GC28-1295-0
File No. S370-39

Program Product

MVS/Extended Architecture TSO Guide to Writing a Terminal Monitor Program or a Command Processor

MVS/System Product - JES2 Version 2	5740-XC6
MVS/System Product - JES3 Version 2	5665-291
Data Facility Product	5665-284

IBM

First Edition (January, 1983)

This edition applies to Version 2 of MVS/System Product 5665-291 or 5740-XC6 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/370 Bibliography*, GC20-0001, for the editions that are applicable and current.

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

Publications are not stocked at the address given below. Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Information Development, Department D58, Building 920-2, PO Box 390, Poughkeepsie, N.Y. 12602. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Preface

This publication describes features of TSO that can be replaced, modified, or added to by each installation, to adapt the command system to the installation's particular needs. The manual is intended for programmers who are responsible for modifying portions of TSO.

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 application programs. These features include:

- Service routines
- Macro instructions
- SVCs

This publication contains information required by a programmer writing a terminal monitor program or a command processor for TSO. 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 fifteen sections:

- Introduction
- Terminal Monitor Program
- Command Processors
- MVS/Extended Architecture Considerations
- Processing Terminal Requests -- The TSO Service Routines
- 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/TPG SVC for Terminal I/O
- Using Terminal Control Macro Instructions
- Command Scan and Parse -- Determining the Validity of Commands
- Catalog Information Routine (IKJEHCIR)
- Default Service Routine (IKJEHDEF)

The first three sections describe the functions performed by terminal monitor programs and command processors. The fourth section describes programming considerations for MVS/Extended Architecture systems that affect the tasks documented in this manual. The fifth section describes how to interface with the TSO service routines to process terminal requests.

The next ten sections describe the macro instructions and service routines that a programmer can use to provide the required functions. The macro instructions and service routines can be used to:

- Issue messages
- Schedule and process attention interruptions

- Allocate, free, concatenate, and deconcatenate data sets during program execution
- Provide I/O between a program and a terminal
- Control terminal functions and attributes
- Determine the validity of commands, subcommands, and operands entering the system
- Retrieve information from the system catalog
- Construct a fully-qualified data set name

The packaging of the TSO library for MVS/Extended Architecture (MVS/XA) is as follows:

- A base book as updated by an MVS/System Product Version 2 System Library Supplement,
- A new book (*System Programming Library: TSO, TSO Terminal User's Guide, TSO Guide to Writing a TMP or a CP*).

Note that the titles for the TSO library for MVS/XA begin with the "MVS/Extended Architecture" system prefix.

Prerequisite and Reference Publications

The reader of this publication should have a knowledge of the structure of TSO.

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

Principles of Operation

MVS/Extended Architecture Data Management Macro Instructions, GC26-4014

MVS/Extended Architecture Data Management Services, GC26-4013

MVS/Extended Architecture VSAM Programmer's Guide, GC26-4015

MVS/Extended Architecture System Programming Library: Data Management, GC26-4010

MVS/Extended Architecture System Programming Library: Initialization and Tuning, GC28-1149

MVS/Extended Architecture JCL, GC28-1148

MVS/Extended Architecture System Programming Library: Supervisor Services and Macro Instructions, GC28-1154

MVS/Extended Architecture System Programming Library: TSO, GC28-1173

MVS/Extended Architecture System Programming Library: System Modifications, GC28-1152

MVS/Extended Architecture System Programming Library: System Macros and Facilities

Vol. I, GC28-1150

Vol. II, GC28-1151

Data Areas

(for MVS/System Product Version 2 JES2) LYB8-1191

(for MVS/System Product Version 2 JES3) LYB8-1195

Data Area Usage Table

(for MVS/System Product Version 2 JES2) LYB8-1193

(for MVS/System Product Version 2 JES3) LYB8-1197

Symbol Usage Table

(for MVS/System Product Version 2 JES2) LYB8-1192

(for MVS/System Product Version 2 JES3) LYB8-1196

OS/VS2 Directory, SYB8-0743

MVS/Extended Architecture TSO Command Language Reference

(*OS/VS2 TSO Command Language Reference*, GC28-0646, as amended by GD23-0259)

MVS/Extended Architecture TSO Command Processor Logic, Volume I - ACCOUNT

(*OS/VS2 TSO Command Processor Logic, Volume I - ACCOUNT*, SY28-0651, as amended by LD23-0270)

MVS/Extended Architecture TSO Command Processor Logic, Volume II - EDIT,

(*OS/VS2 TSO Command Processor Logic, Volume II - EDIT*, SY33-8548, as amended by LD23-0272)

MVS/Extended Architecture TSO Command Processor Logic, Volume IV,

(*OS/VS2 TSO Command Processor Logic, Volume IV*, SY28-0652, as amended by LD23-0265)

MVS/Extended Architecture TSO Terminal Monitor Program and Service Routines Logic

(*OS/VS2 TSO Terminal Monitor Program and Service Routines Logic*, SY28-0650, as amended by LD23-0262)

MVS/Extended Architecture TSO Terminal User's Guide, GC28-1274

MVS/Extended Architecture Message Library: TSO Terminal Messages

(*OS/VS Message Library: TSO Terminal Messages*, GC38-1046, as amended by GD23-0269)

MVS/Extended Architecture System Programming Library: 31-Bit Addressing, GC28-1158

Referenced Products

1. All references to MVS/Extended Architecture (or to MVS/XA) indicate **Data Facility Product (5665-284)** and **MVS/System Product Version 2 - JES2 (5740-XC6)** or **MVS/System Product Version 2 - JES3 (5665-291)**.
2. All references to VTAM, TSO/VTAM, and ACF/VTAM indicate the program product **ACF/VTAM Version 2 (5665-280)**.
3. All references to TCAM and TSO/TCAM indicate the program product **ACF/TCAM Version 2 Release 4 (5735-RC3)**.
4. All references to TSO/E indicate the program product **TSO Extensions (5665-293)**.

Contents

Figures

Introduction

TSO consists of many relatively small, functionally distinct modules of code. One major benefit of this modular construction is that the installation may add to or modify TSO to better suit the needs of its users. You can add to or replace IBM-supplied code with your own, and delete those functions of TSO which you do not require.

TSO is composed of modules that communicate with the user and perform the work requested by him. Modifications to the control program should be made only by system programmers responsible for the proper functioning of TSO within MVS. Each installation can replace or augment 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 problem program that accepts and interprets commands. The TMP causes the appropriate command processors to be scheduled and executed.

A terminal monitor program must be able to communicate with the user at the terminal, load 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.

When a user logs on to TSO, he must either specify the name of a LOGON procedure via the LOGON command or accept the use of his default procedure name from the user attribute data set (UADS). In either case, the program named in the EXEC statement of the LOGON procedure is attached during the logon 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.

Once the logon has completed, the terminal monitor program requests the user at the terminal to enter a command name. The IBM-supplied TMP writes a READY message to the terminal to indicate that a command should be entered. The TMP determines if the response entered is a command. If the response is a command, the TMP attaches the requested command processor, and the command processor performs the computing functions requested by the user at the terminal.

When writing your own command processors, keep in mind that you can add them to the IBM-supplied command library, concatenate your own command library to the one supplied by IBM, or 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, and process attention interruptions. If required command processors must be able to load and pass control to subcommand processors, as well as process abnormal terminations of those 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 load and pass control to a command processor. If a command processor has subcommand processors, it must be able to load and pass control to them.
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 for 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

There are three ways a program running under TSO can communicate with a user:

1. The BSAM or QSAM access methods.
2. The STACK, GETLINE, PUTLINE, and PUTGET I/O service routines. These I/O service routines are invoked via the STACK, GETLINE, PUTLINE, and PUTGET macro instructions respectively. They 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 and returns all input lines other than commands, 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 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 terminal that requires a response is called a conversational message.

3. The TGET, TPUT, and TPG supervisor call. A supervisor call routine, SVC 93, is invoked via the TGET, TPUT, and TPG macro instructions. TGET, TPUT, and TPG provide a route for I/O to a terminal.

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 Command and Subcommand Processors

A terminal monitor program must be able to recognize a command name entered into the system, load the requested command processor, and pass control to it. A command processor must be able to perform similar functions when a subcommand name is entered.

Your TMP or command processor can use the command scan service routine to search the input line for a syntactically valid command name or subcommand name. The ATTACH macro instruction can then be issued to pass control to the requested routines and to establish ESTAI exits. (See "Responding to Abnormal Terminations" below.)

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 syntactical 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 routine the functions the user is requesting.

Responding to Abnormal Terminations

A programmer coding a routine to run under TSO should do all that is possible to keep that routine from causing an abnormal termination of a TSO user. If you write your own terminal monitor program or command processors, you should use the ESTAE or FESTAE macro instruction and the ESTAI operand on the ATTACH macro instruction to provide error handling exits.

Use the ESTAE or FESTAE 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 ESTAI 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

A terminal monitor program and any command processor that accepts subcommands must be able to respond to an attention interruption entered from the terminal. TSO interprets an attention interruption as a signal that the user wants to halt current program execution, possibly 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.

To build the control blocks and queues necessary for the system to recognize and schedule your attention handling routines, use the STAX service routine, which is invoked via the STAX macro instruction.

The Dynamic Allocation of Data Sets

Aside from the four basic functions provided by a terminal monitor program or a command processor, other useful time sharing functions can be obtained using routines provided by IBM. You can invoke dynamic allocation routines using the dynamic allocation interface routine (DAIR) to:

- Obtain the current status of a data set
- Allocate a data set
- 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

It is recommended, however, that you invoke dynamic allocation directly whenever possible (especially when writing new command processors) to take advantage of the additional functions available and to decrease system overhead. For a detailed description of these functions and how to invoke dynamic allocation directly, refer to *SPL: System Macros and Facilities.*

The DAIR interface, designed to invoke dynamic allocation for you, is maintained so that existing command processors do not have to be modified to invoke dynamic allocation directly. DAIR acts as a translator; it converts the DAIR parameter list it receives as input to a dynamic allocation parameter list, which it then passes to dynamic allocation.

Summary

Most of the functions of terminal monitor programs and command processors can be provided with macro instructions, service routines, or supervisor call routines supplied by IBM. The following sections describe how to use these macros and routines in a TMP or command processor.

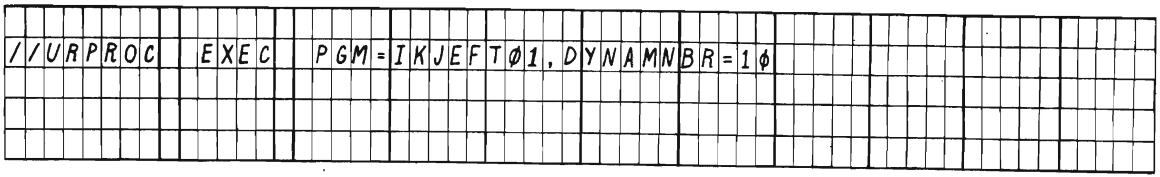
The terminal monitor program (TMP) is a problem program that provides an interface between the terminal user, command processors, and the TSO control program. TSO LOGON causes the system initiator to attach the program named on the EXEC statement of the user's LOGON cataloged procedure. This may be the IBM-supplied TMP or any user-supplied alternate.

The maximum number of concurrently allocated data sets allowed in a given TSO session is defined in the user's LOGON procedure. The LOGON procedure indicated on the LOGON command may contain DD statements that define data sets to be used during the TSO session, other DD statements, called DD DYNAM statements, and the DYNAMNBR parameter of the EXEC statement. These statements are used in combination to determine the maximum number of data sets that may be allocated to the user at any one time during the session. The formula for determining the maximum is:

Maximum = # DD statements + # DD DYNAM statements + the number supplied on the DYNAMNBR parameter of the EXEC statement.

The DYNAMNBR parameter obsoletes the DD DYNAM statement as a way of establishing the maximum number of data sets. While existing DD DYNAMS continue to be included in the formula for determining the maximum, use of the DYNAMNBR keyword is recommended because it involves only one statement (the EXEC statement) and can be modified easily.

Figure 1 shows an example of the EXEC statement in a user LOGON procedure. This procedure is equivalent to a LOGON procedure containing 10 DD DYNAM statements and no DYNAMNBR operand. For a complete discussion of a LOGON procedure, see *SPL: TSO*.



```
//URPROC  EXEC  PGM=IKJEFTØ1,DYNAMNBR=1Ø
```

Figure 1. A LOGON Procedure

The terminal monitor program you use can be the TMP supplied with TSO, 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 TSO.

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 interruption from the terminal, halting execution 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 IBM-supplied TMP functions, and how it fits together with the rest of TSO. For a more specific description of the IBM-supplied TMP, see *TSO Terminal Monitor Program and Service Routines Logic.*

Terminal Monitor Program Initialization

In a LOGON procedure, the terminal monitor program (TMP) name must appear as the first operand of the PGM= keyword operand on the EXEC statement.

When the TMP is attached:

- Register 1 contains the address of the field containing the length and data of the EXEC parameter. The IBM-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 ESTAE and ESTAI 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.

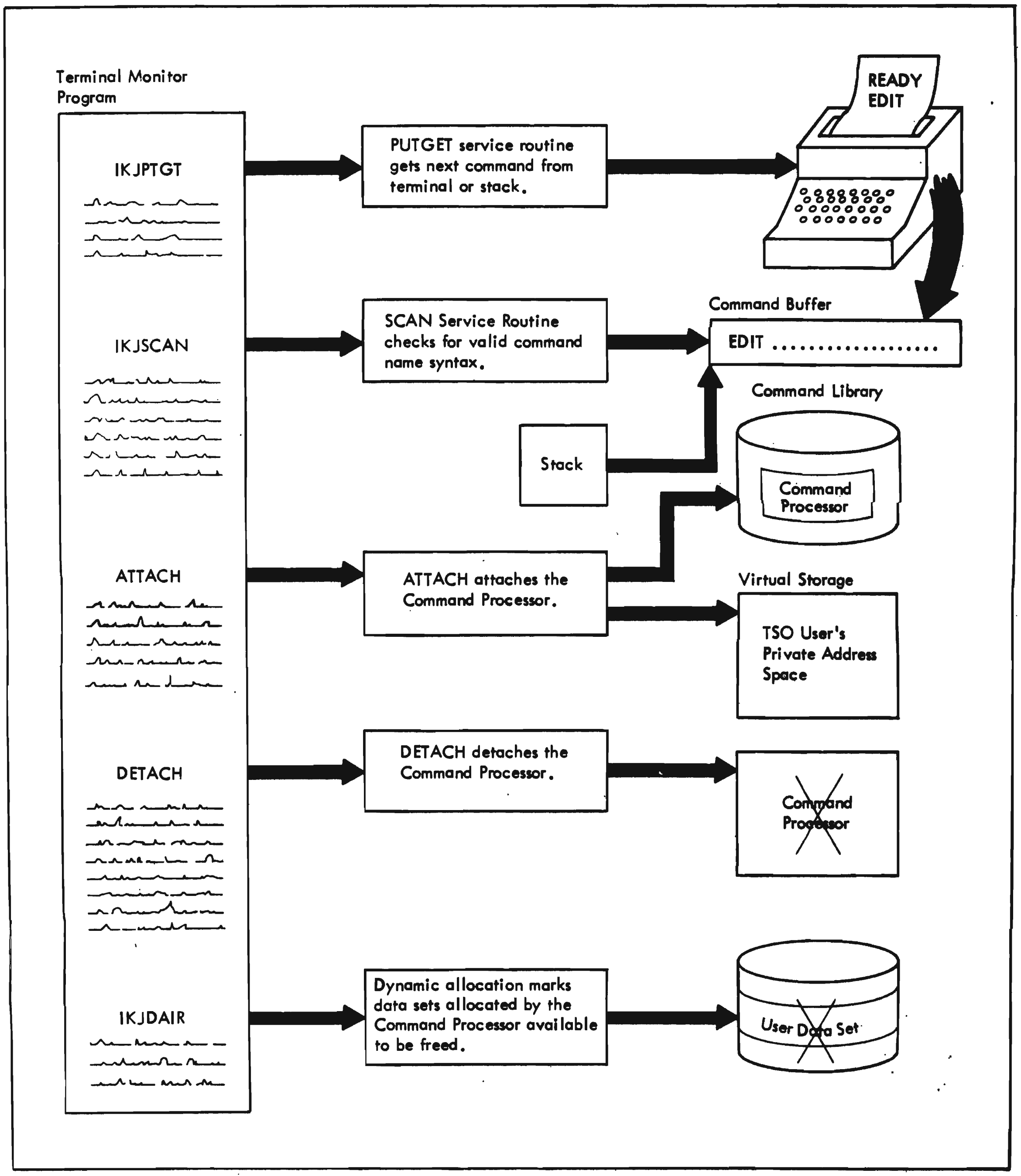
The TMP determines whether it is running in a TSO or a batch environment by testing the time-sharing bit in the TCB. If the TMP is running in a batch environment, it will use the DATASET keyword while invoking the STACK service routine to cause GETLINE and PUTLINE to be directed to data sets. The TMP must also build the same control blocks that LOGON would have built.

