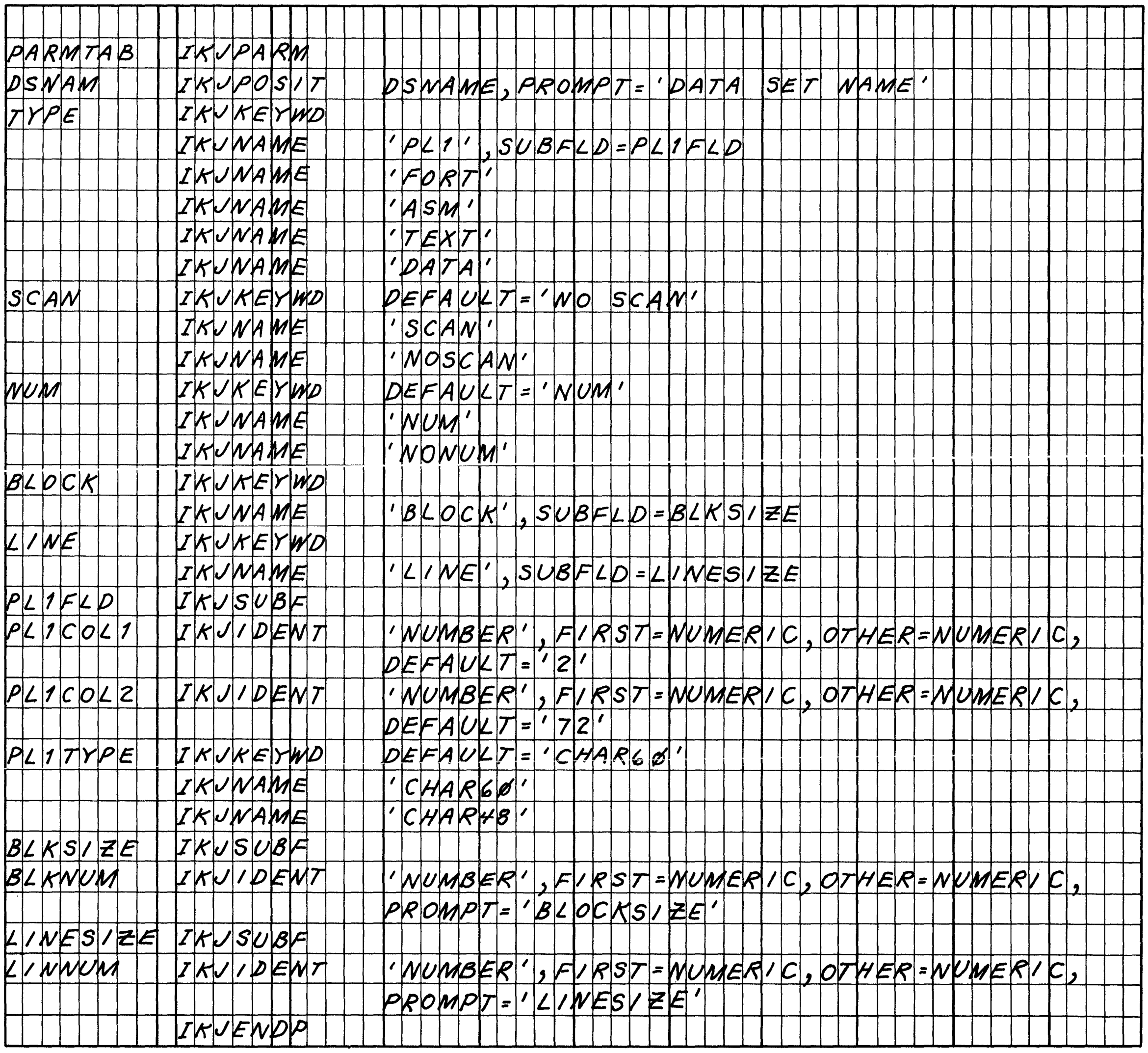
EXAMPLE 1

This example shows how the Parse macro instructions could be used within a Command processor to describe the syntax of an EDIT command to the Parse service routine.

The EDIT command we are describing to Parse has the following format:

```
 EDIT   | dsname
        | [PLI   [( [number [number]] [CHAR60] )]]
        | [          2        72      CHAR48     ]
        | [                                      ]
        | [FORT                                  ]
        | [ASM                                   ]
        | [TEXT                                  ]
        | [DATA                                  ]
        |
        | [SCAN  ]
        | [NOSCAN]
        |
        | [NUM  ]
        | [NONUM]
        |
        |  BLOCK(number)
        |
        |  LINE(number)
```

Figure 132 shows the sequence of Parse macro instructions that would describe the syntax of this EDIT command to the Parse service routine.



```
PARMTAB  IKJPARM
DSNAM    IKJPOSIT  DSNAME,PROMPT='DATA SET NAME'
TYPE     IKJKEYWD
         IKJNAME   'PL1',SUBFLD=PL1FLD
         IKJNAME   'FORT'
         IKJNAME   'ASM'
         IKJNAME   'TEXT'
         IKJNAME   'DATA'
SCAN     IKJKEYWD  DEFAULT='NO SCAN'
         IKJNAME   'SCAN'
         IKJNAME   'NOSCAN'
NUM      IKJKEYWD  DEFAULT='NUM'
         IKJNAME   'NUM'
         IKJNAME   'NONUM'
BLOCK    IKJKEYWD
         IKJNAME   'BLOCK',SUBFLD=BLKSIZE
LINE     IKJKEYWD
         IKJNAME   'LINE',SUBFLD=LINESIZE
PL1FLD   IKJSUBF
PL1COL1  IKJIDENT  'NUMBER',FIRST=NUMERIC,OTHER=NUMERIC,
                   DEFAULT='2'
PL1COL2  IKJIDENT  'NUMBER',FIRST=NUMERIC,OTHER=NUMERIC,
                   DEFAULT='72'
PL1TYPE  IKJKEYWD  DEFAULT='CHAR6Ø'
         IKJNAME   'CHAR6Ø'
         IKJNAME   'CHAR48'
BLKSIZE  IKJSUBF
BLKNUM   IKJIDENT  'NUMBER',FIRST=NUMERIC,OTHER=NUMERIC,
                   PROMPT='BLOCKSIZE'
LINESIZE IKJSUBF
LINNUM   IKJIDENT  'NUMBER',FIRST=NUMERIC,OTHER=NUMERIC,
                   PROMPT='LINESIZE'
         IKJENDP
```

Figure 132. Coding Example 1 -- Using Parse Macros to Describe Command Parameter Syntax

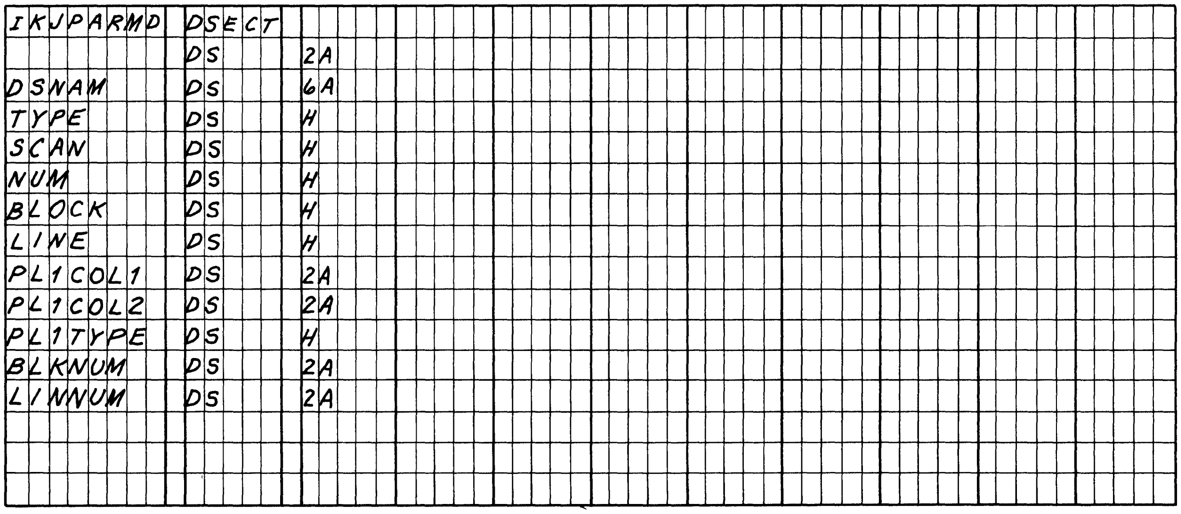
The Parse macro instructions shown in Figure 132, when executed, perform two distinct functions.