The IBM-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.

The TMP should not pass in-line parameter lists to commands or to TSO service routines. Subpool 251 should not be used for parameter lists. The command processor parameter list (CPPL), described later in this book, should be in subpool 1. You may use the IKJTMPWA macro to map the TMP workarea.

Requesting a Command

Figure 2 summarizes the steps taken by a terminal monitor program to obtain a command, to pass control to the commmand processor, and to detach the command processor when it has finished.



Terminal Monitor Program
IKJPTGT
IKJSCAN
ATTACH
DETACH
IKJDAIR
PUTGET service routine gets next command from terminal or stack.
READY
EDIT
SCAN Service Routine checks for valid command name syntax.
Command Buffer
EDIT ....
Command Library
Stack
Command Processor
ATTACH attaches the Command Processor.
Virtual Storage
TSO User's Private Address Space
DETACH detaches the Command Processor.
Command Processor
Dynamic allocation marks data sets allocated by the Command Processor available to be freed.
User Data Set

Figure 2. Requesting a Command

Use the PUTGET service to request a command from the terminal, 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 whether the line of input is a syntactically valid command name.

Use the ATTACH macro instruction (specifying an ESTAI exit routine) to pass control to the requested command processor.

Your TMP must create any parameter lists expected by the command processor and pass them to the newly attached command processor. The IBM-supplied TMP passes the address of a command processor parameter list in register one. See the sections entitled "MVS/Extended Architecture Considerations" and "Processing Terminal Requests -- The TSO Service Routines" for more information about the interface between the TMP and command processors.

When the command processor completes, the TMP releases it via a DETACH macro instruction, uses dynamic allocation to indicate that dynamically allocated data sets may be freed, and uses the PUTGET service routine to obtain another command.

The TSEVENT macro facilitates the use of the generalized trace facility (GTF) to trace the attaching of a command processor by an installation-supplied terminal monitor program. The TSEVENT macro results in control being passed to a GTF hook located in the system resources manager (SRM) interface program.

User written TMPs should issue the TSEVENT macro before attaching each command processor.

Issue the TSEVENT macro instruction as follows:

1. Load register 1 with the first four characters of the command name being attached or released.
2. Load register 15 with the last four characters of the command name.
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, command scan) is terminating abnormally.

Intercepting a Subtask ABEND

When a subtask of the terminal monitor program begins to terminate abnormally, the TMP ESTAI exit, specified by the TMP when it attached the subtask, receives control. The TMP ESTAI exit receives control under the TCB of the abending subtask. The subtask will already have performed its own ESTAE processing, if any was specified. Figure 4 shows the relationship between the ABEND, the ESTAE, and the ESTAI. For additional information about expanded recovery facilities available through ESTAI, refer to *Supervisor Services and Macro Instructions.*

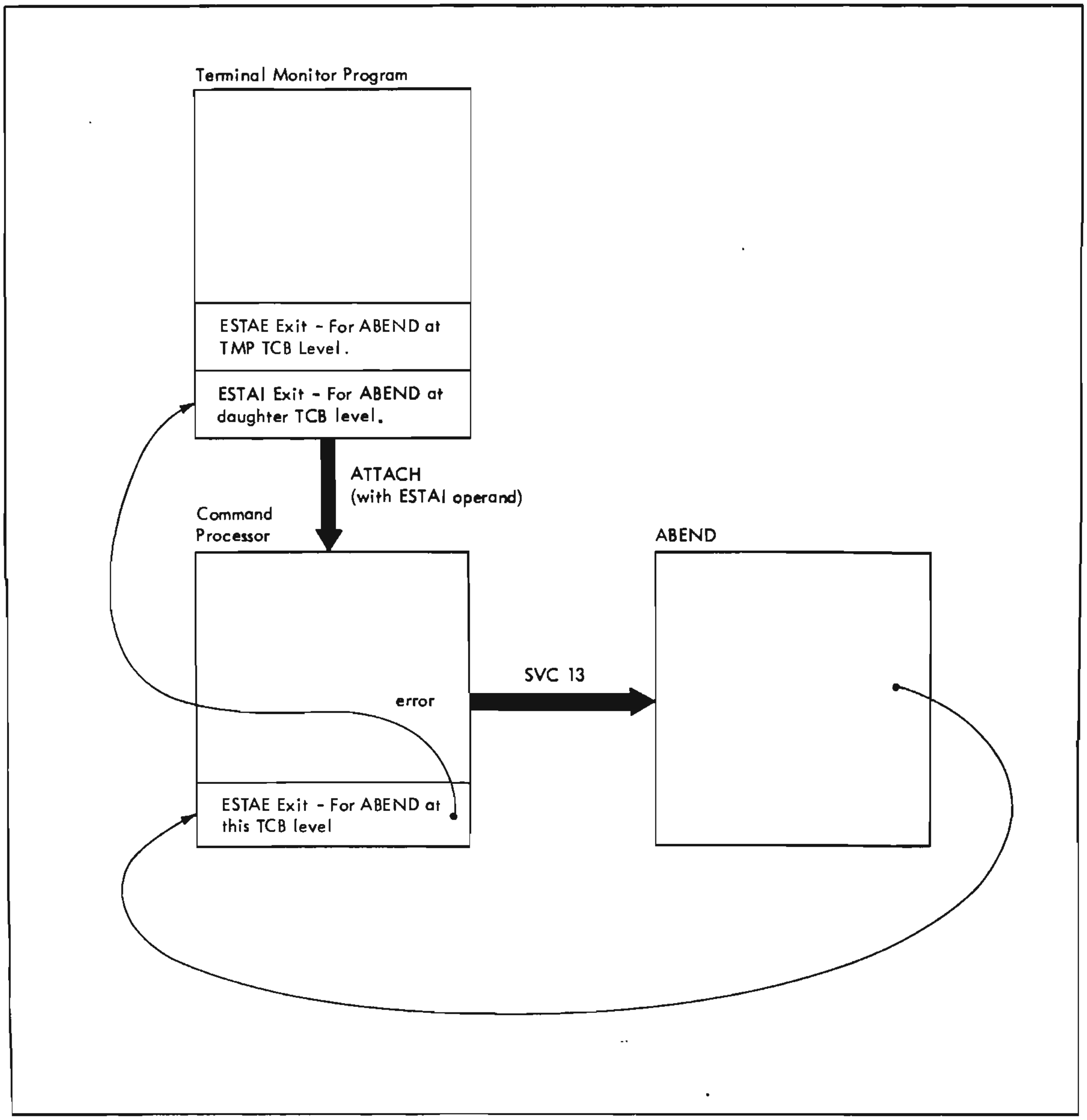

Terminal Monitor Program
ESTAE Exit - For ABEND at TMP TCB Level.
ESTAI Exit - For ABEND at daughter TCB level.
ATTACH (with ESTAI operand)
Command Processor
ABEND
SVC 13
error
ESTAE Exit - For ABEND at this TCB level

Figure 4. ABEND, ESTAI, STAE Relationship

The TMP must inform the user at the terminal of the ABEND situation, and allow the user to enter another command. 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 terminal.

The terminal user has three options:

1. He can allow the ABEND to continue by entering a null line (pressing the ENTER key).
2. He can terminate processing of the ABEND by entering a command name other than TIME.
3. He can request any second-level messages concerning the terminating program by entering a question mark.

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 than TIME, the TMP processes the new command name after detaching the subtask.

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

When the TIME command is entered, the TMP links to the TIME command processor to obtain the time information. Upon completion of the TIME command, the user still has the above three options.

Intercepting a TMP Task ABEND

When the TMP (or any program linked to by the TMP) causes an ABEND, the TMP ESTAE exit gains control. The TMP specifies its own ESTAE exit routine by issuing the ESTAE macro instruction. (See *SPL: System Macros and Facilities,* for a discussion of the ESTAE macro instruction.)

For a discussion of interface considerations for establishing ESTAE and ESTAI exit routines, refer to "ESTAE/ESTAI Exit Routines -- Intercepting an ABEND" in the section on command processors in this manual.

Your TMP ESTAE exit routine can use the contents of the ESTAE 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
- The contents of the program registers

If your TMP ESTAE 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's TCB is terminating abnormally. Then, the TMP ESTAE routine should 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, 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 and the TMP fails again, the ESTAE routine should pass control to the PUTLINE service routine to notify the user. A logoff should be forced by returning to the LOGON/LOGOFF scheduler.

For additional information about expanded recovery facilities available through ESTAE and ESTAI, refer to *Supervisor Services and Macro Instructions*.

Processing an Attention Interruption

After having been attached, the TMP must set up its attention handling facilities. For this initialization process, you can use the STAX macro instruction to pass the address of your attention handling routine to the system.

For a discussion of interface considerations for attention exit routines, refer to "Specifying a Terminal Attention Exit -- The STAX Macro Instruction" later in this book.

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. For a description of how the user can request different levels of attention exits, see "Attention Interruption Handling -- The STAX Service Routine" later in this book.

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), that has not provided an attention handling routine, is in control.
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 5.

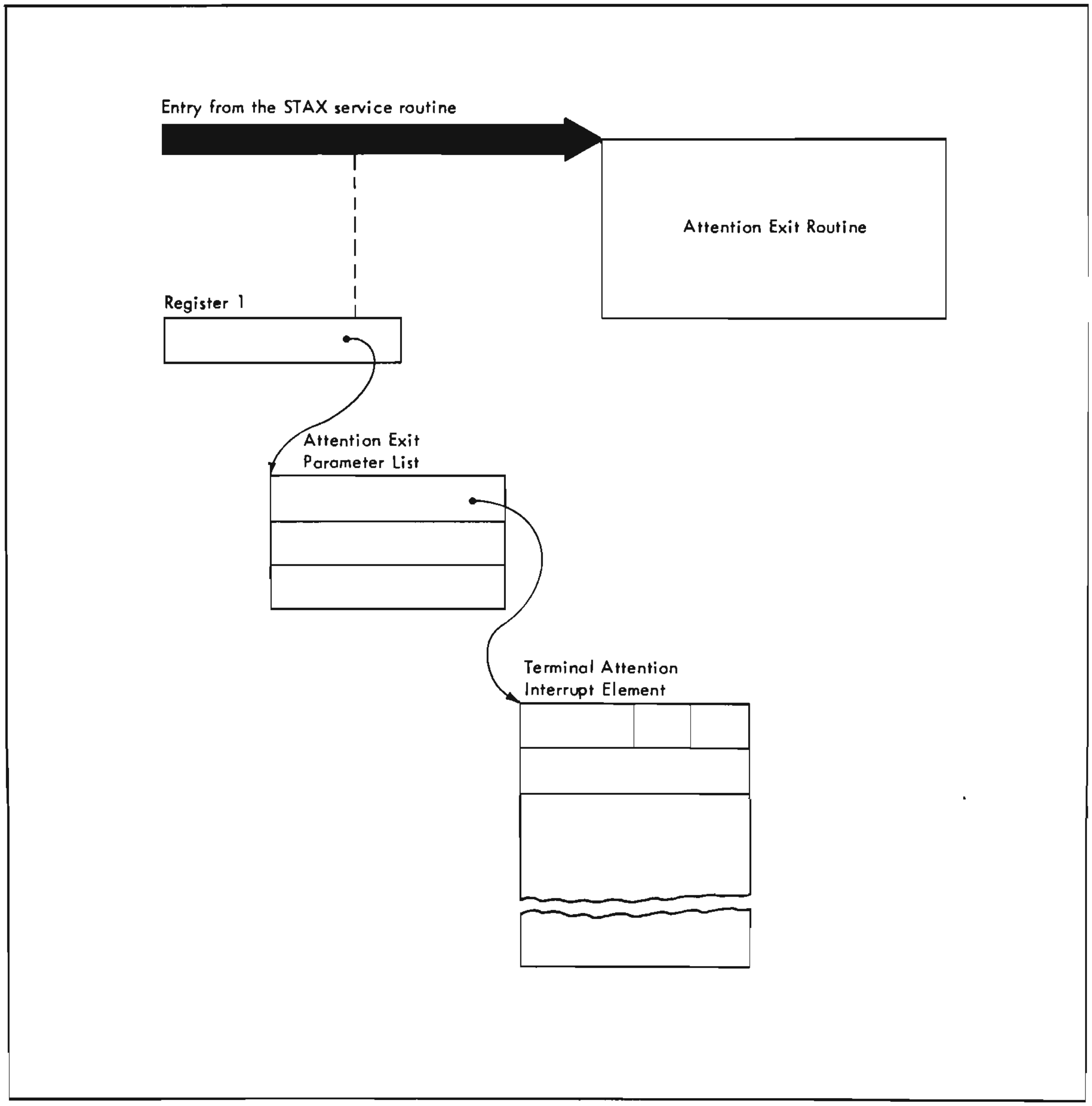


Figure 5. Parameters Passed to the Attention Exit Routine

The Attention Exit Parameter List

Figure 6 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 include the USADDR operand in the STAX macro instruction.

Figure 6. 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 7 shows the format of the TAIE, which is mapped by the IKJTAIE macro.

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 7. 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 second-level informational message chain (if one exists) to be written to the terminal, puts out the mode message again, 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 second-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. It may be link-edited into any library in the system link library list (LNKLSTxx) or SYS1.LPALIB. The command processor may be placed in a date set that is specified on the STEPLIB DD statement in a LOGON procedure. Execution should normally not be handled from a STEPLIB because of a decrease in performance during a system and TSO session. Refer to "Adding Commands to TSO" for a description of when a STEPLIB should be used.

The internal logic of the IBM-supplied command processors is described in *TSO Command Processor Logic,* Volumes I, II, and IV. The command language used to request each of these command processors is described in *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 TSO, and provides guidelines for coding your own command processors.

This section is divided into the following topics:

- Adding Commands to TSO - Describes how to add a new command processor to TSO
- Command Processor Coding Conventions - Describes *normal* interface conventions
- 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 ESTAE and ESTAI 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
- The HELP Data Set - Discusses the HELP data set, how to write HELP members, and how to update existing HELP members

Adding Commands to TSO

There are three methods you can use to add a new command processor to TSO.

1. You can enter your command processor as a member of the partitioned data set SYS1.CMDLIB, via the linkage editor.
2. You can create your own command library and concatenate it to the SYS1.CMDLIB data set. In this case, use the utility IEBUPDTE to create new statements in the link list (LNKLST00 or LNKLSTxx) in SYS1.PARMLIB.
3. Generally, unauthorized users can request that a LOGON procedure be created that specifies, on the STEPLIB DD statement, the name of the partitioned data set containing the command.

Command Processor Coding Conventions

The TMP uses standard linkage conventions in passing control to a command processor. The command processor parameter list (CPPL) is the input parameter list to all command processors. For more information on the CPPL, see the section called "Interfacing with the TSO Service Routines" later in this book.

Command processors should contain logic that issues error messages. These messages should handle all error codes, expected or unexpected, from any routine or SVC they invoke. Whenever possible, generalized routines such as DAIRFAIL should be used. Use of these routines allows the issuance of meaningful error messages for return codes.

When returning control to the TMP, the command processor should use standard linkage and set a return code in general register 15. Command procedures (CLISTs) may then check this code for the following conventions:

0-normal execution

12-termination error during execution (no error exists if a command processor is able to obtain required information by prompting)

Command Processor Use of the TSO Service Routines

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

Note: "MVS/Extended Architecture Considerations" lists the linkage attributes for the TSO service routines. Additional descriptions of considerations caused by 31-bit addressing are provided in the sections describing the routines and macros.

STACK Service Routine

Use the STACK service routine to change the source of input by adding an element to the input stack or 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 should build an in-storage list containing the commands to be executed and issue 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. Your command processor performs functions similar to those performed by the IBM-supplied EXEC command (that is, it executes a command procedure). Your command processor should 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 clear the input stack, before returning control to the TMP. The input stack must be cleared or command procedure (CLIST) processing will not be handled correctly. Commands such as DELETE and FREE do not flush the stack if the module requested was not found.

Catalog Information Routine

The catalog information routine (IKJEHCIR) retrieves information from the system catalog. This information may include a data set name, index name, control volume address, or volume ID. The information may be requested from a specific user catalog. If you do not specify a specific catalog, IKJEHCIR searches the system default catalog. An entry code indicates what kind of information is being requested.

Use the CALL, CALLTSSR, or LINK macro instruction to invoke the catalog information routine.

Note: For additional information concerning the catalog information routine, see "Catalog Information Routine (IKJEHCIR)" later in this book.

Default Service Routine

The default service routine (IKJEHDEF) constructs a fully-qualified data set name when the calling routine provides a partially-qualified data set name. A fully-qualified data set name has three fields: a userid, a data set name, and a descriptive qualifier.

Use the CALL, CALLTSSR or LINK macro instruction to invoke the default service routine. At entry, general register 1 must point to the default parameter list (DFPL). IKJEHDEF then invokes the catalog information routine (IKJEHCIR) to search the system catalog for the required qualifiers. When the search argument is satisfied, the default service routine returns to the calling control program. All of the general registers are restored except for register 15 which contains the return code.

Note: For additional information concerning the default service routine, see *TSO Terminal Monitor Program and Service Routines Logic.*

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. For efficient execution, issue FREEMAIN macro instructions within each command processor, or within each subtask created by the command processor, to free the GETLINE buffers it obtains.

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. TPUT should not be used under these circumstances. The GNRLFAIL service routine should be used to issue meaningful error messages for return codes from PUTLINE.

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 enters a question mark in response to a message issued with a PUTGET macro instruction, the PUTGET service routine displays 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 "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.

IKJEFF02 Message Issuer Service Routine

If you make numerous insertions into messages, you should use this service routine instead of PUTLINE and PUTGET. Also, when you use IKJEFF02, all of your messages can be placed in a single CSECT or a single module.

DAIR Service Routine

You may use the DAIR service routine to obtain information about a data set and, if necessary, to invoke dynamic allocation routines to perform the requested function. However, additional functions are available if you invoke dynamic allocation (SVC 99) directly. Another drawback to using DAIR is that DAIR, which normally invokes SVC 99, increases system overhead. For a discussion of how to invoke dynamic allocation directly, refer to *SPL: System Macros and Facilities.*

If you are going to use DAIR, you should read the section called "Dynamic Allocation of Data Sets - The Dynamic Allocation Interface Routine (DAIR)" later in this book and adhere to the following guidelines:

- Command processors should allocate data sets by DSNAME and use the DDNAMES returned by DAIR to open the data sets. If necessary, command processors should pass the ddnames on to any subcommands or problem programs running under them.
- Command processors should allow DAIR, the default service routine, or the parse service routine to prefix an identifier on the data set name so the PROFILE command's PREFIX and NOPREFIX options are automatically supported. You can use the default service routine to add any data set suffix that exists for the data set. (The default service routine is documented in *TSO Terminal Monitor Program and Service Routines Logic*).
- Whenever the user specifies a password for a data set, the command processor should send the password to DAIR when allocation is requested.
- Command processors should normally invoke DAIR to free all data sets at termination so other TSO users or submitted jobs can have full use of the data sets.
- Before detaching terminated subcommands, command processors that accept subcommands should use DAIR to free any data sets allocated by the subcommands.
- Command processors should use the DAIRFAIL service routine to issue meaningful error messages for non-zero return codes from DAIR.

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 because none are 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.

In the parse macro instructions that define command syntax, command processors and subcommand processors can indicate to the parse service routine that validity checking exits be taken on certain types of operands. Because the parse service routine checks the operands only for syntax errors, you should indicate in the parse macro instructions that validity checking routines be entered whenever a logical, rather than a syntactical, error might occur.

The GNRLFAIL service routine should be used to issue meaningful error messages for non-zero return codes from the parse service routine.

ESTAE/ESTAI Exit Routine -- Intercepting an ABEND

Use the ESTAE and ESTAI exits in your command processors, if they are needed, to keep the system operable if abnormal termination occurs. ESTAE/ESTAI exits should be used in such a way that the command processor gets control if a subcommand abnormally terminates. If you issue an ESTAE, issue it as early as possible in your command processor. Any ESTAE should be issued before any STAX. ESTAE provides the command processor with the ability to intercept an ABEND so that cleanup, bypass, and if possible, execution retry can be accomplished. (See *SPL: System Macros and Facilities* for a discussion of the ESTAE macro instruction. See *Supervisor Services and Macro Instructions* for a discussion of the ESTAI operand of the ATTACH macro instruction and for information about ESTAE and ESTAI exit routines.)

Linkage Considerations

Programs may issue the ESTAE and FESTAE macros, as well as the ATTACH macro with the ESTAI operand, in either 24- or 31-bit addressing mode. The ESTAE, FESTAE, and ESTAI recovery routines receive control in the same addressing mode in which the ESTAE, FESTAE, and ATTACH macros are issued. When the macros are issued in 31-bit addressing mode, ESTAE, FESTAE, and ESTAI routines may reside above the 16-megabyte virtual storage line.

The ESTAE, FESTAE, and ATTACH macros are downward incompatible. The MVS/Extended Architecture versions of these macros do not execute properly in 370 mode. For an explanation of how to select the desired macro level, see *SPL: System Macros and Facilities*.

While not recommended, the STAE macro and the STAI operand of ATTACH may be used to provide error handling exits. However, programs executing in 31-bit addressing should **not** establish STAE or STAI recovery exits.

Command Processor Functions that Rely on Exit Routine Support

The following types of command processors should use ESTAE 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 an ESTAI exit to intercept abnormally terminating subcommand processors.

Simple command or subcommand processors should not issue an ESTAE or ESTAI if the terminal monitor program or calling command processor ESTAI exits provide adequate processing.

Guidelines for ESTAE and ESTAI Exit Routines

ESTAE and ESTAI exit routines should observe the following guidelines:

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

   1st level $\left\{\begin{array}{l}\text{command-name}\\\text{subcommand-name}\end{array}\right\}$ ENDED DUE TO ERROR+

   2nd level COMPLETION CODE IS xxxx

   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 ESTAE or ESTAI exit from ABEND. The GNRLFAIL service routine may be used to issue the diagnostic error message, although it requires additional storage space (see guideline number 4).

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

   When an ABEND is intercepted, the command processor ESTAE exit routine should determine whether retry is to be attempted. If so, the exit routine should issue the diagnostic message and return, indicating via a return code that an ESTAE retry routine is available. If a retry is not to be attempted, the exit routine should return, indicating via a return code that no retry is to be attempted. The TMP ESTAI 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 ESTAE or ESTAI routine that receives control from ABEND should perform all necessary steps to provide system cleanup. This cleanup should be performed in the ESTAE exit routine rather than in the ESTAE retry routine because DETACH with the ESTAE=YES operand does not allow the subtask to retry from an ESTAE/ESTAI exit. (The TMP issues DETACH with ESTAE=YES when a command processor has been interrupted with an attention.)

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 ESTAE exit should not specify the RETRY option.

4. Storage might not be available when the ESTAE or ESTAI routine receives control. Any storage the routine requires should be acquired before it receives control, and be passed to it.

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 "Attention Interruption Handling - The STAX Service Routine" for a complete discussion of the STAX macro instruction. Simple command processors should not issue a

STAX if the TMP or the calling command processor STAX exits provide adequate processing.

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 messages, 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 second level informational message chain (if one exists) to be written to the terminal, puts out the mode message again, 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 second 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:

- Post the command processor's event control block to cause active service routines to return to the command processor.
- Exit.
- Reset the input stack in the command processor mainline. (A stack flush in an attention routine may cause severe errors.)

The HELP Data Set

A terminal user can enter the HELP command to retrieve information about commands or subcommands. This information is stored in a data set labeled SYS1.HELP (the HELP data set). If you add command processors to TSO, you should either add HELP information to the SYS1.HELP data set, or to a private HELP data set.

Attributes of SYS1.HELP

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 punch-card images. The logical record length is therefore 80 characters.

Format of HELP Members

Each of the 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 appears only if the command has subcommands.
- Functional description - This provides a brief description of the function of the command or subcommand.
- Syntax - This describes the syntax of the command or subcommand.
- Message identifier description - This provides information pertaining to messages issued by the command or its subcommand.
- Operand description - This provides information on the command positional operands, followed by individual sections containing brief descriptions of each keyword and its parameters.

Private HELP Data Sets

You may concatenate your data set to the SYS1.HELP data set (or vice versa). Concatenated data sets need not have the same attributes as the SYS1.HELP data set, but the first concatenated data set must have the largest blocksize of the concatenated data sets, and it must not specify a fixed blocksize.

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.

Private HELP data sets must be allocated with filename SYSHELP, either in the LOGON procedure or on an ALLOCATE command. When data sets are concatenated, the filename SYSHELP is required. If only SYS1.HELP is required, the filename SYSHELP would not have to be allocated. (See the DAIR entry code X'24' later in this book.)

Updating SYS1.HELP

Use the IEBUPDTE utility program or the EDIT command to update SYS1.HELP. SYS1.HELP is a system data set, so it will generally require operator intervention when it is updated.

Writing HELP Members

To add a new member to a data set named PRIVATE.HELP using the EDIT command, enter:

```
edit 'private.help(mbrname)' data new
```

Use the information described in Figure 8 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 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.
)M	This card indicates that message ID information follows. The information is only printed by the HELP command when the MSGID keyword is specified.
))messageid	This card indicates that information follows describing the named messageid. One of these control cards should be present for each message issued by the command. Each card contains the identifier of the message it describes. Message IDs can be any length and the first character must be alphabetic.
)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 and the alias.

Figure 8. Format of 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.

The only restrictions on data cards are that columns 72-80 are reserved for sequence numbers, and column one must contain a right parenthesis, an equal sign, or a blank. The sequence numbers are not printed when the HELP command is executed.

The OPERAND cards must be the last section of the HELP member. The)O card may only be followed by the)) or = cards.

For example, information concerning a user's SAMPLE command, shown in Figure 9, could be formatted for entry into the HELP data set (or your own private help data set).

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

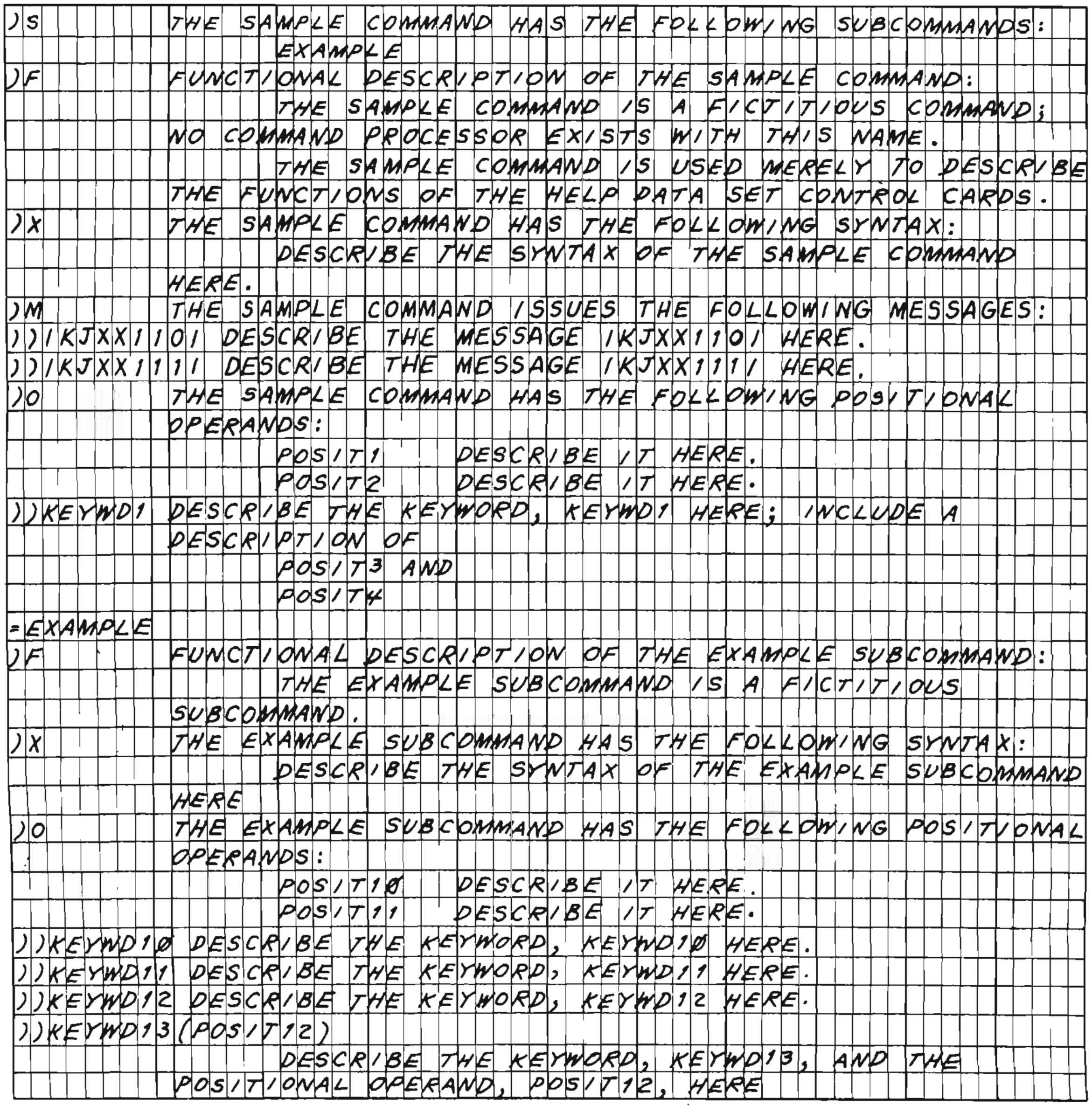
Figure 9. An Example of a User's SAMPLE Command Format

The SAMPLE command has one subcommand, the EXAMPLE subcommand (see Figure 11). Both the command and the subcommand can issue messages IKJXX110I and IKJXX111I.