1. They build the Parameter Control List describing the syntax of the EDIT command parameters. The PCL is used by the Parse service routine during its scan of the parameters within the command buffer.

2. They create the IKJPARMD DSECT (defaulted to on the IKJPARM macro instruction) that you use to map the Parameter Descriptor List returned by the Parse service routine after it scans the parameters within the command buffer.

Your code never refers to the PCL; it is used only by the Parse service routine. Therefore, it is not shown in the example.

Figure 133 shows the IKJPARMD DSECT created by the expansion of the Parse macro instructions.



```
IKJPARMD DSECT
         DS    2A
DSNAM    DS    6A
TYPE     DS    H
SCAN     DS    H
NUM      DS    H
BLOCK    DS    H
LINE     DS    H
PL1COL1  DS    2A
PL1COL2  DS    2A
PL1TYPE  DS    H
BLKNUM   DS    2A
LINNUM   DS    2A
```

Figure 133. An IKJPARMD DSECT (Example 1)

If a terminal user entered the above described EDIT command in the form:

```
EDIT  SYSFILE/X  PL1(3)  NONUM  BLOCK   cr
```

the Parse service routine would prompt for the blocksize as follows:

```
"ENTER BLOCKSIZE"
```

The user at the terminal might respond with:

```
160
```

The Parse service routine would then complete the scan of the command parameters, build a Parameter Descriptor List (PDL), place the address of the PDL into the fullword pointed to by the fifth word of the Parse Parameter List, and return to the calling program.

The calling routine uses the address of the PDL as a base address for the IKJPARMD DSECT.

Figure 134 shows the PDL returned by the Parse service routine. The symbolic addresses within the IKJPARMD DSECT are shown to the left of the PDL at the points within the PDL to which they apply, and the meanings of the fields within the PDL are explained to the right of the PDL.

IKJPARMD DSECT	PDL		Description of Field Contents
IKJPARMD			
			PDL Header. Used only by IKJRLSA
DSNAM	Pointer to SYSFILE		Data Set Name
	7	1 0	
	0		No member name
	0	0	
	Pointer to X		Password
	1	1	
TYPE, SCAN	1	2	PL1, NOSCAN
NUM, BLOCK	2	1	NONUM, BLOCK
LINE	0	Unused	LINE not specified
PL1COL1	Pointer to 3		3 was specified
	1	1	
PL1COL2	Pointer to 72		72 is the default
	2	1	
PL1TYPE	1	Unused	CHAR60 is the default
BLKNUM	Pointer to 160		160 was prompted for
	3	1	
LINNUM	0		LINNUM not specified
	0	0	

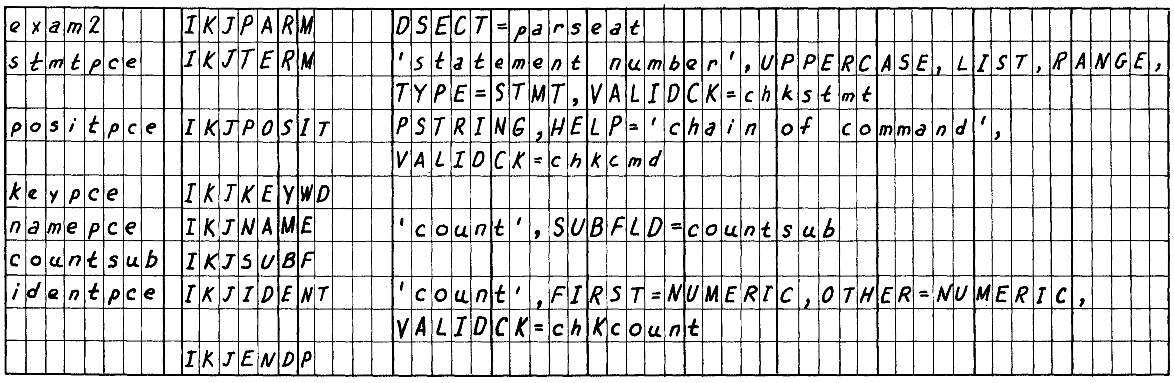
Figure 134. The IKJPARMD DSECT and the PDL (Example 1)

EXAMPLE 2

This example shows how the Parse macro instructions could be used to describe the syntax of a sample AT command that has the following syntax:

COMMAND	OPERANDS
AT	{ stmt (stmt-1,stmt-2,...) stmt-3:stmt-4 } (cmd,chain) COUNT(integer)

The following figure shows the sequence of Parse macro instructions that describe this sample AT command to the Parse service routine.



```
exam2    IKJPARM    DSECT=parseat
stmtpce  IKJTERM    'statement number',UPPERCASE,LIST,RANGE,
                    TYPE=STMT,VALIDCK=chkstmt
positpce IKJPOSIT   PSTRING,HELP='chain of command',
                    VALIDCK=chkcmd
keypce   IKJKEYWD
namepce  IKJNAME    'count',SUBFLD=countsub
countsub IKJSUBF
identpce IKJIDENT   'count',FIRST=NUMERIC,OTHER=NUMERIC,
                    VALIDCK=chkcount
         IKJENDP
```

Figure 135. Coding Example 2 -- Using Parse Macros to Describe Parameter Syntax

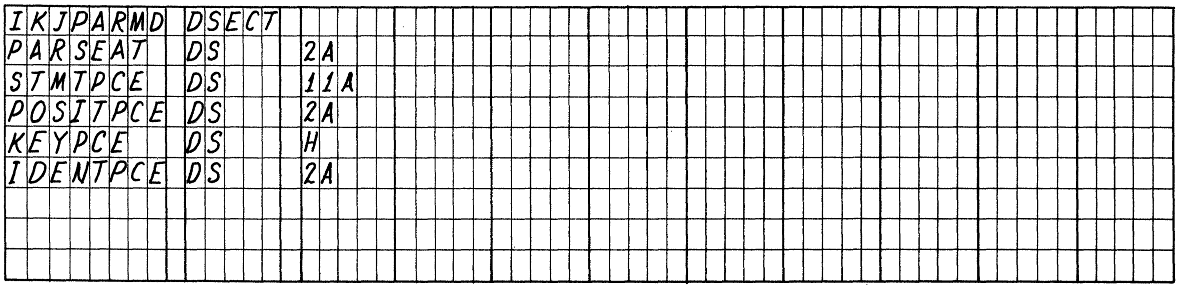
The Parse macro instructions shown in Figure 135, when executed perform two distinct functions.

1. They build the Parameter Control List describing the syntax of the command parameters. This PCL is then used by the Parse service routine during its scan of the parameters in the command buffer.

2. They create the IKJPARMD DSECT that you use to map the Parameter Descriptor List. The PDL is returned by Parse after it scans the parameters in the command buffer.

Note: Your code never refers to the PCL; it is used only by the Parse service routine. Therefore, the Parameter Control List is not shown in the example.

Figure 136 shows the IKJPARMD DSECT created by the expansion of the Parse macro instructions.



```
IKJPARMD DSECT
PARSEAT  DS    2A
STMTPCE  DS    11A
POSITPCE DS    2A
KEYPCE   DS    H
IDENTPCE DS    2A
```

Figure 136. An IKJPARMD DSECT (Example 2)

In this example, if the terminal user entered the above described command incorrectly like this:

```
AT  200/3  (list all) COUNT(a)
```

the Parse routine would prompt the terminal user with the message:

```
INVALID STATEMENT NUMBER, 200/3
REENTER
```

The user might respond with:

```
200.3
```

the Parse routine would then prompt the user with:

```
INVALID COUNT, a
REENTER
```