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

Figure 10. An Example of a User's EXAMPLE Subcommand Format

Figure 11 shows data cards that would present and format information about the SAMPLE command and EXAMPLE subcommand 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.
)M       THE SAMPLE COMMAND ISSUES THE FOLLOWING MESSAGES:
))IKJXX1101 DESCRIBE THE MESSAGE IKJXX1101 HERE.
))IKJXX1111 DESCRIBE THE MESSAGE IKJXX1111 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:
               POSIT1Ø   DESCRIBE IT HERE.
               POSIT11   DESCRIBE IT HERE.
))KEYWD1Ø DESCRIBE THE KEYWORD, KEYWD1Ø 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 and EXAMPLE Subcommand in the HELP Data Set

MVS/Extended Architecture Considerations

This section discusses considerations for MVS/Extended Architecture in terms of its impact on the tasks documented in this manual. You should be familiar with the publications that describe comprehensive programming considerations for MVS/Extended Architecture, as well as with those that describe the routines and macros discussed in this manual. Henceforth, MVS/Extended Architecture is referred to as MVS/XA.

Note: Interfaces for service routines and macro instructions mentioned in this section are covered in more detail in the sections of this manual describing the individual service routines and macro instructions.

Testing a Program

MVS/XA users cannot use the TEST command to test a program unless TSO Extensions (TSO/E 5665-293) is installed. If TSO/E is not installed and a user enters the TEST command, he receives a message informing him that the command is not recognized.

31-Bit Addressing - General Interface Considerations

The interfaces described in this section reflect what is *possible* on an MVS/XA system. When determining the attributes and linkage conventions for a program, you should analyze the program's individual interfaces and its overall interactions with other programs. This section provides general guidelines for making these determinations.

MVS/XA requires that addressing modes and program residency be considered when determining linkage conventions. See "Specific Interfaces and Functions" later in this section for brief descriptions of those considerations for the service routines and macro instructions described in this manual.

Assuming you are running programs on an MVS/XA system, you may wish to take advantage of the added virtual storage provided by extended addressing, or you may wish to prepare for doing so in the future. Before describing linkage considerations, it is important to note that if a program is to be run on MVS/370 systems or on both MVS/370 and MVS/XA systems, it cannot perform any functions unique to MVS/XA.

Some MVS/XA macro instructions are downward incompatible; their MVS/XA expansions do not function correctly in MVS/370. Of the macros discussed in this manual, ATTACH, ESTAE, FESTAE, and STAX are downward incompatible. For a description of how to generate the desired level of a macro instruction, refer to *SPL: System Macros and Facilities.*

When making linkage decisions, you should analyze:

- Who passes control to whom
- Whether return is desired
- AMODE and RMODE attributes

The first two items are discussed in *SPL: 31-Bit Addressing.*

The following discussion provides a general description of AMODE and RMODE attributes; it does not attempt to cover AMODE and RMODE considerations in depth. For a detailed discussion of 31-bit addressing, refer to *SPL: 31-Bit Addressing.*

The following paragraphs pertain to programs running exclusively in 370-XA mode.

AMODE=24, RMODE=24

Programs with these attributes expect to (or are designed to) receive control in 24-bit addressing mode, and are loaded below 16 megabytes.

If you do not assign AMODE and RMODE attributes to a program, the attributes default to AMODE=24 and RMODE=24.

The IBM-supplied terminal monitor program and command processors have these attributes, and are loaded below 16 megabytes.

AMODE=ANY, RMODE=24

AMODE=ANY indicates that a program expects to (or is designed to) receive control in the addressing mode of the program that invoked it. Note that a program with the AMODE=ANY attribute may have to switch addressing modes for certain processing. However, such a program must switch back to the addressing mode in which it received control before returning to the caller.

AMODE=ANY programs must be given the RMODE=24 attribute.

AMODE=ANY does not indicate whether the program should be passed input that resides below 16 megabytes; the particular interfaces should be analyzed to determine where input may reside. However, a program should meet certain criteria in order to be assigned the AMODE=ANY attribute. Refer to *SPL:31-Bit Addressing* for a description of the criteria.

AMODE=31

AMODE=31 indicates that a program expects to (or is designed to) receive control in 31-bit addressing mode. Such a program may have the RMODE=24 or RMODE=ANY attribute, depending on its residency requirements. Regardless of the program's RMODE attribute, the residency of its input depends on the program's requirements. The program may require that some of its input reside below 16 megabytes, while other input may reside anywhere.

A program that runs exclusively in 31-bit addressing mode (AMODE=31) may do so provided it complies with the restrictions of invoking, and being invoked by, programs that run in 24-bit addressing mode (AMODE=24 or AMODE=ANY).

Refer to *SPL: 31-Bit Addressing* for more information on the AMODE=31 attribute.

Specific Interfaces and Functions

The interfaces described in this section reflect what is *possible* on an MVS/XA system. When determining the attributes and linkage conventions for a program, you should analyze the program's individual interfaces and its overall interactions with other programs. This section provides specific guidelines for making these determinations.

Control Program Interfaces

The IBM-supplied command processors are loaded below 16 megabytes and must receive control in 24-bit addressing mode.

The command processor parameter list (CPPL) passed by IBM-supplied control programs resides below 16 megabytes.

User-written TMPs and CPs may execute in either 24- or 31- bit addressing mode provided they honor the restrictions involved in invoking programs that have 24-bit dependencies.

Service Routine Interfaces

The following routines execute, and must receive control, in 24-bit addressing mode. All input passed to these routines must reside below 16 megabytes:

IKJEBEPS	data type processor
IKJEFF02	TSO message issuer routine
IKJEFF18	DAIRFAIL
IKJEFF19	GNRLFAIL/VSAMFAIL
IKJEHSIR	STA interface routine
IKJGETL	GETLINE
IKJPARS	parse service routine
IKJPTGT	PUTGET
IKJPUTL	PUTLINE
IKJSCAN	command scan service routine
IKJSTCK	STACK

If a program running in 31-bit addressing mode invokes any one of these routines, the LINK macro should be used to invoke it because LINK does not require the invoking program to switch to 24-bit addressing mode. In this case, LINK switches to 24-bit mode in behalf of the invoking program. If a program is loaded above 16 megabytes, it *must* use LINK to invoke one of these routines. LINK returns control to the invoking program in the same mode in which the LINK macro is issued.

A program that is running in 31-bit addressing mode and is loaded below 16 megabytes may issue CALLTSSR to invoke IKJEFF02, IKJPARS, or IKJSCAN, provided the program switches to 24-bit addressing mode before issuing CALLTSSR. Control is returned to the invoking program in 24-bit addressing mode. If the requested routine has not been placed in the link pack area, CALLTSSR generates a LINK macro to invoke it. In this case, the invoking program may issue CALLTSSR in either addressing mode. Note that CALLTSSR can be used to invoke only certain *routines*.

A program running in 24-bit addressing mode may invoke these 11 routines using any of the methods suggested in "Passing Control to the TSO Service Routines."

IKJEHCIR (catalog information routine) and IKJEHDEF (default service routine) may be invoked in either addressing mode, but all input passed to these routines must reside below 16 megabytes. These routines execute in 24-bit addressing mode and return control in the same addressing mode in which they are invoked.

STAX (specify terminal attention exit routine) may be invoked in either 24- or 31-bit addressing mode. Refer to "Attention Interruption Handling -- The STAX Service Routine" for more information.

IKJDAIR (dynamic allocation interface routine) may be invoked in either 24- or 31-bit addressing mode. When invoked in 31-bit addressing mode, IKJDAIR may be passed input that resides above 16 megabytes. IKJDAIR returns control in the same addressing mode in which it is invoked.

Macro Interfaces

Figure 12 shows the MVS/XA rules for the macros discussed in this manual.

Note: In Figure 12, a dash (-) indicates that the category does not apply to the macro because the macro does not generate executable code. The addressing mode of the program that accesses the date generated by the macro must agree with the residence of the data.

Macro	(X) May Be Issued In		(P) May Be Issued by Program (I) Input May Be	
	24-Bit Mode	31-Bit Mode	Below 16 Mb	Above 16 Mb
ATTACH	X	X	I, P	I, P
CALL	X	X	I, P	I, P
CALLTSSR	X	X	P	P
ESTAE	X	X	I, P	I, P
FESTAE	X	X	I, P	I, P
GETLINE	X	X	I, P	
GTSIZE	X	X	P	P
GTTERM	X		P	
IKJENDP	–	–	P	P
IKJIDENT	–	–	P	P
IKJKEYWD	–	–	P	P
IKJNAME	–	–	P	P
IKJOPER	–	–	P	P
IKJPARM	–	–	P	P
IKJPOSIT	–	–	P	P
IKJRLSA	X	X	P	P
IKJRSVWD	–	–	P	P
IKJSUBF	–	–	P	P
IKJTERM	–	–	P	P
IKJTSMSG	–	–	P	P
LINK	X	X	I, P	I, P
LOAD	X	X	I, P	I, P
PUTGET	X	X	I, P	
PUTLINE	X	X	I, P	
RTAUTOPT	X	X	P	P
SAM Macros	X		I, P	
SPAUTOPT	X	X	P	P
STACK	X	X	I, P	
STAE	X		I, P	

Figure 12. MVS/XA Interface Rules for Macro Instructions (Part 1 of 2)

Macro	(X) May Be Issued In		(P) May Be Issued by Program (I) Input May Be	
	24-Bit Mode	31-Bit Mode	Below 16Mb	Above 16Mb
STATTN	X		I, P	
STAUTOCP	X	X	P	P
STAUTOLN	X		I, P	
STAX	X	X	I, P	See section on STAX.
STBREAK	X		I, P	
STCC	X		I, P	
STCLEAR	X		I, P	
STCOM	X		I, P	
STFSMODE	X		I, P	
STLINENO	X		I, P	
STSIZE	X		I, P	
STTIMEOU	X		I, P	
STTMPMD	X		I, P	
STTRAN	X		I, P	
TCLEARQ	X		I, P	
TGET	X		I, P	
TPG	X		I, P	
TPUT	X		I, P	
XCTL	X	X	I, P	I, P

Figure 12. MVS/XA Interface Rules for Macro Instructions (Part 2 of 2)

Notes on Figure 12

ATTACH, LINK, LOAD, XCTL

A program may issue the ATTACH, LINK, LOAD, and XCTL macro instructions while executing in either 24- or 31-bit addressing mode. These system services determine where to load the requested program in storage and in which addressing mode to invoke it based on the program's AMODE and RMODE attributes. Note that LOAD only loads a program; it does not invoke the program. LOAD returns the address of the loaded program. The high-order bit of this address reflects the AMODE attribute of the loaded program.

If a program is invoked via a LINK, ATTACH, or XCTL macro, it receives control in the addressing mode specified or allowed by its AMODE attribute. On the other hand, if a program branches to another program without changing addressing modes via the BASSM or BSM branch instructions, the requested program receives control in whatever addressing mode is active at the time of the branch -- that is, in the addressing mode of the caller. For more information on these macros, refer to *System Macros and Facilities.*

CALL

You may use the CALL macro to invoke a program if that program may be invoked in the current addressing mode.

CALLTSSR

The CALLTSSR macro instruction may be issued in either 24- or 31-bit addressing mode. See "Passing Control to the TSO Service Routines" later in this book for more information on issuing the CALLTSSR macro.

ESTAE, FESTAE, STAE, ESTAI

The ESTAE and FESTAE macros may be issued in either 24- or 31-bit addressing mode. Refer to "ESTAE/ESTAI Exit Routines -- Intercepting an ABEND" for more information. Use of the STAE macro and the ESTAI operand on the ATTACH macro to establish recovery exits and routines is not recommended. If they are used, the recovery exits and routines must receive control in 24-bit addressing mode -- that is the STAE and ATTACH macros must be issued in 24-bit addressing mode.

ESTAI

See ESTAE.

FESTAE

See ESTAE.

GETLINE, PUTGET, PUTLINE, STACK

The GETLINE, PUTGET, PUTLINE, and STACK macros must be issued in 24-bit addressing mode.

IKJTSMSG

The IKJTSMSG macro must be issued by a program loaded below 16 megabytes.

LINK

See ATTACH.

LOAD

See ATTACH.

Parse Macros

The parse parameter list passed to the parse service routine must reside below 16 megabytes. As a result, the parse macro instructions that generate input to parse must be issued by a program loaded below 16 megabytes. See Figure 12 for a list of the parse macros and their linkage requirements. The IKJRLSA parse macro must be issued in 24-bit addressing mode.

PUTGET

See GETLINE.

PUTLINE

See GETLINE.

SAM Macros

The sequential access method (SAM) terminal macro instructions must be issued in 24-bit addressing mode.

STACK

See GETLINE.

STAE

See ESTAE.

STAX

A program may issue the STAX macro in either 24- or 31-bit addressing mode. Refer to "Specifying a Terminal Attention Exit -- The STAX Macro Instruction" for specific restrictions.

SVC 93 (TGET, TPUT, TPG)

SVC93 (TGET, TPUT, and TPG macros) executes in 24-bit addressing mode. Programs must issue TGET, TPUT, and TPG in 24-bit addressing mode.

SVC 94 (Terminal Control Macros)

SVC 94 (terminal control macros) executes in 24-bit addressing mode. With a few exceptions, terminal control macros must be issued in 24-bit addressing mode. The exceptions are the GTSIZE, RTAUTOPT, SPAUTOPT, and STAUTOCP terminal control macros, which may be issued in 31-bit addressing mode. See Figure 12 for a list of the terminal control macros and their linkage requirements.