The user might respond with:

```
3
```

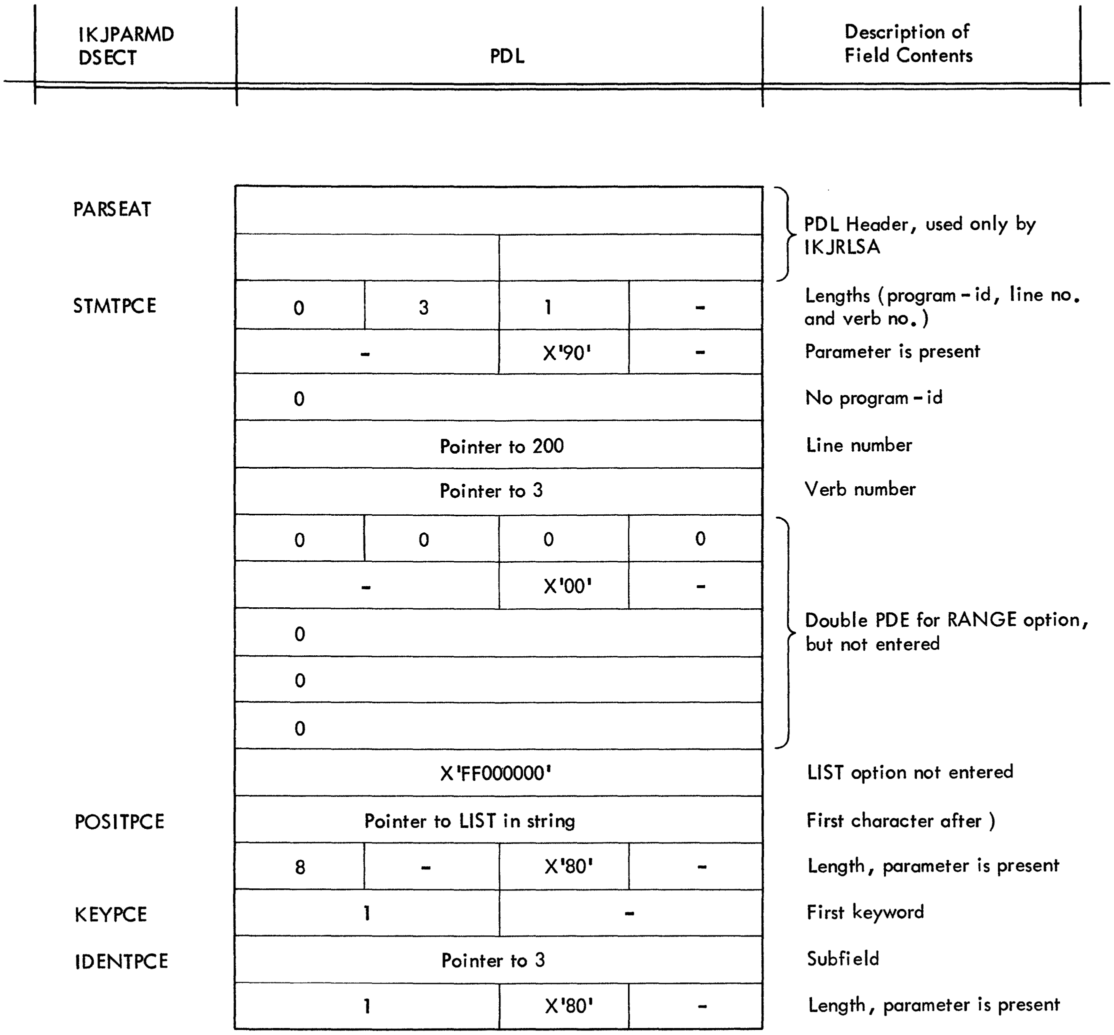
This sequence resulted in the syntactically correct command of:

```
AT  200.3  (list all) COUNT(3)
```

The Parse routine would then build a Parameter Descriptor List (PDL) and place the address of the PDL into the fullword pointed to by the fifth word of the Parse Parameter List.

Parse then returns to the caller and the caller uses the address of the PDL as a base address for the IKJPARMD DSECT.

Figure 137 shows the PDL returned by the Parse routine. The symbolic addresses of the IKJPARMD DSECT are shown to the left of the PDL at the points within the PDL to which they apply. A description of the fields within the PDL is shown on the right.



IKJPARMD DSECT	PDL				Description of Field Contents
PARSEAT					PDL Header, used only by IKJRLSA
STMTPCE	0	3	1	-	Lengths (program - id, line no. and verb no.)
	-		X'90'	-	Parameter is present
	0				No program - id
	Pointer to 200				Line number
	Pointer to 3				Verb number
	0	0	0	0	Double PDE for RANGE option, but not entered
	-		X'00'	-	
	0				
	0				
	0				
	X'FF000000'				LIST option not entered
POSITPCE	Pointer to LIST in string				First character after)
	8	-	X'80'	-	Length, parameter is present
KEYPCE	1		-		First keyword
IDENTPCE	Pointer to 3				Subfield
	1		X'80'	-	Length, parameter is present

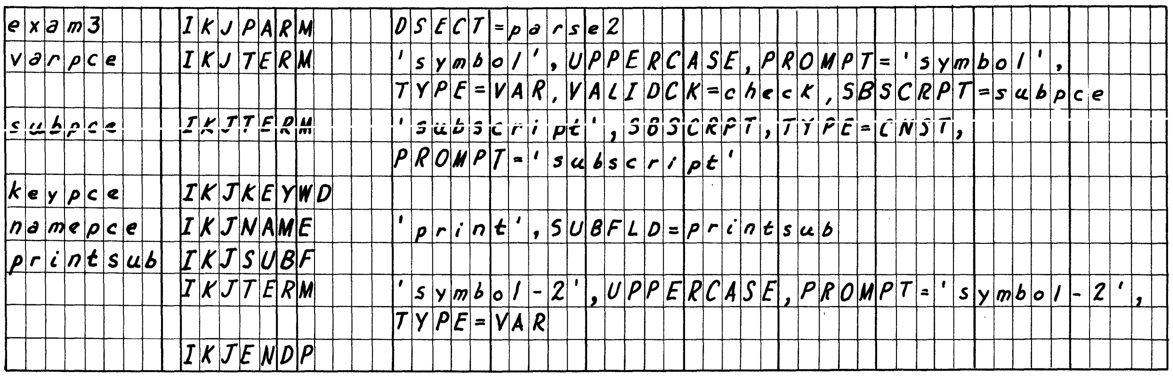
Figure 137. The IKJPARMD DSECT and the PDL (Example 2)

EXAMPLE 3

This example shows how the Parse macro instructions could be used to describe the syntax of a sample LIST command that has the following syntax:

COMMAND	OPERANDS
LIST	symbol PRINT(symbol)

The following figure shows the sequence of Parse macro instructions that describe this sample LIST command to the Parse service routine.



```
exam3    IKJPARM    DSECT=parse2
varpce   IKJTERM    'symbol',UPPERCASE,PROMPT='symbol',
                    TYPE=VAR,VALIDCK=check,SBSCRPT=subpce
subpce   IKJTERM    'subscript',SBSCRPT,TYPE=CNST,
                    PROMPT='subscript'
keypce   IKJKEYWD
namepce  IKJNAME    'print',SUBFLD=printsub
printsub IKJSUBF
         IKJTERM    'symbol-2',UPPERCASE,PROMPT='symbol-2',
                    TYPE=VAR
         IKJENDP
```

Figure 138. Coding Example 3 -- Using Parse Macros to Describe Parameter Syntax

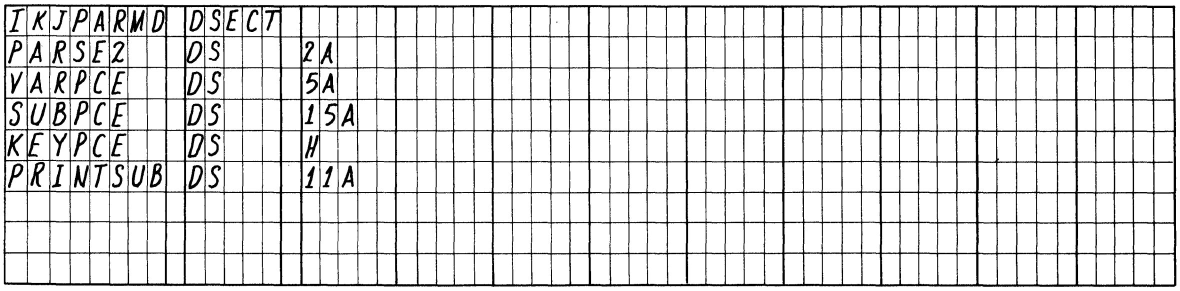
The Parse macro instructions shown in Figure 138, when executed perform two distinct functions.

1. They build the Parameter Control List describing the syntax of the command parameters. This PCL is then used by the Parse service routine during its scan of the parameters in the command buffer.

2. They create the IKJPARMD DSECT that you use to map the Parameter Descriptor List. The PDL is returned by Parse after it scans the parameters in the command buffer.

Note: Your code never references the PCL; it is used only by the Parse service routine. Therefore, the Parameter Control List for the example is not shown.

Figure 139 shows the IKJPARMD DSECT created by the expansion of the Parse macro instructions.



```
IKJPARMD DSECT
PARSE2   DS    2A
VARPCE   DS    5A
SUBPCE   DS    15A
KEYPCE   DS    H
PRINTSUB DS    11A
```

Figure 139. An IKJPARMD DSECT (Example 3)

In this example, if the terminal user entered the above described command incorrectly like this:

```
LIST a of 1 in 3(1) PRINT(d)
```

the Parse routine would prompt the terminal user with:

```
INVALID SYMBOL, a...1 in 3(1)
REENTER
```

The user might respond:

```
a of b in 3(1)
```

the Parse routine would then prompt with:

```
INVALID SYMBOL, a...3(1)
REENTER
```

The user might respond with:

```
a of b in c(1)
```

This sequence resulted in the syntactically correct command of:

```
LIST a of b in c(1) PRINT(d)
```

The Parse routine would then build a Parameter Descriptor List (PDL) and place the address of the PDL into the fullword pointed to by the fifth word of the Parse Parameter List.

Parse then returns to the caller and the caller uses the address of the PDL as a base address for the IKJPARMD DSECT.

Figure 140 shows the PDL returned by the Parse routine. The symbolic addresses of the IKJPARMD DSECT are shown to the left of the PDL at the points within the PDL to which they apply. A description of the fields within the PDL is shown on the right.

IKJPARMD DSECT	PDL				Description of Field Contents
PARSE2					PDL Header - Used only by IKJRLSA
VARPCE	Pointer to a				Data - name
	1	-	X'A0'	-	Length, parameter is present
	Pointer to first qualifier				Qualifier
	0				No program - id
	0	2	1	-	Length, qualifiers, subscript
SUBPCE	1	0	-	-	Length
	0		X'C800'		Flags, CNST
	Pointer to 1				Subscript
	0				No exponent
	0				No decimal point
	0	0	0	-	2nd element in subscript - (Not entered)
	0		X'0000'		
	0				
	0				
	0				
	0	0	0	-	3rd element in subscript - (Not entered)
	0		X'0000'		
	0				
	0				
	0				
KEYPCE	1		-		First keyword
PRINTSUB	Pointer to d				Data - name
	1	-	X'A0'	-	Length, parameter, variable
	0				No qualifiers
	0				No program - id
	0	0	0	-	No length, qualifier, or subscript
(First Qualifier) *	Pointer to b				First qualifier
	1	-	X'00'	-	Length, parameter, variable
	Pointer to next qualifier				Next qualifier
(Next Qualifier) *	Pointer to c				Second qualifier
	1	-	X'00'	-	Length, parameter, variable
	X'FF000000'				End of qualifiers

*Note: May not be contiguous in storage at this point.

Figure 140. The IKJPARMD DSECT and the PDL (Example 3)

EXAMPLE 4

This example shows how the Parse macro instructions could be used to describe the syntax of a sample WHEN command that has the following syntax:

COMMAND	OPERANDS
WHEN	{ addr expression } (subcommand chain)

The following figure shows the sequence of Parse macro instructions that describe this sample WHEN command to the Parse service routine.

```
exam4     IKJPARM   DSECT=parse3
oper      IKJOPER   'expresion',OPERND1=symbol1,
                    OPERND2=symbol2,RSVWD=operator,
                    CHAIN=addr1,PROMPT='term',VALIDCK=check
symbol1   IKJTERM   'symbol1',UPPERCASE,TYPE=VAR,
                    PROMPT='symbol2'
operator  IKJRSVWD  'operator',PROMPT='operator'
          IKJNAME   'eq'
          IKJNAME   'neq'
symbol2   IKJTERM   'symbol2',TYPE=VAR
addr1     IKJTERM   'address',TYPE=VAR,VALIDCK=check1
lastone   IKJPOSIT  PSTRING,VALIDCK=CHECK2
          IKJENDP
```

Figure 141. Coding Example 4 -- Using Parse Macros to Describe Parameter Syntax

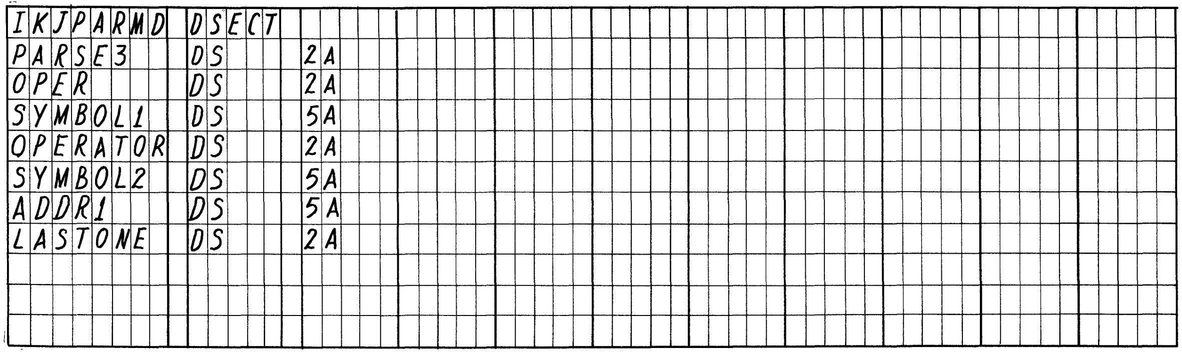
The Parse macro instructions shown in Figure 141, when executed perform two distinct functions.

1. They build the Parameter Control List describing the syntax of the command parameters. This PCL is then used by the Parse service routine during its scan of the parameters in the command buffer.

2. They create the IKJPARMD DSECT that you to map the Parameter Descriptor List. The PDL is returned by Parse after it scans the parameters in the command buffer.

Note: Your code never references the PCL; it is used only by the Parse service routine. Therefore, the Parameter Control List for this example is not shown.

Figure 142 shows the IKJPARMD DSECT created by the expansion of the Parse macro instructions.



```
IKJPARMD DSECT
PARSE3   DS    2A
OPER     DS    2A
SYMBOL1  DS    5A
OPERATOR DS    2A
SYMBOL2  DS    5A
ADDR1    DS    5A
LASTONE  DS    2A
```

Figure 142. An IKJPARMD DSECT (Example 4)

In this example, if the terminal user entered the above described command incorrectly like this:

```
WHEN  (a)  (LIST b)
```

the Parse routine would prompt the terminal user with:

```
ENTER OPERATOR
```

The user might then respond:

```
eq
```

the Parse routine would then prompt with:

```
INVALID EXPRESSION, (a eq)
REENTER
```

The user might respond then with:

```
(a eq b)
```

This sequence resulted in a syntactically correct command of:

```
WHEN  (a eq b)  (LIST b)
```

The Parse routine would then build a Parameter Descriptor List (PDL) and place the address of the PDL into the fullword pointed to by the fifth word of the Parse Parameter List.

Parse then returns to the caller and the caller uses the address of the PDL as a base address for the IKJPARMD DSECT.

Figure 143 shows the PDL returned by the Parse routine. The symbolic addresses of the IKJPARMD DSECT are shown to the left of the PDL at the points within the PDL to which they apply. A description of the fields within the PDL is shown on the right.

IKJPARMD DSECT	PDL				Description of Field Contents
PARSE3					PDL Header - Used only by IKJRLSA
OPER	-				
	-		X'80'	-	Parameter is present
SYMBOL1	Pointer to a				First operand
	1	-	X'A0'	-	Length, parameter is present
	X'FF000000'				No qualifiers
	0				No program - id
	0	0	0	-	No lengths for program - id, subscripts, or qualifiers
OPERATOR	-		1		First keyword entered
	-		X'80'	-	Parameter is present
SYMBOL2	Pointer to b				Second operand
	1	-	X'A0'	-	Length, parameter, variable
	X'FF000000'				No qualifiers
	0				No program - id
	0	0	0	-	No lengths for program - id, subscripts or qualifiers
ADDR1	0				(Address - Not entered)
	0	-	X'00'	-	
	0				
	0				
	0	0	0	-	
LASTONE	Pointer to LIST				Subcommand
	6		X'80'	-	Length, parameter is present

Figure 143. The IKJPARMD DSECT and the PDL (Example 4)

RETURN CODES FROM THE PARSE SERVICE ROUTINE

When it returns to the program that invoked it, the Parse service routine provides one of the following return codes in general register 15:

CODE decimal	MEANING
0	Parse completed successfully.
4	The command parameters were incomplete and Parse was unable to prompt.
8	Parse did not complete. An attention interruption occurred during Parse processing. The communications ECB is posted.
12	The Parse Parameter Block contains invalid information.
16	Parse issued a GETMAIN and no space was available.
20	A validity checking routine requested termination.
24	Conflicting parameters were found on the IKJTERM, IKJOPER or IKJRSVWD macro instruction.