XCTL

See ATTACH.

Processing Terminal Requests - The TSO Service Routines

The TSO service routines process terminal 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 IBM-supplied terminal monitor program with one of your own design, you should use the service routines to process terminal requests.

The TSO service routines build, modify, or make use of various control blocks. The following control block DSECTS are provided in SYS1.MACLIB for your use, and are listed in *Data Areas*.

IKJCPPL	The command processor parameter list
IKJCSOA	The command scan output area
IKJCSPL	The command scan parameter list
IKJDAPL	The dynamic allocation parameter list
IKJDAP0C	DAIR entry code parameter block
IKJDAP00	DAIR entry code parameter block
IKJDAP04	DAIR entry code parameter block
IKJDAP08	DAIR entry code parameter block
IKJDAP1C	DAIR entry code parameter block
IKJDAP10	DAIR entry code parameter block
IKJDAP14	DAIR entry code parameter block
IKJDAP18	DAIR entry code parameter block
IKJDAP2C	DAIR entry code parameter block
IKJDAP24	DAIR entry code parameter block
IKJDAP28	DAIR entry code parameter block
IKJDAP30	DAIR entry code parameter block
IKJDAP34	DAIR entry code parameter block
IKJDFPB	The default parameter block
IKJDFPL	The default parameter list
IKJECT	The environment control table for GETLINE/PUTLINE/PUTGET/STACK
IKJEFFDF	PARMLIST to IKJEFF18 (DAIRFAIL)
IKJEFFGF	PARMLIST to IKJEFF19 (GNRLFAIL)
IKJEFFMT	PARMLIST to IKJEFF02
IKJGTPB	The GETLINE parameter block
IKJIOPL	The input output parameter list for GETLINE/PUTLINE/PUTGET/STACK
IKJLSD	The list source descriptor for STACK
IKJPGPB	The PUTGET parameter block
IKJPPL	Defines the parse parameter list
IKJPSCB	The protected step control block
IKJPTPB	The PUTLINE parameter block
IKJSTPB	The STACK parameter block
IKJSTPL	The STACK parameter list
IKJTAIE	Terminal attention interrupt element from STAX
IKJTAXE	Terminal attention exit element from STAX
IKJTMPWA	The terminal monitor program work area
IKJUPT	User profile table

Interfacing with the TSO 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. The control block interface between the TMP and an attached CP is shown in Figure 13.

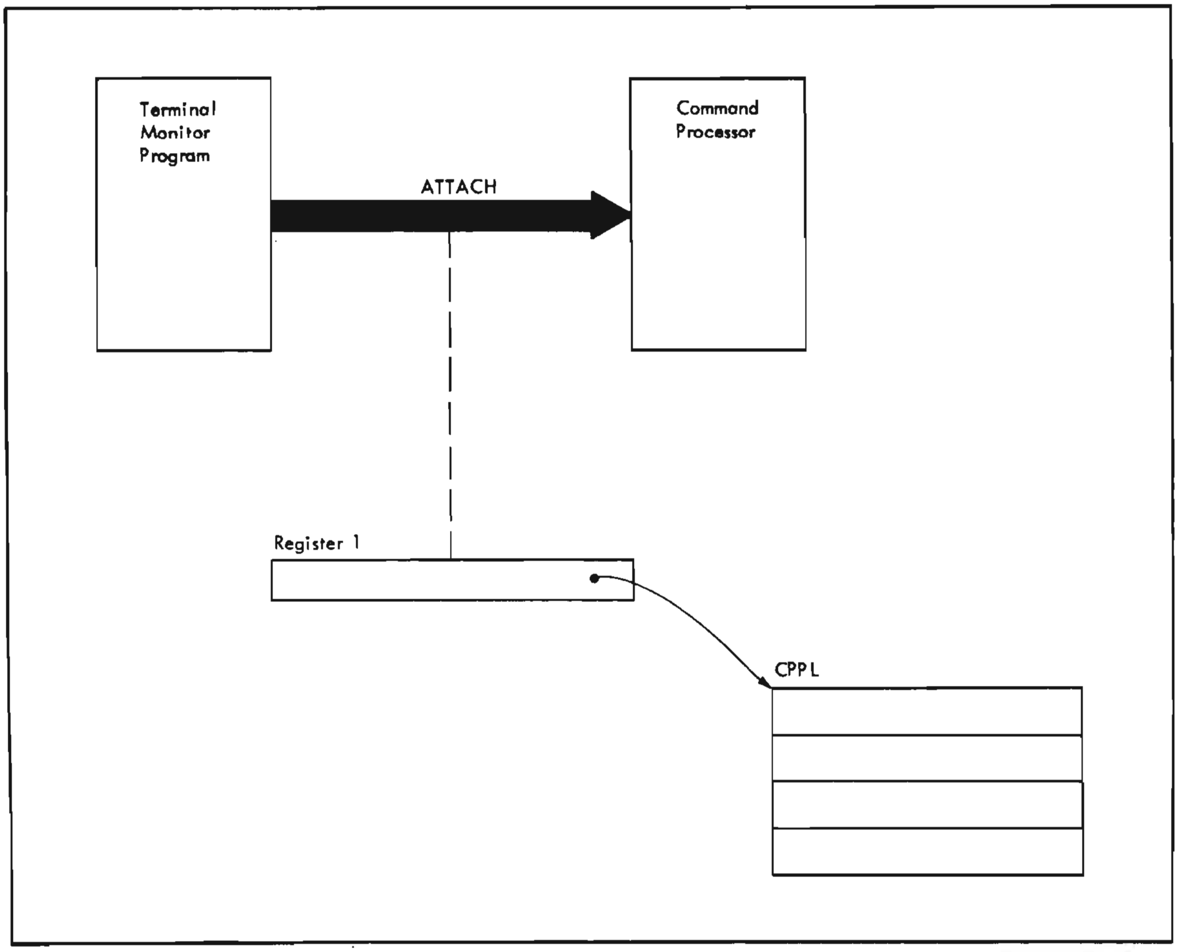


Figure 13. Control Block Interface between the TMP and CP

The Command Processor Parameter List

You must pass certain addresses contained in the CPPL to the TSO 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 14 describes the contents of the CPPL.

When the TMP invokes a problem program, whether a command processor or not, register 1 contains the address of the CPPL. The CPPL is required by any program that is going to use TSO service routines. If any problem program or user-written command processor is going to invoke an IBM-supplied command processor, the CPPL address must be supplied in register 1.

All input passed to IBM-supplied command processors and the IBM-supplied TMP must reside below 16 megabytes.

Number of Bytes	Field	Contents or Meaning
4	CPPLCBUF	The address of the command buffer for the currently attached command processor.
4	CPPLUPT	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 obtained from the PSCBUPT field of the protected step control block (PSCB).
4	CPPLPSCB	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 using the EXTRACT macro instruction.
4	CPPLECT	The address of the environment control table (ECT). The ECT must be built by the TMP during its initialization process; it is used by the TSO service routines and by some TSO commands and subcommands.

Figure 14. The Command Processor Parameter List (CPPL)

For a description of linkage and program residency considerations, refer to "Service Routine Interfaces" in the previous section.

Passing Control to the TSO Service Routines

There are four ways you can pass control to the TSO service routines.

1. You can issue an I/O macro instruction without the ENTRY parameter for the I/O service routines.

2. You can issue a LINK instruction to a service routine, but this requires more system overhead than other methods.

   The LINK macro instruction loads the routine in storage based on its RMODE attribute, and passes control to the routine in the addressing mode specified or allowed by its AMODE attribute.

3. You can issue a LOAD instruction for a service routine and then do branches to the loaded routine, but this also requires more system overhead than other methods.

   The LOAD macro loads the routine in storage based on its RMODE attribute. LOAD returns the address of the loaded program. The

Message Handling

TSO messages are divided into three classes:

- Prompting messages
- Mode messages
- Informational messages

Prompting messages begin with ENTER or REENTER, and require a response from the user. Prompting messages should be initiated by the parse service routine, rather than by parse validity check exits, using the text supplied by the command processor as the PROMPT operand of the IKJPOSIT, IKJTERM, IKJOPER, IKJRSVWD or IKJIDENT parse macro instructions. See "Using the Parse Service Routine (IKJPARS)" for a discussion of the PROMPT operand on these macro instructions.

Mode messages are the READY messages 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 is in control and let him know that the system is waiting for him to enter a new command or subcommand.

Informational messages do 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. The TPUT routine does not support multi-level messages, message id stripping, and text insertion, and does not function in background mode (it acts as a NOP).

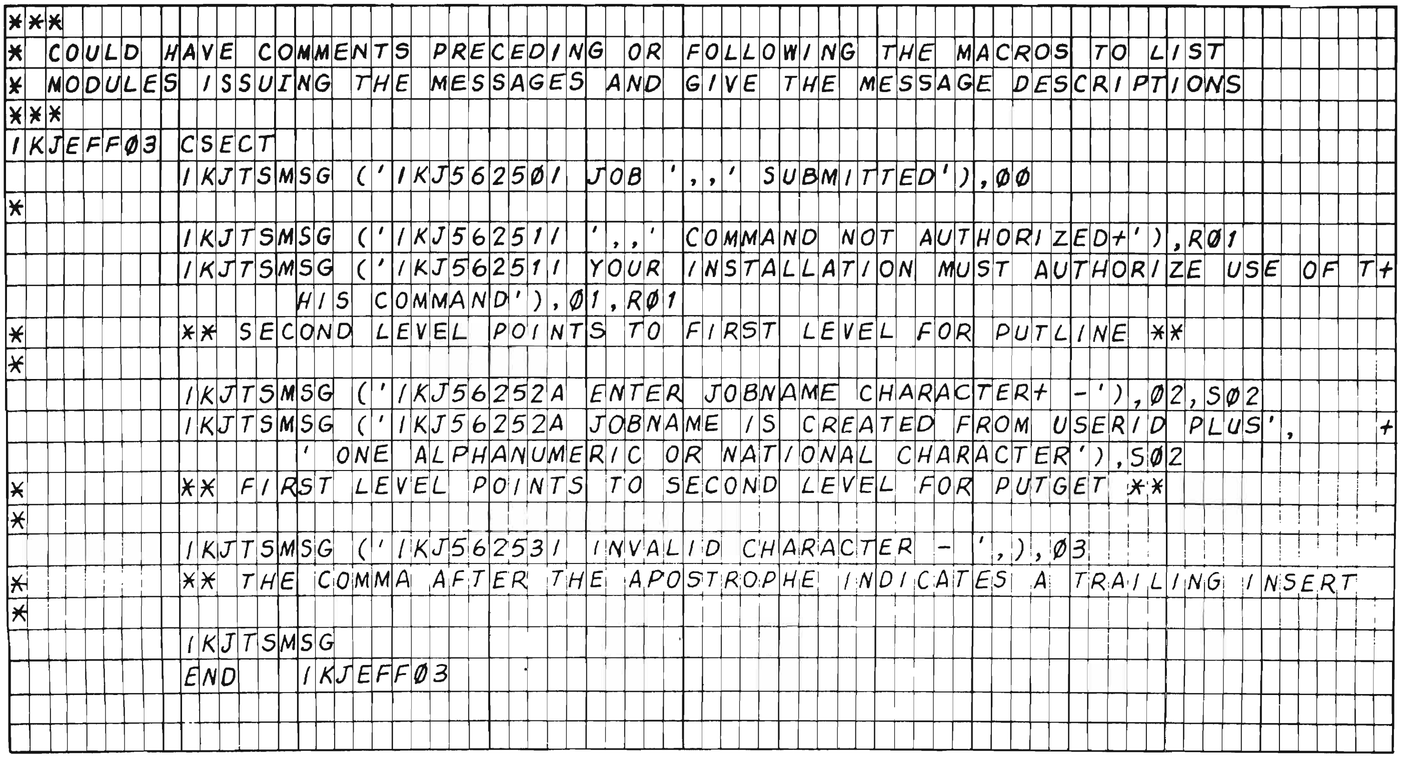
Message Levels

Messages usually should have associated with them other messages that more fully explain the initial message. These messages, called second level messages, 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 messages 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

- Message IKJ56250I is a single level PUTLINE message with one insert.
- Message IKJ56251I is a PUTLINE message with two levels.
- Message IKJ56252A is a PUTGET message with two levels.
- Message IKJ56253I is a PUTLINE message with an insert at the end of the text.
- The IKJTSMSG macro with no operands is required to indicate the end of the message CSECT.



```
***
* COULD HAVE COMMENTS PRECEDING OR FOLLOWING THE MACROS TO LIST
* MODULES ISSUING THE MESSAGES AND GIVE THE MESSAGE DESCRIPTIONS
***
IKJEFFØ3 CSECT
         IKJTSMSG ('IKJ5625ØI JOB ',' SUBMITTED'),ØØ
*
         IKJTSMSG ('IKJ56251I ',' COMMAND NOT AUTHORIZED+'),RØ1
         IKJTSMSG ('IKJ56251I YOUR INSTALLATION MUST AUTHORIZE USE OF T+
               HIS COMMAND'),Ø1,RØ1
*        ** SECOND LEVEL POINTS TO FIRST LEVEL FOR PUTLINE **
*
         IKJTSMSG ('IKJ56252A ENTER JOBNAME CHARACTER+ -'),Ø2,SØ2
         IKJTSMSG ('IKJ56252A JOBNAME IS CREATED FROM USERID PLUS',    +
               ' ONE ALPHANUMERIC OR NATIONAL CHARACTER'),SØ2
*        ** FIRST LEVEL POINTS TO SECOND LEVEL FOR PUTGET **
*
         IKJTSMSG ('IKJ56253I INVALID CHARACTER - ',),Ø3
*        ** THE COMMA AFTER THE APOSTROPHE INDICATES A TRAILING INSERT
*
         IKJTSMSG
         END   IKJEFFØ3
```

Figure 17. An Example of an IKJTSMSG Macro Instruction

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. Simple command or subcommand procedures should not issue a STAX macro instruction unless the STAX routine specified by the TMP or the calling command processor cannot process an attention interruption adequately.

The STAX service routine may be invoked in either 24-bit or 31-bit addressing mode. The attention exit routine receives control in the same addressing mode in which the respective STAX macro is issued.

With the exception of the TPUT ASID buffers for TCAM, when the user enters an attention interruption from the terminal, the TGET, TPUT, and TPG 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.

Attention processing gives the user the ability to specify exit routines that receive control asynchronously when the attention key is struck or when an interruption occurs as a result of the simulated attention facility (STATTN macro). The mechanism used to request attention exits is the STAX macro. When the STAX macro is issued, a TAXE (*terminal* attention exit element) is created and placed on a queue. The TAXE queue is ordered according to the attention level, and the attention level determines the order in which the attention exits are given control. If the ATTENTION key is struck once, the first level attention exit is given control. If the key is struck twice, the second level attention exit is given control. When placing a TAXE on the TAXE queue, two rules apply:

1. An attention exit routine for a task will always occupy a higher attention level than the attention exit of any of its subtasks.
2. The attention exit routine is placed at the lowest possible attention level, without violating the first rule.

In other words, the placement at an attention level is determined by the position of the task in the subtask queue relative to the position of the other tasks creating attention exits. The lower the subtask the lower the attention level assigned. The subtask queue is considered to be the mother-daughter queue only. If for any reason a complex task structure is created that would have a mother task with multiple daughter tasks, then the order in which the daughters issue STAX macros must be synchronized in order to ensure predictability from day to day. Note that the order in which the daughters issue STAX macros, not the order in which the daughters are attached, determines the order in which the associated TAXEs are placed on the TAXE queue.

If a task has issued multiple STAX macros, the order in which the associated TAXE is placed on the TAXE queue is determined by the second rule.

Attention levels can change during execution of the session for three reasons:

1. A task has issued STAX and its daughter then issues STAX. In this case the attention exit for the first task would have an attention level of one until its subtask had issued STAX. The daughter task would then have an attention level of one and the original task would have a level of two.
2. A task that has established an attention exit environment abnormally terminates or exits. When this occurs the TAXEs for that task are freed. The remaining TAXEs then assume the new attention level relative to its position on the TAXE queue.
3. The STAX macro is used to cancel the last attention exit established by a task.

When generating an attention interruption by striking the ATTENTION key, the ATTENTION level is recorded by counting the number of times the ATTENTION key has been struck. If the number of times the key is struck exceeds the number of available attention levels, an"!I" message is sent to the terminal. If the attention has been accepted, an "!" message is sent to the terminal to indicate that the attention exit is being scheduled. If an attention interruption is received while a previously requested (lower attention level) attention exit is in the process of being scheduled, the first attention exit is canceled and the new attention exit is scheduled. This will be true until control has been passed to the user's attention exit.

Prior to passing control to the attention exit, the task under which the attention exit is running will have all its subtasks stopped. Note, however, that if a system routine (SVRB on RB chain) is executing for one of the TCBs and has not specified STAX DEFER=NO (see below for expanded explanation), then the scheduling of the attention exit will be deferred until the completion of such system routines. All SVRBs start execution in a STAX DEFER=YES state and all other RBs start execution in a STAX DEFER=NO state. Consequently, the presence of an SVRB on a TCB's RB chain normally means attention exits will be deferred. When the user's attention exit completes processing the subtasks are automatically restarted. If, for any reason, the attention routine requires one of the subtasks to be restarted, it is the responsibility of the attention exit to restart the task through the use of the status start facility. If the subtasks should not be restarted, it is the responsibility of the attention exit to use the status stop facility to ensure that the subtasks will not become dispatchable when the attention exit completes processing. See *Supervisor Services and Macro Instructions* for additional information.

The attention level at which the attention exit is running and all of the lower attention levels are considered unavailable as soon as scheduling of the exit takes place. Therefore, once the attention scheduling has begun, only higher attention levels are available for use until the attention exit completes processing.

Message Handling

TSO messages are divided into three classes:

- Prompting messages
- Mode messages
- Informational messages

Prompting messages begin with ENTER or REENTER, and require a response from the user. Prompting messages should be initiated by the parse service routine, rather than by parse validity check exits, using the text supplied by the command processor as the PROMPT operand of the IKJPOSIT, IKJTERM, IKJOPER, IKJRSVWD or IKJIDENT parse macro instructions. See "Using the Parse Service Routine (IKJPARS)" for a discussion of the PROMPT operand on these macro instructions.

Mode messages are the READY messages 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 is in control and let him know that the system is waiting for him to enter a new command or subcommand.

Informational messages do 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. The TPUT routine does not support multi-level messages, message id stripping, and text insertion, and does not function in background mode (it acts as a NOP).

Message Levels

Messages usually should have associated with them other messages that more fully explain the initial message. These messages, called second level messages, 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 messages 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 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 without the prompt option 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, the parse service routine issues an informational message to the terminal, and returns an error code 4 to its caller. 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 informational 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 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.

TSO Message Issuer Routine (IKJEFF02)

The TSO message issuer routine issues a message as a PUTLINE, PUTGET, write-to-operator (WTO), or a write-to-programmer (WTP). You may invoke IKJEFF02 just to issue the message to the terminal, both to issue the message and return the requested message to the caller in the caller's buffers, or just to return the message to the caller. This process of returning the message is referred to as extracting the message. This routine simplifies the issuing of messages with inserts because hexadecimal inserts can be converted to printable characters and the same parameter list used to issue any message. It also makes it more convenient to place all messages for a command in a single CSECT or assembly module, which is important when message texts must be modified. Adding or updating a message is simpler when IKJEFF02 is used, rather than PUTLINE or PUTGET.

Refer to "Interfacing with the TSO Service Routines" earlier in this book for a description of the ways in which IKJEFF02 may be invoked. Regardless of the linkage method used, IKJEFF02 must receive control in 24-bit addressing mode. If a program invokes IKJEFF02 via the CALLTSSR macro and IKJEFF02 has been placed in the link pack area, the program should issue CALLTSSR in 24-bit addressing mode. If IKJEFF02 has not been placed in the link pack area, the program may issue CALLTSSR in either addressing mode. In this case, the LINK macro generated by CALLTSSR invokes IKJEFF02 in 24-bit addressing mode.

All input passed to IKJEFF02 must reside below the 16-megabyte virtual storage line.

Generally, you will invoke the TSO message issuer routine via the CALLTSSR or LINK macro, passing the address of the following parameter list in register 1:

Offset			
Dec	**Hex**	**Field Name**	**Contents**
0	0	MTPLPTR	Address of message description section of this parameter list. (The message description section begins with the MTCSECTP entry.)
4	4	MTCPPLP	Address of TMP's CPPL control block (required for PUTLINE or PUTGET).
8	8	MTECBP	Address of optional communications ECB for PUTLINE or PUTGET.
12	C	MTRESV1	Reserved.
12	C	MTHIGH	High-order bit of reserved field turned on for standard linkage.
16	10	MTCSECTP	Address of an assembly module or a CSECT containing IKJTSMSG macros that build message identifications and associated texts.
20	14	MTSW1	One byte field of switches

Offset				
Dec	Hex	Field Name	Contents	
		MTNOIDSW	1...	Message is printed; no messageid is needed.
		MTPUTLSW	.1..	Message issued as PUTLINE. (Message inserts for a second level message must be listed before inserts for a first level message.) If this bit is zero, message issued as a PUTGET, with second level message required and inserts for second level messages necessarily following inserts for first level messages.
		MTWTOSW	..1.	Message issued as a WTO. Default is PUTGET.
		MTHEXSW	...1	Number translations to printable hexadecimal rather than default of printable decimal.
		MTKEY1SW	.... 1...	Modeset from key 1 to key 0 before issuing a PUTLINE or PUTGET message. Default is no modeset.
		MTJOBISW	.... .1..	Blanks are compressed out of xx(yy) format inserts. Default is no compression.
		MTWTPSW	.... ..1.	Message issued as WTO with write-to-programmer routing code. Inserts are handled the same as for PUTLINE. Default is PUTGET.
		MTNHEXSW	1	Number translations to printable decimal, even if larger than X'FFFF'. Default is printable hex above X'FFFF'.
21	15	MTREPLYP	Address of reply from PUTGET. The reply text is preceded by a two-byte field containing length of text plus header field.	
24	18	MTSW2	One byte field of switches.	
		MT2OLDSW	1...	Field MTOLDPTR points to second level message already in PUTLINE/PUTGET (Output Line Descriptor) format. Default is IKJTSMSG format.
		MTDOMSW	.1..	Delete WTP or WTO messages from the display console, using the delete operator message macro.
		MTNOXQSW	..1.	Override default of X"around inserts converted to printable hex.
		MTNPLMSW	...1	Override default of error message if PUTLINE fails.
		MTPGMSW	.... 1...	Request an error message if PUTGET fails.
25	19	MTRESV2	Reserved.	
28	1C	MTOLDPTR	Pointer to O.L.D. for second level message, required if MT2OLDSW bit is on.	
32	20	MTRESV3	Reserved.	
36	24	MTRESV4	Reserved.	
40	28	MTMSGID	Message's identifier in message CSECT, padded with blanks on the right.	
44	2C	MTINSRTS	Insert information for message. The following two fields are supplied for each insert:	
44	2C	MTLEN	Length of an insert for the message.	
44	2C	MTHIGHL	High-order bit is on if translate the first 1-4 bytes of the insert from hexadecimal to character (printable hexadecimal or decimal depending on whether MTHEXSW is set to ON or OFF).	
44	2C	MTINSRT	Refers to an insert entry.	
45	2D	MTADDR	Address of an insert for the message.	

The return code from the TSO message issuer routine is contained in register 15 as follows:

0	- Message issued successfully.
76	- Error in parameter list. A diagnostic message is also issued.
Other	- PUTLINE or PUTGET return code.

The IKJEFFMT macro maps the input parameter list. Specify MTDSECT=YES option to obtain DSECT MTDSECTD instead of storage.

IKJTSMSG -- Describes Text and Insert Locations

The IKJTSMSG macro is used to generate assembler language DC instructions describing the text and locations of inserts for a message which may be issued by the TSO message issuer routine (IKJEFF02). All of the messages which a command issues should be grouped into an assembly module consisting entirely of IKJTSMSG macros preceded by a CSECT card and followed by an END card. The last IKJTSMSG macro in the CSECT must be a dummy entry with no operands.

The IKJTSMSG macro must be issued in by a program loaded below 16 megabytes. Figure 16 shows the syntax of the IKJTSMSG macro instruction; each of the operands is explained following the figure. Appendix A describes the notation used to define macro instructions.

```
[symbol]    IKJTSMSG ('msgid msgtext'),id1[,id2]
```

Figure 16. The IKJTSMSG Macro Instruction

msgid - The identifier which will be displayed when the message is issued.

msgtext - The text of the message. If an insert is necessary within the text of a message or at the end of a message, use the following rules:

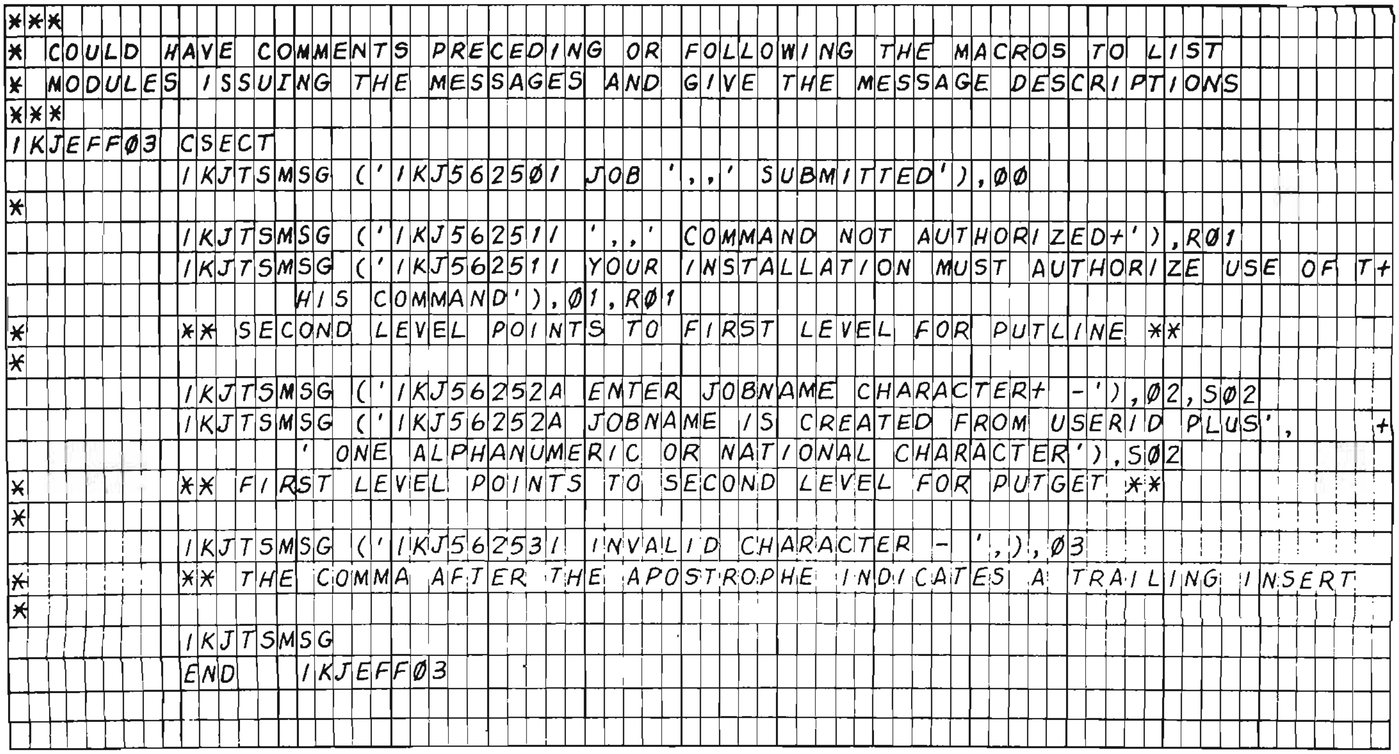
- The location of an insert in the middle of a message should be indicated by a ‘,,’.
- If the insert is to be located at the end of a message, indicate it by a ', following the message text.

id1 - The internal identifier of the message. It may be from one to four characters and should not contain a blank, comma, parentheses, or an apostrophe. This id is passed to IKJEFF02 in the MTMSGID field of the parameter list.

id2 - The internal identifier of a message to be chained to this message. For a PUTGET message, the first level message would have an id2 field identifying the second level, and the second level message could have an id2 field to identify a another second level, etc. For a PUTLINE, WTO, or write-to-programmer message, the second level message would have an id2 field identifying the first level.

For an example of the coding involved for a CSECT containing the IKJTSMSG macro, see Figure 17. The example shows how a message module can be created for a SUBMIT command, using the IKJTSMSG macro.

- Message IKJ56250I is a single level PUTLINE message with one insert.
- Message IKJ56251I is a PUTLINE message with two levels.
- Message IKJ56252A is a PUTGET message with two levels.
- Message IKJ56253I is a PUTLINE message with an insert at the end of the text.
- The IKJTSMSG macro with no operands is required to indicate the end of the message CSECT.



```
***
* COULD HAVE COMMENTS PRECEDING OR FOLLOWING THE MACROS TO LIST
* MODULES ISSUING THE MESSAGES AND GIVE THE MESSAGE DESCRIPTIONS
***
IKJEFFØ3 CSECT
         IKJTSMSG ('IKJ5625ØI JOB ',,' SUBMITTED'),ØØ
*
         IKJTSMSG ('IKJ56251I ',,' COMMAND NOT AUTHORIZED+'),RØ1
         IKJTSMSG ('IKJ56251I YOUR INSTALLATION MUST AUTHORIZE USE OF T+
               HIS COMMAND'),Ø1,RØ1
*        ** SECOND LEVEL POINTS TO FIRST LEVEL FOR PUTLINE **
*
         IKJTSMSG ('IKJ56252A ENTER JOBNAME CHARACTER+ -'),Ø2,SØ2
         IKJTSMSG ('IKJ56252A JOBNAME IS CREATED FROM USERID PLUS',    +
               ' ONE ALPHANUMERIC OR NATIONAL CHARACTER'),SØ2
*        ** FIRST LEVEL POINTS TO SECOND LEVEL FOR PUTGET **
*
         IKJTSMSG ('IKJ56253I INVALID CHARACTER - ',),Ø3
*        ** THE COMMA AFTER THE APOSTROPHE INDICATES A TRAILING INSERT
*
         IKJTSMSG
         END   IKJEFFØ3
```

Figure 17. An Example of an IKJTSMSG Macro Instruction

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. Simple command or subcommand procedures should not issue a STAX macro instruction unless the STAX routine specified by the TMP or the calling command processor cannot process an attention interruption adequately.