Testing a Newly Written Program -- The TEST Command

The TEST command permits a user at a terminal to test an assembly language program, including a user written TMP, Command Processor, or applications program.

You test a program by issuing the TEST command and the various TEST subcommands that perform the following basic functions:

- Execute the program under test from its starting address or from any address within the program. The GO, CALL, or RUN subcommand does this. An example is GO ERRTN which starts the program being tested at symbolic location ERRTN.

- Display selected areas of the program as it currently appears in main storage, or display the contents of any of the registers. The LIST subcommand does this. An example is LIST 4R, which displays the contents of general register four.

- Interrupt the program under test at a specified location or locations. Once you have interrupted the program you can display areas of the program or any of the registers, or you can issue other subcommands of TEST to be executed before returning control to the program under test. You can specify the subcommands you want executed at any of these break points by issuing the AT subcommand before execution of the program. In this case the subcommands named on the AT subcommand are executed automatically without your having to enter them when the program is interrupted.

- Change the contents of specified program locations in main storage or the contents of specific registers. You do this with the TEST "assignment" function. An example is ERRTN=X'18D1'. This places the hexadecimal value 18D1 at symbolic location ERRTN.

In addition to these basic debugging functions, the TEST command processor provides other facilities. Examples are the listing of data extent blocks (DEBs), data control blocks (DCBs), task control blocks (TCBs), program status words (PSWs), and providing a main storage map of the program being tested. A complete list of the TEST subcommands and a short description of their functions is provided in Figure 144.

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

Figure 144. The TEST Subcommands

WHEN YOU WOULD USE TEST

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

1. You want to TEST a program currently active in the system.

2. You want to TEST a program not currently being executed.

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

If a program has begun to abnormally terminate, you receive a diagnostic message from the Terminal Monitor Program and then a READY message. The TMP is in effect asking, "Do you want to terminate your program or test it?" If you respond with anything but TEST, your program is abnormally terminated by the ABEND routine. If, however, you issue the TEST command (no program name should be supplied), the TEST command processor is given control, and you can use the TEST subcommands to debug the defective program.

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

Note that in the case of both the ABEND or the attention interruption, you do not enter a program name following the TEST command. If you enter the TEST command followed by the name of the currently active program, you lose the current in-storage copy of the program and TEST loads a new copy.

The second use of the TEST command processor, testing a program not currently being executed, requires that you enter a program name along with the TEST command. When the Terminal Monitor Program issues a READY message to request a command, enter the command, TEST program name. (There are other optional operands of the TEST command but they are not necessary for this example.) The TEST command processor is given control and it loads a copy of the named program. The program can be a newly written TMP, CP, or applications program.

Programs to be tested in this manner must be linkage edited members of partitioned data sets, or object modules in sequential or partitioned data sets, loadable by the Operating System Loader.

While the program is under the control of TEST, you can step through the program, investigate or alter the environment at any time, change instructions or register contents, force entry into various subroutines, and perform other debugging operations online and immediately.

It is this second use of the TEST command processor, especially the debugging of newly written code, that this section discusses.

This section is not intended to be a complete discussion of the TEST command processor. For additional discussion of the TEST command and its operands, see TSO Command Language Reference and Terminal User's Guide.

ADDRESSING RESTRICTIONS

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

External symbols, such as CSECT names, can be available for both object modules and load modules. Object modules require that the OS Loader had enough main storage to build in-core CESD entries. Load modules must have been processed by the Linkage Editor with the TEST parameter specified, or must have been fetched to main storage by the TEST command or its LOAD subcommand.

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

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

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

EXECUTING A PROGRAM UNDER THE CONTROL OF TEST

Any program, if it is a linkage edited member of a partitioned data set or an object module in a sequential or partitioned data set, can be executed under the control of the TEST command processor.

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

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

If the program to be tested is a command processor, include the keyword CP (the default is NOCP). The test routine creates a Command Processor Parameter List, and places its address into register 1 before loading the program.

Figure 145 shows the sequence of operations leading up to and following the issuance of the TEST command.

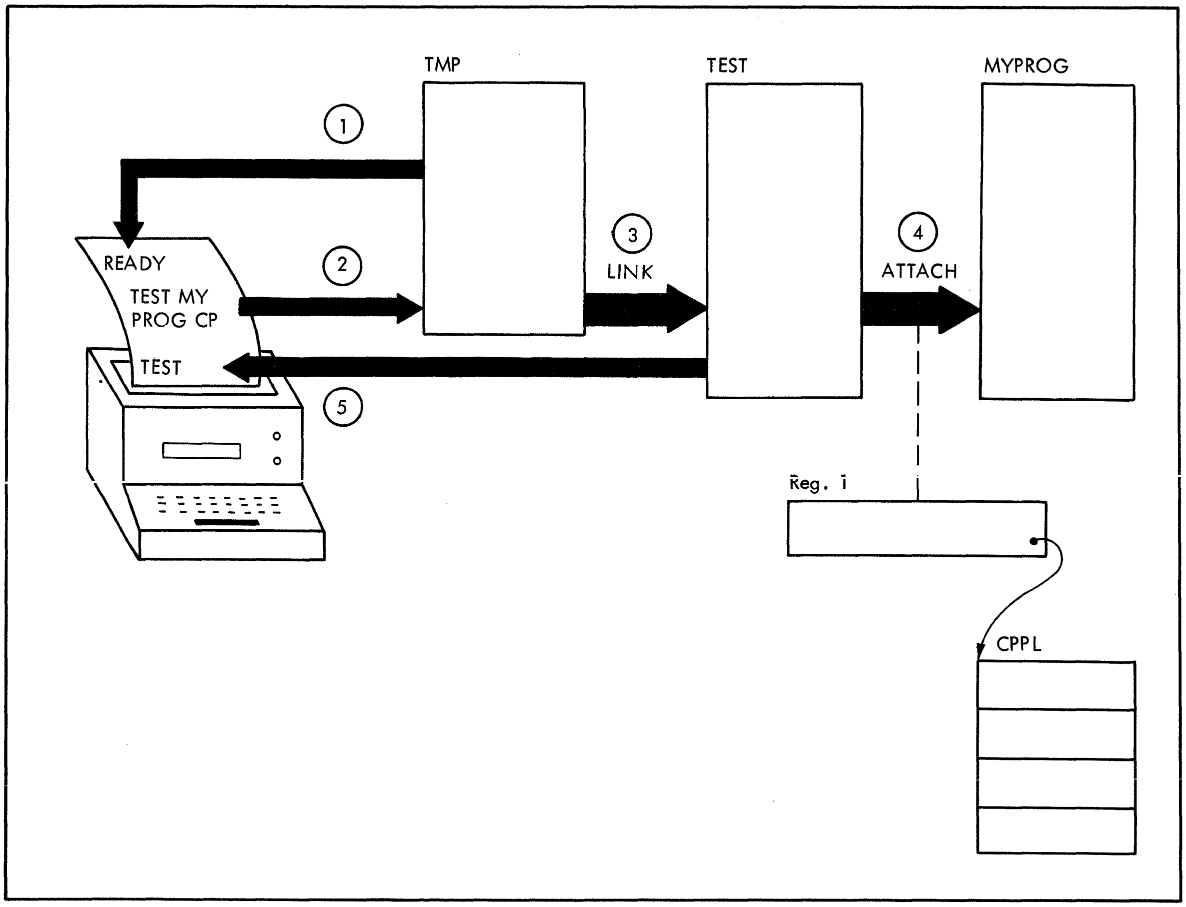


Figure 145. Issuing the TEST Command

1. The Terminal Monitor Program issues a READY message to the terminal to indicate that the user should enter a command.

2. The user at the terminal answers with the command:

   TEST MYPROG CP

3. The TMP uses the Command Scan service routine to determine that a valid command has been entered, and links to the TEST command processor.

4. The TEST command processor, using the PARSE service routine, determines that the user wants a Command Processor Parameter List built and passed to the load module (LOAD is the default) MYPROG. TEST builds the CPPL, places its address into register one, and attaches the TEST loader which XCTLs to MYPROG.

5. The TEST command processor informs the user at the terminal that it is ready to accept subcommands. TEST does this by writing the message TEST at the terminal.

From this point on, the user can use any of the facilities provided by the TEST subcommands to test his program.

ESTABLISHING AND REMOVING BREAKPOINTS WITHIN A PROGRAM:

Use the AT subcommand to establish breakpoints within the program being tested. Then issue the GO subcommand to begin execution of the program. To begin executing a newly loaded program, merely enter the subcommand GO - no address is required. When the breakpoints are encountered, as the program is being executed, processing is temporarily halted, and the message, AT address, is written to the terminal. You can then examine the executing program, its registers, and data areas to see that it has been executing properly.