The STAX service routine may be invoked in either 24-bit or 31-bit addressing mode. The attention exit routine receives control in the same addressing mode in which the respective STAX macro is issued.

With the exception of the TPUT ASID buffers for TCAM, when the user enters an attention interruption from the terminal, the TGET, TPUT, and TPG 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.

Attention processing gives the user the ability to specify exit routines that receive control asynchronously when the attention key is struck or when an interruption occurs as a result of the simulated attention facility (STATTN macro). The mechanism used to request attention exits is the STAX macro. When the STAX macro is issued, a TAXE (terminal attention exit element) is created and placed on a queue. The *TAXE queue* is ordered according to the attention level, and the attention level determines the order in which the attention exits are given control. If the ATTENTION key is struck once, the first level attention exit is given control. If the key is struck twice, the second level attention exit is given control. When placing a TAXE on the TAXE queue, two rules apply:

1. An attention exit routine for a task will always occupy a higher attention level than the attention exit of any of its subtasks.
2. The attention exit routine is placed at the lowest possible attention level, without violating the first rule.

In other words, the placement at an attention level is determined by the position of the task in the subtask queue relative to the position of the other tasks creating attention exits. The lower the subtask the lower the attention level assigned. The subtask queue is considered to be the mother-daughter queue only. If for any reason a complex task structure is created that would have a mother task with multiple daughter tasks, then the order in which the daughters issue STAX macros must be synchronized in order to ensure predictability from day to day. Note that the order in which the daughters issue STAX macros, not the order in which the daughters are attached, determines the order in which the associated TAXEs are placed on the TAXE queue.

If a task has issued multiple STAX macros, the order in which the associated TAXE is placed on the TAXE queue is determined by the second rule.

Attention levels can change during execution of the session for three reasons:

1. A task has issued STAX and its daughter then issues STAX. In this case the attention exit for the first task would have an attention level of one until its subtask had issued STAX. The daughter task would then have an attention level of one and the original task would have a level of two.
2. A task that has established an attention exit environment abnormally terminates or exits. When this occurs the TAXEs for that task are freed. The remaining TAXEs then assume the new attention level relative to its position on the TAXE queue.
3. The STAX macro is used to cancel the last attention exit *established* by a task.

When generating an attention interruption by striking the ATTENTION key, the ATTENTION level is recorded by counting the number of times the ATTENTION key has been struck. If the number of times the key is struck exceeds the number of available attention levels, an"!I" message *is* sent to the terminal. If the attention has been accepted, an "!" message is sent to the terminal to indicate that the attention exit is being scheduled. If an attention interruption is received while a previously requested (lower attention level) attention exit is in the process of being scheduled, the first attention exit is canceled and the new attention exit is scheduled. This will be true until control has been passed to the user's attention exit.

Prior to passing control to the attention exit, the task under which the attention exit is running will have all its subtasks stopped. Note, however, that if a system routine (SVRB on RB chain) is executing for one of the TCBs and has not specified STAX DEFER=NO (see below for expanded explanation), then the scheduling of the attention exit will be deferred until the completion of such system routines. All SVRBs start execution in a STAX DEFER=YES state and all other RBs start execution in a STAX DEFER=NO state. Consequently, the presence of an SVRB on a TCB's RB chain normally means attention exits will be deferred. When the user's attention exit completes processing the subtasks are automatically restarted. If, for any reason, the attention routine requires one of the subtasks to be restarted, it is the responsibility of the attention exit to restart the task through the use of the status start facility. If the subtasks should not be restarted, it is the responsibility of the attention exit to use the status stop facility to ensure that the subtasks will not become dispatchable when the attention exit completes processing. See *Supervisor Services and Macro Instructions* for additional information.

The attention level at which the attention exit is running and all of the lower attention levels are considered unavailable as soon as scheduling of the exit takes place. Therefore, once the attention scheduling has begun, only higher attention levels are available for use until the attention exit completes processing.

You can use the STAX macro not only to specify and cancel attention exits, but also to defer the dispatching of attention exits. The DEFER operand of STAX macro instruction can be specified to set an indicator that will postpone the dispatching of attention exits for a TCB and all of the TCBs above it on the mother-daughter TCB chain. When STAX with the DEFER=YES option is specified, a bit in the RB that represents the issuer's routine is set (or reset). The indicator in the TCB, which allows or defers the dispatching of attention exits, is set equal to the result of ORing all of these bits in the RBs on the TCB RB chain. When the TCB defer indicator is off for the a TCB and all its subtasks, then attention exits will be dispatched. If the defer indicator is on for a TCB or any of its subtasks, then attention exits will be deferred until the defer indicator(s) for the TCB and all of its subtasks are off. When an attention exit can once again be dispatched, the DEFER=NO operand can be used to enable it to be dispatched.

The deferral bit setting of a routine (RB) can be changed or propagated to other routines (RBs) which are used by the original RB. There are three cases to be considered.

1. A new RB is created and placed on the RB queue along with the original RB. This can occur if the original RB issues a LINK. In this situation, the routine that has been linked maintains its own deferral bit setting. The deferral bit setting of the original RB is not passed to the new RB, nor is the deferral bit setting of the new RB passed back to the original RB.
2. A new RB is created and placed on the RB queue and the original RB is destroyed. This can occur if the original RB issues an XCTL macro. The routine receiving control under the new RB receives the deferral bit setting of the original RB.
3. No new RB is created but control is passed to a routine running under the original RB. This can occur if the original RB issues a CALL or LOAD macro. The called or loaded routine runs under the original RB. If the called or loaded routine issues a STAX macro instruction with the DEFER option, then the deferral bit setting is changed for the original RB.

Note: Tasks within a tree structure being stopped for the attention exit scheduling will be stopped in an indeterminate order when any are deferring attention exits. As a result, care must be taken to control intertask dependencies and dependencies on scheduling attention exits. Failure to do so may result in an intertask deadlock that can only be relieved by canceling the TSO user.

Specifying a Terminal Attention Exit -- The STAX Macro Instruction

Use the STAX macro instruction to specify the address of an attention exit routine that is to be given control asynchronously when a user strikes the attention key 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 any operands.

The STAX macro instruction is used only in a time sharing environment. When a task other than a TSO user issues the STAX macro, no action is taken. In addition, attention exits can be established only for time sharing tasks operating in the foreground.

The system routines that process attention handling require that the STAX parameter list remain unchanged for the life of the program. Because the expansion of the STAX parameter list is usually located in an area that is reusable by the active program, you should either code the necessary protection to prevent overlays or you should make a copy of the parameter list in an area that is non-reusable.

Issue the STAX macro instruction to provide the information required by the STAX service routine.

The STAX macro may be issued in 24- or 31-bit addressing mode. An attention exit routine receives control in the same addressing mode in which the STAX macro is issued.

The STAX macro instruction has a list, an execute, and a standard 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. The standard form does not require you to specify MF=L or MF=E.

Figure 18 shows the format of the list and execute forms of the STAX macro instruction; each of the operands is explained following the figure. Appendix A 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 18. The STAX Macro Instruction -- List and Execute Forms

Note: When the STAX macro is issued in 31-bit addressing mode, *exit addr* and *USADDR* may reside above 16 megabytes. All other input must reside below 16 megabytes.

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 code no other operands, the STAX service routine cancels the previous attention exit established by the task issuing this STAX macro instruction.

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 establishing a new attention exit routine for the task, if no previous attention exit has been established.

NO indicates that this attention exit be established as a new attention exit for this task, in addition 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 canceled. Any attention interruptions received while the defer option was in effect will be processed. 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, the program 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.

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 19 shows how an execute form of the STAX macro instruction may look if you load all the required parameters into registers.

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

Figure 19. Using Registers in the STAX Macro Instruction

The STAX Parameter List

When the list form of the STAX macro instruction expands, it builds a six-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 20 describes the contents of the STAX parameter list.

Number of Bytes	Field	Contents or Meaning
4	STXEXIT	Contains 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 32,767 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 32,767 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.

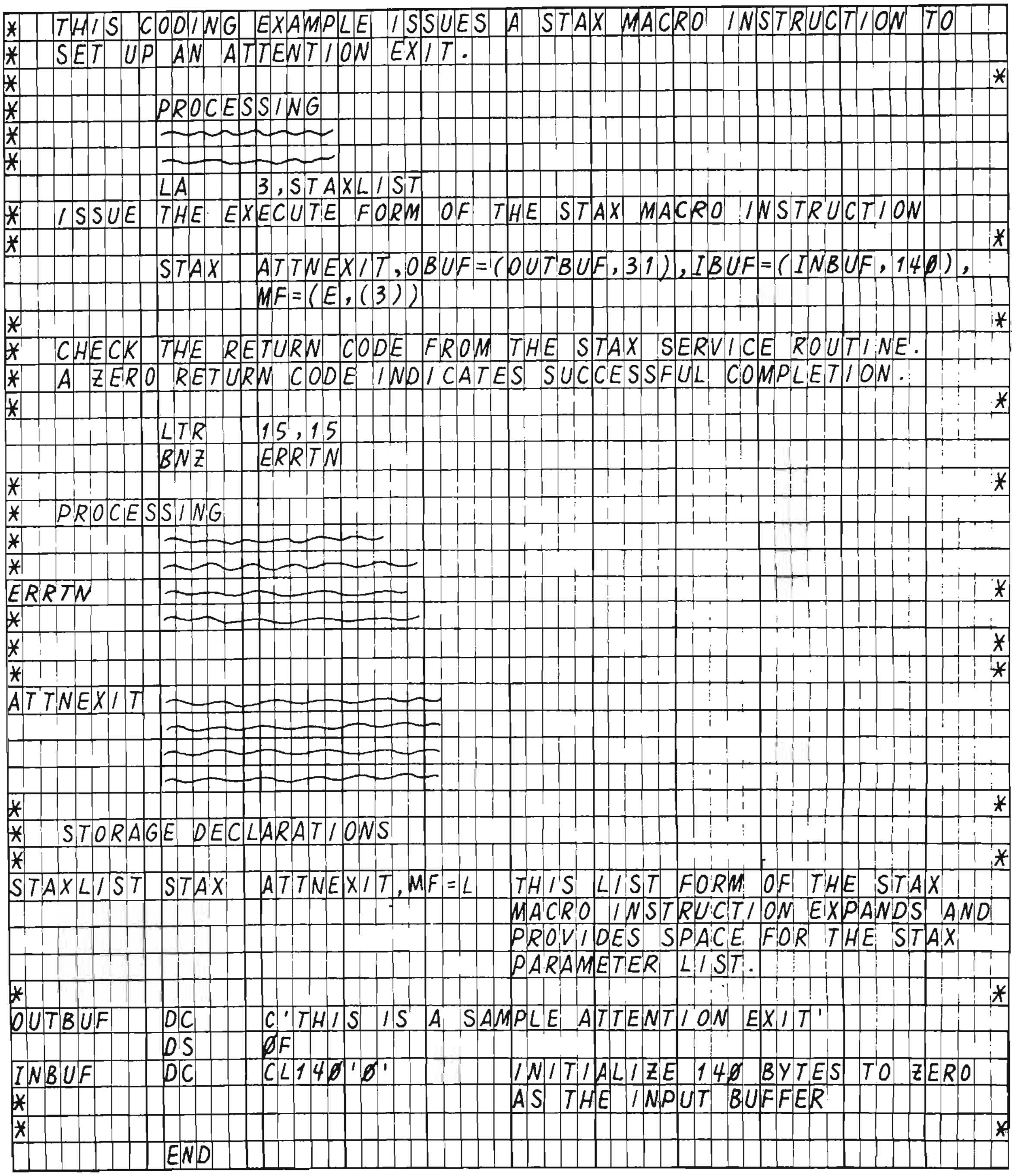
Figure 20. The STAX Parameter List (Part 1 of 2)

Number of Bytes	Field	Contents or Meaning
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.
	.... 1...	Indicates that the CLIST attention counter should be incremented.
	.... .1..	Indicates that the CLIST attention counter should be decremented.
	.... ..1.	Indicates that STXFNUM contains a format number.
	x... ...x	Reserved bits.
1	STXFNUM	STAX format flags.
	1	Indicates that the MVS/XA version of the STAX parameter list is used. If this bit is zero, the MVS/370 version of the parameter list is used.
	xxxx xxx.	Not used.
2		Reserved bytes.
4	STXNUSER	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 20. The STAX Parameter List (Part 2 of 2)

Coding Example of the STAX Macro Instruction

The coding example shown in Figure 21 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 21. Coding Example - STAX Macro Instruction

Return Codes from the STAX Service Routine

Control is returned to the instruction following the STAX macro instruction. When control is returned, register 1 will contain the address of the user parameter list provided for the previous exit for this task or will contain zero. The register will contain zero if this is the first STAX issued for this task, a STAX with a cancel option, or a STAX with only the DEFER option. If an error was detected (return code 8), then the contents of register 1 will be the same as it was at entry. Register 15 will contain one of the following return codes:

CODE	MEANING
0	The STAX service routine successfully completed the function you requested.
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 user of DEFER option (asynchronous exit routine).

If any combination of parameters or the parameters themselves are invalid, an ABEND will be issued.

The types of errors that will cause an ABEND are:

- Both DEFER=YES and DEFER=NO are specified.
- Invalid input buffer address (storage not in same key as user's TCB).
- Invalid buffer size (input or output).

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 TSO, 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 *SPL: System Macros and Facilities.*

The dynamic allocation routines may be accessed by TSO directly or 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
- 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

Considerations

- The user must correctly initialize the DAIR parameter block (DAPB) before calling DAIR. Unused fields should be zeroed or blanked (if character items).
- Specifying the data set name and the member name for DAIR entry code X'08' causes the data set to be allocated but no check is done to see if the member exists. To verify that the member really exists:
  - Allocate the data set with the member name using DAIR entry code X'08'.
  - Open the data set with DSORG=PO, MACRF=R.
  - Issue BLDL for the member. (The BLDL return code will indicate whether the member is there or not.)
  - Close the data set.
  - If BLDL indicates that the member does not exist, unallocate the data set using ddname and DAIR entry code X'18'.

Using DAIR

Invoke the DAIR service routine via a CALLTSSR macro instruction, specifying the entry point IKJDAIR in load module IKJDAIR. The DAIR service routine may be invoked in either 24- or 31-bit addressing mode. When invoked in 31-bit addressing mode, IKJDAIR may be passed input above 16 megabytes.

The control block structure required by the DAIR service routine is shown in Figure 22. 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. (See "Processing Terminal Requests -- The TSO Service Routines" for a description of the CALLTSSR macro and a list of IBM-supplied mapping macros for parameter lists.)