There are two methods of accomplishing this.

1. You can specify a list of subcommands when you issue the AT subcommand. When a breakpoint is encountered, the TEST command processor issues each of the specified subcommands as if it had been entered from the terminal at that time. The subcommands execute and display the results of their execution at the terminal. If you specify GO as the last subcommand, control is automatically returned to the program under TEST at the point of interruption. If you do not specify GO as the last subcommand in the list, control is returned to you, at the terminal, after the last subcommand is executed. If you determine from the information displayed by the subcommands, that your program has executed properly up to that breakpoint, issue the GO subcommand. Your program resumes execution at the point of interruption and continues execution until another breakpoint, or the end of the program, is reached.

2. If you do not specify a list of subcommands when you issue the AT subcommand, the TEST command processors returns control to you at the terminal each time a breakpoint is encountered. You can then check on your program's execution by entering the TEST subcommands directly from the terminal.

Issue the OFF subcommand with no address operand to remove all breakpoints previously established. Issue the OFF subcommand followed by an address, a list of addresses, or a range of addresses to remove a single breakpoint, several breakpoints, or all breakpoints occurring within the range of addresses.

DISPLAYING SELECTED AREAS OF STORAGE

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

1. LIST
2. LISTMAP
3. LISTTCB
4. LISTDEB
5. LISTDCB
6. LISTPSW

LIST: Use the LIST subcommand to display areas of storage or the contents of registers. The address required as an operand of the LIST subcommand can be one address, a list of addresses, or a range of addresses. The address may be specified as a symbolic address if a symbol table exists and contains the requested symbolic address. If no symbol table exists (the program was not linkage edited or did not save a symbol table), you can use the EQUATE subcommand to create a symbolic address for any location within the program, or you can specify the address as a relative address, an absolute address, or as a register containing an address.

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

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

LISTMAP: Use the LISTMAP subcommand to display at your terminal a map of all storage assigned to the program under test. Some of the information displayed after issuance of the LISTMAP subcommand is:

- Region size.
- Task Control Block address.
- Program name, length, and location in storage.
- Active Request Blocks, RB types, and the names of the programs associated with each of the RBs.

LISTTCB: Use the LISTTCB subcommand to display the entire Task Control Block of the program under test, or any fields of that TCB. The information displayed is formatted, and each field is identified according to the field names contained in the publication System Control Blocks.

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

LISTDEB: Use the LISTDEB subcommand to display the Basic section and any direct access sections of any valid Data Extent Block (DEB), or any fields of that DEB. The information displayed is formatted according to the field names of the Data Extent Block as contained in the System Control Blocks publication.

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

LISTDCB: Use the LISTDCB subcommand to display the contents of a Data Control Block (DCB). The information displayed is formatted, and each field is identified according to the field names contained in the System Control Blocks publication.

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

LISTPSW: Use the LISTPSW subcommand to display the current Program Status Word or any of the PSWs at your terminal. If you issue the subcommand LISTPSW with no address following the subcommand, the current PSW is displayed at your terminal. If you want to display any of the other PSWs at your terminal, supply the address of the PSW you want to see as an operand of the LISTPSW subcommand. A list of the permanent in-storage locations of all PSWs can be found in the Principles of Operation publication.

The PSW is displayed formatted by field, i.e., system mask, key, AMWP, interruption code, ILD, CC, program mask, and instruction address.

CHANGING INSTRUCTIONS, DATA AREAS, OR REGISTER CONTENTS

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

Simply enter the address at which you want the new data entered, a code indicating the data type, and the new data you want entered at that address. The address must conform to the address restrictions already discussed. The new data must be contained within single quotes. The data type codes can be found in the publication TSO Command Language Reference.

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

Issue the TEST subcommand GETMAIN to obtain a work area in which to build your replacement segment of code. The GETMAIN subcommand writes out the address of the beginning of the storage area it obtained for you. Use the Assignment Subcommand of the TEST command to place a branch to the new area at the address in your module that begins the code you want to replace. Use the Assignment or COPY Subcommand to build your code segment in the newly obtained area. As the last instruction in your newly written code, place a branch back to the point within your module at which you want processing to resume. You can then use the GO subcommand to restart your program at some point before the branch. Your program will execute through the branch instruction and into the newly written code. If the new code works, you will execute the new instructions and branch back into your original code. Later, you can use the LIST subcommand to display the newly written code in a form useful to you, enter it into your program with the TSO EDIT command, and reassemble your now executable module.

FORCING EXECUTION OF PROGRAM SUBROUTINES

Certain paths through some programs are difficult to test because the combination of events leading to that path is difficult to produce.

One example of this problem is processing after return codes. Your module might respond differently according to the codes returned to it by some other module or some other, not yet written, section of code. You can use the AT subcommand to insert a breakpoint in your program at the point where it passes control to the not yet existing code; the assignment function of TEST to set register 15 to the desired return code; and the GO subcommand to begin execution of your program at the point where control would have been returned. Using this sequence of TEST subcommands, you can test your module's response to each possible return code.

USING TEST AFTER A PROGRAM ABEND

If a program running under TSO begins to ABEND, a diagnostic message containing the ABEND code is written to the terminal, ABEND processing is halted, and control is returned to either the TMP or TEST. If the program was running under the control of the TEST command processor, control is returned to TEST and you can immediately begin to use the TEST subcommands to determine the cause of the error. If the program was not running under TEST, control is returned to the Terminal Monitor Program. You can then enter the command TEST (no program name should be entered), to place the abnormally terminating program under control of the TEST command processor.

Use the ABEND code to determine the type of interruption that occurred. Issue the WHERE subcommand to determine where the interruption occurred.

The WHERE subcommand is especially helpful. If you enter the WHERE subcommand, the current instruction address is displayed at the terminal. If you then enter WHERE followed by that instruction address, WHERE responds by printing out the program name, the CSECT name, the offset of the current instruction address within the CSECT, and the address of the abnormally terminating task's TCB.

The instruction address, and the information returned by the WHERE subcommand pinpoint the point of error.

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

Determining Data Set Information

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

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

These control blocks contain the addresses of other control blocks useful in the debugging process. See System Control Blocks publication.

Appendix A: TSO Control Blocks

This appendix contains those control blocks frequently referenced by a programmer writing a Terminal Monitor Program or a command processor. They are:

1. The Environment Control Table (ECT)

2. The Protected Step Control Block (PSCB)

3. The Time Sharing Job Block (TJB)

4. The User Profile Table (UPT)

These control blocks are shown exactly as they appear in the System Control Blocks publication.

For each field the following information is provided: Offset, bytes and alignment, field name and field description.

The "offset" column contains the decimal and hexadecimal displacement of the beginning of the field from the start of the data area.

"Bytes and alignment" indicate the length of the field in bytes and the position of the beginning of the field based on a fullword boundary For example, . . 6 indicates that the field is six bytes in length and the field begins on the third byte of a fullword.

If the field is composed of bits, each bit in a byte is shown in the "bytes and alignment" column. For example,

```
. . 1

1... ....
.1.. 1...
..xx .xxx
```

indicates that the third byte of a fullword is composed of significant bit settings. The "field description" defines the bits when they are set as indicated. Bits indicated by an "x" are reserved for future use. When no settings are indicated (.... ....), the field has the definition provided in "field description" only when the byte is equal to zero.

The name of the field or bit is found in the "field name" column, and the meaning of the field is found in the "field description" column.

Environment Control Table

The environment control table (ECT) is a 32-byte data area constructed by the Terminal Monitor Program (TMP). It contains information about the user's environment in the foreground region. This data area resides in subpool 1 and is updated by the command processors. It is used by the command processors and the TMP. Its address is in the CPPL.

Offset Dec	Offset Hex	Bytes and Alignment	Field Name	Field Description
0	0	1	ECTRCDF	
		1...		Indicates that the command processor has abnormally terminated.
		.xxx xxxx		(Reserved bits)
1	1	. 3	ECTRTCD	The return code from the last command processor. If ECTRCDF is set, this field contains the ABEND code.
4	4	4	ECTIOWA	Address of the I/O work area.
8	8	1	ECTMSGF	
		1...		Indicates that the second level messages are to be deleted.
		.xxx xxxx		(Reserved bits)
9	9	. 3	ECTSMSG	The address of the second level message chain.
12	C	8	ECTPCMD	Name of the last primary command entered correctly by the terminal user.
20	14	8	ECTSCMD	Name of the last subcommand entered correctly by the terminal user.
28	1C	1	ECTSWS	Switches
		1...	ECTNOPD	No operands exist in the command buffer.
		..1.	ECTATRM	The command processor is being terminated by the Terminal Monitor Program.
		...1	ECTLOGF	The Logon/Logoff command processor has requested the Terminal Monitor Program to log the user off.
		.... 1...	ECTNMAL	No user messages (MAIL) to be received at logon.
		.... .1..	ECTNNOT	No broadcast notices (NOTICES) at logon.
		.x.. ..xx		(Reserved bits)
29	1D	. 3	ECTDDNUM	Counter for temporary DDNAMES.
32	20	8	ECTUSER	Reserved for installation use.
36	24	4	none	Reserved.

Figure 146. Environment Control Table

Protected Step Control Block

The Protected Step Control Block (PSCB) contains accounting information related to a single user. All timing information is in software timer units. A software timer unit is equal to 26.04166 micro seconds. Both the CPPL and the job step control block (JSCB), offset 264, point to the PSCB. (See System Control Blocks for further information on the JSCB.)

Offset		Bytes and		
Dec	Hex	Alignment	Field Name	Field Description
0	0	7	PSCBUSER	Contains the user ID left aligned and followed by blanks if necessary.
7	7	. . . 1	PSCBUSRL	The length of the non-blank portion of the user ID.
8	8	8	PSCBGPNM	Group name is initialized by Logon with information from the User Attribute Data Set (UADS). Used by DAIR to obtain the default unit name, if invoker of DAIR does not specify unit name. (See DA08UNIT, DA24UNIT, and DA30UNIT fields in DAIR parameter blocks.)
16	10	1	PSCBATR1	IBM user attributes.
		1...	PSCBCTRL	OPERATOR command user.
		.1..	PSCBACCT	ACCOUNT command user.
		..1.	PSCBJCL	SUBMIT, STATUS, CANCEL, OUTPUT command user.
		...x xxxx		(Reserved bits)
17	11	. 1	Byte 2	Reserved for IBM use.
18	12	. . 2	PSCBATR2	Available for use by the installation.
20	14	4	PSCBCPU	The cumulative time used by this terminal user during this session. This field is set to zero during logon.
24	18	4	PSCBSWP	The cumulative time that this terminal user has been resident in the region. This field is set to zero during logon.
28	1C	4	PSCBLTIM	The actual time of day that this user logged onto the time sharing system for this session.
32	20	4	PSCBTCPU	The total CPU time used by this terminal user, excluding the current session.
36	24	4	PSCBTSWP	The total time that the terminal user has been resident in the region during this accounting period, excluding the current session.

Figure 147. Protected Step Control Block (Part 1 of 2)

Offset Dec	 Hex	Bytes and Alignment	Field Name	Field Description
40	28	4	PSCBTCON	The first four bytes of an eight byte field containing the total "connect" time for the user during this accounting period, excluding the current session. Note: All times are in 26.04166-microsecond timer units.
44	2C	4	PSCBTC01	Second word of PSCBTCON. Total time that the user terminal is connected during the current session.
48	30	4	PSCBRLGB	Address of the re-logon buffer (RLGB).
52	34	4	PSCBUPT	Address of the User Profile Table (UPT).
56	38	2 . .	PSCBUPTL	Length of the User Profile Table (UPT) in bytes.
58	3A	. . 2		Reserved for IBM use.
60	3C	4	PSCBRSZ	Requested region size in 2K units.
64	40	8	PSCBU	Available for use by the installation.

Figure 147. Protected Step Control Block (Part 2 of 2)

Time-Sharing Job Block

The Time Sharing Job Block (TJB) contains information about a time sharing job's status. This information must be retained in storage while a user is swapped out. TJBs are obtained during time sharing initialization and reside in the time sharing control task region. The address of a TJB table, containing the TJBs, is located at offset zero in the time sharing CVT. The time sharing CVT's address is stored at location 229 (E5) in the system CVT.

Status information about terminals is contained in the Terminal Status Block (TSB). The address of the Terminal Status Block is the first word of the TJB. See TSO Control Program PLM, for a description of the Terminal Status Block (TSB).

Offset		Bytes and		
Dec	Hex	Alignment	Field Name	Field Description
0	0	4	TJBTSB	The address of the Terminal Status Block (TSB) that owns this terminal job. If this byte is zero, this job was started by operator command.
4	4	1	TJBATTN	The number of unprocessed attentions for this job.
5	5	. 1	TJBSTAX	The number of scheduled STAX exits.
6	6	. . 1	TJBSTAT	First byte of status flags.
		1...	TJBNJB	This TJB is currently unused by TSO.
		.1..	TJBINCOR	This user is currently swapped in.
		..1.	TJBLOGON	Set by terminal input/output control (TIOC) at dial-up to request logon.
		...1	TJBIWAIT	Terminal job is in input wait state.
		.... 1...	TJBOWAIT	Terminal job is in output wait state.
		.... .1..	TJBSILF	Indicates that the user is to be logged off. Set by IKJSILF subroutine. Indicates that the region control task should invoke IKJEAT07 to either post with a '622' ABEND an out-of-storage job, or cancel the current job.
		.... ..1.	TJBDISC	Set by Logon/Logoff to request TIOC to disconnect line.
		1	TJBSILF2	System-initiated logoff is in progress.

Figure 148. Time-Sharing Job Block (Part 1 of 3)

Offset Dec	Hex	Bytes and Alignment	Field Name	Field Description
7	7	. . . 1	TJBSTAT2	Second byte of status flags.
		1...	TJBHUNG	User's communication line is disconnected without logging off.
		.1..	TJBHOLD	User is in an output wait due to a hold option.
		..1.	TJBOCAB	TSO failure resulting in an out-of-main-storage abnormal termination.
		...1	TJBRNAV	The user cannot be logged onto TSO because of a machine check in region or the lack of a large enough region.
		.... 1...	TJBSURSV	Do not mark the swap unit available for use on next swap-in.
		.... .1..	TJBQUIS	Quiesce functions have started for the user.
		.... ..1.	TJBUSERR	User is ready to run.
		1	TJBDEAD	Used by IKJEAT07 to indicate abnormal termination recursion.
8	8	4	TJBEXTNT	Address of the TJB extension.
12	C	4	TJBRCB	Address of the region control block for this job.
16	10	4	TJBUMSM	Address of the user main storage map for this job.
20	14	4	TJBSDCB	Address of the swap DCB for this user.
24	18	2	TJBUTTMQ	Offset in TT map to first track map queue entry for the swap data set.
		Byte 1		
		1...	TJBUTTMP	Parallel swap.
		.xxx xxxx		These bits along with byte 2 contain the offset into the map queue. The map queue contains a chain of allocation units for this user on the swap data set. The address of the queue is in the ROBUTTMQ field of the time sharing region control block.
		Byte 2		
				(See explanation of byte 1.)
26	1A	. . 1	TJBRSTOR	Restore flags. Tested by the Region Control Task (RCT) restore operation (IKJEAR03).
		1...	TJBOWP	Set by Terminal Input/Output Coordinator (TIOC) to end an output wait condition.
		.1..	TJBIWP	Set by TIOC to end an input wait condition.
		...1	TJBLOGP	Post the ECB that the logon image is waiting for. Set by time sharing control logon and by IKJSIIF.
		.... 1...	TJBLWAIT	This user is in a long wait condition. If user is not made ready by restore processing, swap the user out again.
		.... .1..	TJBDDRD	Reset DDR non-dispatchability flag in TCB whose address is in IORMSCOM.
		.... ..1.	TJBFAT	An attention exit has been requested for the user.
		1	TJBDDRND	Set DDR non-dispatchability flag in TCB whose address is in IORMSCOM.
		..x.		Reserved bit.
27	1B	. . . 1	TJBUMSMN	The number of map entries in the User Main Storage Map (UMSM).

Figure 148. Time-Sharing Job Block (Part 2 of 3)

Offset		Bytes and		
Dec	Hex	Alignment	Field Name	Field Description
28	1C	8	TJBUSER	The ID of the user owning this job. Padded with trailing blanks. NOPURGE option.
36	24	4	TJBIPPB	The address of the first in a chain of Inter-Partition Post Blocks (IPPBs) indicating ECBs to be posted by the Restore routine.
40	28	1	TJBNEWID	The region ID of the region into which this user should be logged on. When this field is set by the end-of-task routine for Logon/Logoff, it identifies the new region to which the user will be shifted. The field is zero if the region is selected by the Driver.
41	29	. 1	TJBFLUSL	Level of last STAX macro instruction that was issued with the NOPURGE option.
42	2A	. . 2	TJBTJID	The terminal job ID for this job.
44	2C	1	TJBMONI	Flags indicating information requested. Set by the MONITOR subcommand of the OPERATOR command. Used by job management.
		1...	TJBMDSN	Indicates that the first non-temporary data set allocated to a new volume should be displayed on this user's terminal as part of the MOUNT message. (Dsnames requested.)
		.1..	TJBMJBN	Indicates that the name of each job is to be displayed on this user's terminal when each job is initiated and terminated, and that the unit record allocations are to be displayed when a job step is initiated. (Jobnames requested.)
		..1.	TJBMSES	Indicates that when a terminal session is initiated or terminated, a message is displayed on this user's terminal. (Session requested.)
		...1	TJBMSPA	Indicates that the available space on a direct access device is to be displayed on this user's terminal as part of the DEMOUNT message. (Space requested.)
		.... 1...	TJBMSTA	Indicates that at the end of a job or job step certain data set disposition information should be printed with the DEMOUNT messages. These dispositions are: KEEP, CATLG, or UNCATLG. (Status requested.)
		.... .1..	TJBGETBF	TPUT should try to obtain additional buffers for the user before entering a wait condition.
		xx		(Reserved bits)
45	2D	. 1	TJBSTAT3	
		1...	TJBDISC2	Disconnect this TJB.
		.... .1..	TJBSOEM	Indicates swapout error message recursion.
47	2F	. . . 1		Reserved

Figure 148. Time-Sharing Job Block (Part 3 of 3)

User Profile Table

The User Profile Table (UPT) is a 16-byte data area located in subpool zero. The UPT contains information about the terminal user and is created by the LOGON/LOGOFF Scheduler from information stored in the user attribute data set and from parameters of the LOGON command. It is updated by the PROFILE command processor. The UPT address is in the CPPL.

Offset		Bytes and		
Dec	Hex	Alignment	Field Name	Field Description
0	0	2		Reserved for IBM use.
2	2	. . 10	UPTUSER	Reserved for installation use.
12	C	1	UPTSWS	User environment switches.
		.0..	UPTNPRM	Prompting is to be done.
		.1..		No prompting.
		..0.	UPTMID	Message identifiers suppressed.
		..1.		Message identifiers printed.
		...0	UPTNCOM	Allow user communication via SEND command.
		...1		No user communication.
		.... 0...	UPTPAUS	No prompting pause for '?' when in non-interactive mode (i.e., when next input is not from terminal).
		.... 1...		Prompting pause for '?' when in interactive mode.
		.... .0..	UPTALD	ATTENTION is not a line delete character.
		.... .1..		ATTENTION has been specified as a line delete character.
		x... ..xx		Reserved bits.
13	D	. 1	UPTCDEL	Character deletion character.[1]
14	E	. . 1	UPTLDEL	Line deletion character.[1]
15	F	. . . 1		Reserved.

[1] See System Control Blocks for further information.

Figure 149. User Profile Table

Appendix B: Notation for Defining Macro Instructions

The notation used in this publication is described in the following paragraphs.

1. The set of symbols listed below are used to define macro instructions, but should never be written in the actual macro instruction.

hyphen	-
underscore	_
braces	{}
brackets	[]
ellipsis	...

The special uses of these symbols are explained in paragraphs 5-9.

2. Upper-case letters and words, numbers, and the set of symbols listed below should be written in macro instruction exactly as shown in the definition.

apostrophe	'
asterisk	*
comma	,
equal sign	=
parentheses	()
period	.

3. Lower-case letters, words, and symbols appearing in a macro instruction definition represent variables for which specific information should be substituted in the actual macro instruction.

Example: If name appears in a macro instruction definition, a specific value (for example, ALPHA) should be substituted for the variable in the actual macro instruction.

4. Braces group related items, such as alternatives.

Example: The representation

$$\text{ALPHA=(}\left\{\begin{matrix}\text{A}\\ \underline{\text{B}}\\ \text{C}\end{matrix}\right\}\text{, D)}$$

indicates that a choice should be made among the items enclosed within the braces. If A is selected, the result is ALPHA=(A,D). If B is selected, the result can be either ALPHA=(,D) or ALPHA=(B,D).

5. Brackets also group related items; however, everything within the brackets is optional and may be omitted.

   Example: The representation

   $$\text{ALPHA=(}\begin{bmatrix}\text{A}\\\text{B}\\\text{C}\end{bmatrix}\text{,D)}$$

   indicates that a choice can be made among the items enclosed within the brackets or that the items within the brackets can be omitted. If B is selected, the result is: ALPHA=(B,D). If no choice is made, the result is: ALPHA=(,D).

6. Stacked items represent alternatives. Only one such alternative should be selected.

   Example: The representation

   $$\begin{bmatrix}\text{A}\\\text{B}\\\text{C}\end{bmatrix} \quad \text{or} \quad \begin{Bmatrix}\text{A}\\\text{B}\\\text{C}\end{Bmatrix}$$

   indicates that either A or B or C should be selected.

7. Hyphens join lower-case letters, words, and symbols to form a <u>single</u> variable.

   Example: If member-name appears in a macro instruction definition, a specific value (for example, BETA) should be substituted for the variable in the actual macro instruction.

8. An underscore indicates a default option. If an underscored alternative is selected, it need not be wirtten in the actual macro instruction.

   Example: The representation

   $$\begin{bmatrix}\text{A}\\\underline{\text{B}}\\\text{C}\end{bmatrix} \quad \text{or} \quad \begin{pmatrix}\text{A}\\\underline{\text{B}}\\\text{C}\end{pmatrix}$$

   indicates that either A or B or C should be selected; however, if B is selected, it need not be written, because it is the default option.

9. An ellipsis indicates that the preceding item or group of items can be repeated more than once in succession.

   Example:

   ALPHA[,BETA]...

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

Glossary

The following are definitions of words and phrases which are used in this publication but are not defined in IBM Data Processing Glossary, GC20-1699. For words and phrases which are in general use in IBM publications, refer to IBM Data Processing Glossary.

in-storage list: A chain of input lines in main storage, such as commands in an EXEC procedure, that are used in place of terminal input.

Logical line: One or more lines typed at a terminal and treated as a unit. A logical line may consist of one or more physical lines, in which the symbol"-" indicates continuation.

LOGOFF: The TSO command that terminates a user's terminal session.

LOGON: The TSO command that a user must enter to initiate a terminal session.

LOGON procedure: A cataloged procedure that is executed as a result of a user entering the LOGON command.

profile (user): The set of characteristics that describe the user to the system.

Index

Indexes to systems reference library manuals are consolidated in the publication IBM System/360 Operating System: Systems Reference Library Master Index, Order No. GC28-6644. For additional information about any subject listed below, refer to other publications listed for the same subject in the Master Index.

IBM System/360 Operating System
Time Sharing Option
Guide to Writing a Terminal Monitor Program
or a Command Processor

READER'S COMMENT FORM

Your views about this publication may help improve its usefulness; this form will be sent to the author's department for appropriate action. Using this form to request system assistance or additional publications will delay response, however. *For more direct handling of such requests, please contact your IBM representative or the IBM Branch Office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Index Figures Examples Legibility

What is your occupation? ____
Number of latest Technical Newsletter (if any) concerning this publication: ______________
Please indicate in the space below if you wish a reply.

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.

Cut or Fold Along Line

GC28-6764-2

Your comments, please . . .

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. Your comments on the other side of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Cut or Fold Along Line

Fold Fold

First Class
Permit 81
Poughkeepsie
New York

Business Reply Mail
No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
Department D58, Building 706-2
PO Box 390
Poughkeepsie, New York 12602

Fold Fold

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

IBM System/360 Operating System
Time Sharing Option
Guide to Writing a Terminal Monitor Program
or a Command Processor

READER'S COMMENT FORM

Your views about this publication may help improve its usefulness; this form will be sent to the author's department for appropriate action. Using this form to request system assistance or additional publications will delay response, however. *For more direct handling of such requests, please contact your IBM representative or the IBM Branch Office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Index Figures Examples Legibility

Cut or Fold Along Line

What is your occupation? ____
Number of latest Technical Newsletter (if any) concerning this publication: ______________
Please indicate in the space below if you wish a reply.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.

GC28-6764-2

Cut or Fold Along Line

System/360 OS TSO Guide to Writing a TMP or a CP (S360-36) Printed in U.S.A. GC28-6764-2

Your comments, please . . .

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. Your comments on the other side of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Fold Fold

First Class
Permit 81
Poughkeepsie
New York

Business Reply Mail
No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
Department D58, Building 706-2
PO Box 390
Poughkeepsie, New York 12602

Fold Fold

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)