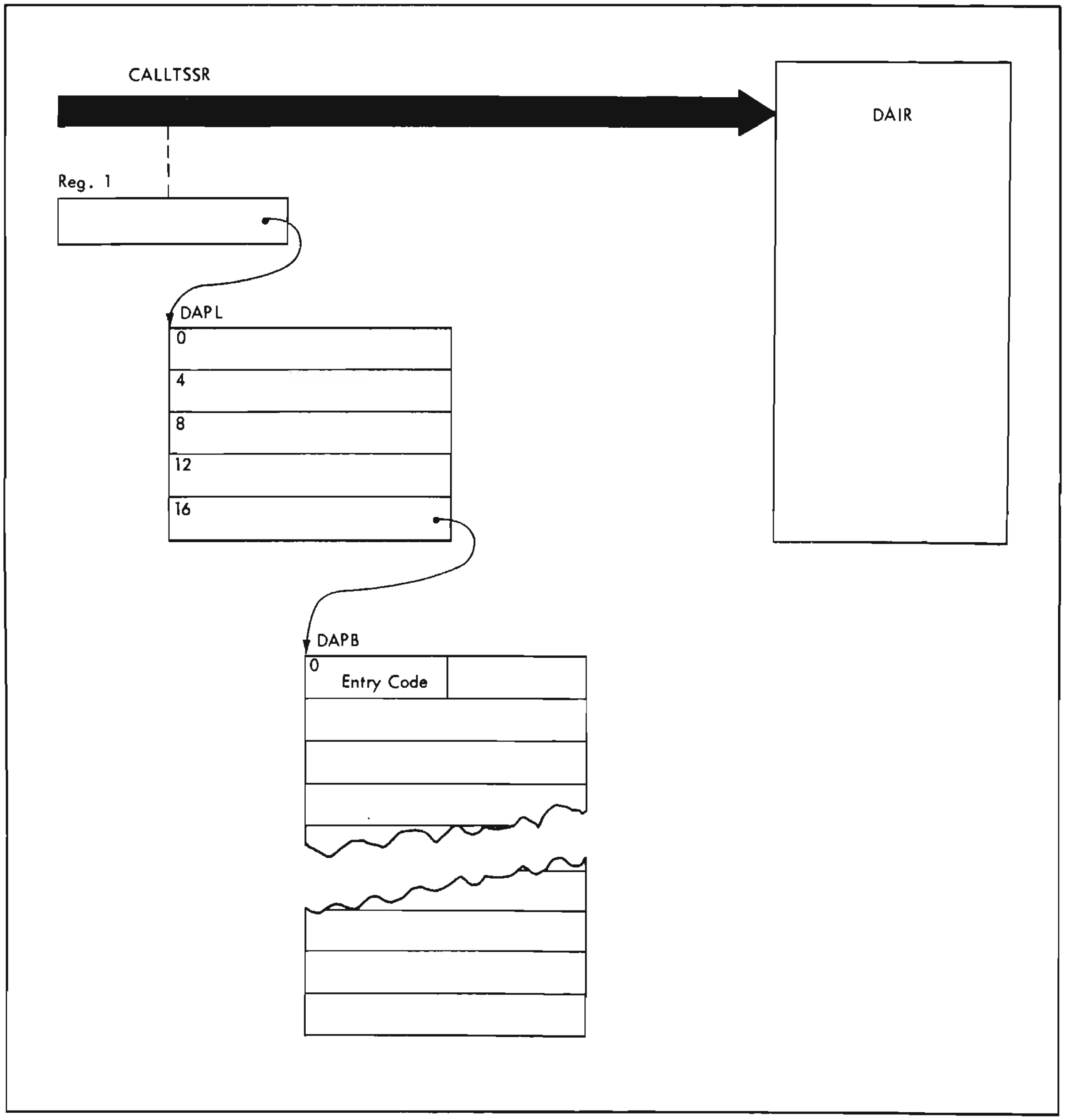


Figure 22. Control Blocks Passed to DAIR

The DAIR Parameter List (DAPL)

At entry to DAIR, register 1 must point to a DAIR parameter list that you have built. Figure 23 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. Additional information on the address and creation of the user profile table, environment control table, and protected step control block is shown in Figure 14 (the command processor 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 real 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 23. 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 24 is a list of the DAIR entry codes and the functions requested by those codes.

Entry Code	Function Performed by DAIR
X'00'	Test if a given DSNAME or DDNAME is currently allocated to the caller.
X'04'	Test if a given DSNAME is currently allocated to the caller, or is in 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'	Associate DCB parameter with a specified name for use with subsequent allocations.

Figure 24. 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.

Code X'00' - Determine if DDNAME or DSNAME Allocated

Build the DAIR parameter block shown in Figure 25 to request that DAIR determine whether or not the specified DSNAME or DDNAME is allocated.

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.
	1	The DDNAME is currently allocated to the terminal.
	Byte 2 0000 0000	Reserved. Set to zero.

Figure 25. DAIR Parameter Block -- Entry Code X'00' (Part 1 of 2)

Number of Bytes	Field	Contents or Meaning
4	DAOOPDSN	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	DAOODDN	Contains the DDNAME for the requested data set. If a DSNAME is present, the DAIR service routine ignores the contents of this field.
1	DAOOCTL	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	DAOODSO	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 organization
	.1..	Physical sequential organization
	..1.	Direct organization
	...1	BTAM or QTAM line group
	.... 1...	QTAM direct access message queue
	.... .1..	QTAM problem program message queue
	.... ..1.	Partitioned organization
	1	Unmovable

Figure 25. DAIR Parameter Block -- Entry Code X‘00’ (Part 2 of 2)

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, 52

See “Return Codes from DAIR” for return code meanings.

Code X‘04’ - Determine if DSNAME Allocated or in System Catalog

Build the DAIR parameter block shown in Figure 26 to request that DAIR determine whether or not the specified DSNAME is allocated. DAIR also searches the system catalog to find an entry for the DSNAME.

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.
	Byte 2	
	0000 0000	Reserved. Set to zero.
2		Reserved. 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.
	1...	Indexed sequential organization
	.1..	Physical sequential organization
	..1.	Direct organization
	...1	BTAM or QTAM line group
	.... 1...	QTAM direct access message queue
	.... .1..	QTAM problem program message queue
	.... ..1.	Partitioned organization
	1	Unmovable

Figure 26. 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, 52

See "Return Codes from DAIR" later in this section for return code meanings.

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

Build the DAIR parameter block shown in Figure 27 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) the data set is cataloged upon successful allocation. This is the only time a data set will be cataloged at allocation time. If the catalog attempt is unsuccessful, the data set is freed. If the proper indices are not present, the indices are built.

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, that data set is used. If the &name is not found, a new data set is allocated.

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

When setting disposition in a parameter list, only one bit should be on.

The DAIR parameter block required for entry code X'08' has the format shown in Figure 27.

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. (See "Return Codes from DAIR.")
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. If this field (DA08PDSN) is zero, the system generates a data set name unless bit 5 in DA08CTL is on, in which case a DUMMY data set is allocated. The system also generates a name if the DA08PDSN field points to a DSNAME buffer which has a length of 44, is initialized to blanks, and bit 5 in DA08CTL is off.
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 set is allocated.
8	DA08UNIT	This is an eight-byte field containing an esoteric group name, a generic group name, or a specific device address (in EBCDIC). If the unit information is less than eight characters, it must be padded to the right with blanks. If no information is to be provided, the field must be blank. In this case, DAIR will obtain information from the protected step control block. If there is no unit information in the PSCB, then a default of all direct access devices is used. The specified unit information will be ignored if volume information is obtained from the catalog, unless the unit specification is a subset of that obtained from the catalog. In this case the specified unit information will override the returned information.

Figure 27. DAIR Parameter Block -- Entry Code X'08' (Part 1 of 3)

Number of Bytes	Field	Contents or Meaning
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. In this case the following is done: if the data set is a new data set, the system determines the volume to be used for the data set based on the unit information. If the data set already exists, volume and unit information are obtained from the catalog. If the information is not found in the catalog, the allocation request is denied.
4	DA08BLK	This is a four-byte field used as follows: if the data set is a new data set and bit 0 in DA08CTL is off and bit 1 in DA08CTL is on, this field is used with DA08PQTY to determine the amount of direct access space to be allocated for the data set. If bit 6 of DA08CTL is off, the field is also used as DCB blocksize specification. The value for blocksize must be placed in the low-order two bytes, and the high-order bytes must be zero.
4	DA08PQTY	Primary space quantity desired. The high-order byte must be set to zero and the low-order three bytes should contain the space quantity required. If the quantity is omitted, the entire field must be set to zero. In the case of new direct access data sets, primary and secondary space and type of space are defaulted. Directory quantity is used if specified in DA08DQTY.
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 27. DAIR Parameter Block -- Entry Code X'08' (Part 2 of 3)

Number of Bytes	Field	Contents or Meaning
1	DA08DSP1 0000 .... 1... .... .1.. .... ..1. 1	Flag byte. Set the following bits to indicate the status of the data set: Reserved. Set these bits to zero. SHR NEW MOD OLD If this byte is zero, OLD is assumed. NEW or MOD is required if DSNAME is omitted.
1	DA08DPS2 0000 .... 1... .... .1.. .... ..1. 1	Flag byte. Set the following bits to indicate the normal disposition of the data set: Reserved bits. Set them to zero. KEEP DELETE CATLG UNCATLG If this byte is zero, it is defaulted as follows: if DA08DSP1 is NEW, DELETE is used; otherwise, KEEP is used.
1	DA08DPS3 0000 .... 1... .... .1.. .... ..1. 1	Flag byte. Set the following bits to indicate the abnormal disposition of the data set: Reserved bits. Set them to zero. KEEP DELETE CATLG UNCATLG If this byte is zero, DA08DPS2 will be used.
1	DA08CTL xx.. .... 01.. .... 10.. .... 11.. .... ..1. ....1 .... 1... .... .1.. .... ..1. 0	Flag byte. These flags indicate to the DAIR service routine what operations are to be performed: Indicate the type of units desired for the space parameters, as follows: Units are in average block length. Units are in tracks (TRKS). Units are in cylinders (CYLS). Prefix userid to DSNAME. RLSE is desired. The data set is to be permanently allocated; it is not to be freed until specifically requested. A DUMMY data set is desired. Attribute list name supplied. Reserved bit; set to zero.
3		Reserved bytes; set them to zero.
1	DA08DSO 1... .... .1.. .... ..1. ....1 .... 1... .... .1.. .... ..1. 1	A flag field. These flags are set by the DAIR service routine; they describe the organization of the data set to the calling routine. Indexed sequential organization Physical sequential organization Direct organization BTAM or QTAM line group QTAM direct access message queue QTAM problem program message queue Partitioned organization Unmovable
8	DA08ALN	Attribute list name, or a ddname from which DCB attributes should be copied (as in a JCL DCB reference). If the name is less than 8 characters, it should be padded to the right with blanks.

Figure 27. 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, 52

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 28 to request that DAIR concatenate data sets. 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.

Number of Bytes	Field	Contents or Meaning
2	DAOCCD	Entry code X'000C'
2	DAOCFLG	Reserved. Set this field to zero.
2	DAOCDARC	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	DAOCNUMB	Place in this field the number of data sets to be concatenated.
2		Reserved. Set this field to zero.
8	DAOCDDN	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 28. 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, 52

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

Code X'10' - Deconcatenate the Indicated DDNAME

Build the DAIR parameter block shown in Figure 29 to request that DAIR deconcatenate a data set. 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 29. 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, 52

See "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 30 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 30. 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 "Return Codes from DAIR" for return code meanings.

Code X'18' - Free the Specified Data Set

Build the DAIR parameter block shown in Figure 31 to request that DAIR free a data set. 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 is allocated several times to the user, all such allocations are freed.

When setting disposition in a parameter list, only one bit should be on.

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 zero and the error information is in DA18DARC.
	.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. (See "Return codes from DAIR.")
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. This field is zero if the DSNAME is not specified.

Figure 31. DAIR Parameter Block -- Entry Code X'18' (Part 1 of 2)

Number of Bytes	Field	Contents or Meaning
8	DA18DDN	Place in this field the DDNAME of the data set to be freed, or blanks. If DSNAME is specified, this field is ignored.
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. The output class may be A-Z or 0-9 in the first byte. The second byte in the field is ignored. If SYSOUT is not specified, the first byte of this field must contain zeros or blanks.
1	DA18DPS2	Flag byte. Set the following bits *to override* the normal disposition of the data set:
	0000	Reserved bits. Set them to zero.
	.... 1...	KEEP
	.... .1..	DELETE
	.... ..1.	CATLG
	1	UNCATLG If the disposition specified at allocation is to be used, this field must contain zero.
1	DA18CTL	Flag byte. These flags indicate to the DAIR service routine what operations are to be performed:
	..1.	Prefix userid to DSNAME (requires DA18PDSN data be available).
	00.. 0000	Reserved bits; set them to zero.
	...1	If this bit is on, permanently allocated data sets are unallocated. If the bit is off, the data set will be marked "not in use," if it is permanently allocated.
8		Reserved bytes; set this field to hexadecimal zeros.

Figure 31. 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, 12, 24, 28, 52

See "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 32 to request that DAIR allocate a DDNAME 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, or the DDNAME of a currently allocated data set from which DCB attributes can be copied (as in a JCL DCB reference).

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	Control byte.
	.... 1...	The data set is to be permanently allocated; it is not to be freed until specifically requested.
	.... ..1.	Attribute list name supplied.
	0000 .0.0	Reserved; set to zero.
8	DA1CDDN	Place in this field the DDNAME for the data set to be allocated to the terminal or blanks if the allocated DDNAME should be returned in this field.
8	DA1CALN	Attribute list name that has been defined previously by a X'34' entry into DAIR, or a DDNAME of a currently allocated data set from which DCB attributes can be copied. This field is used only if Bit 6 of DA1CCTL is set to one.

Figure 32. 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, 52

See "Return Codes from DAIR" later in this section for return code meanings.

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

Build the DAIR parameter block shown in Figure 33 to request that DAIR allocate a data set by DDNAME.

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 the third word of your DAIR parameter block. The DDNAME may be found associated with a DUMMY, and if so an indicator is returned but no allocation takes place.

If DAIR cannot allocate by DDNAME, it will give control to code X'08' to allocate by DSNAME and will generate a new DDNAME.

When setting disposition in a parameter list, only one bit should be on.

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. (See "Return Codes from DAIR.")
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. If the specified DDNAME is used, this field (DA24PDSN) is ignored.
8	DA24DDN	Place here the DDNAME for the data set to be allocated. This DDNAME is required. If the specified DDNAME is not allocated, then a generated DDNAME will be used with the DSNAME and the generated DDNAME will be returned in this field.

Figure 33. DAIR Parameter Block -- Entry Code X'24' (Part 1 of 4)

Number of Bytes	Field	Contents or Meaning
8	DA24UNIT	This is an eight-byte field containing an esoteric group name, a generic group name, or a specific device address (in EBCDIC). If the unit information is less than eight characters, it must be padded to the right with blanks. If no information is to be provided, the field must be blank. In this case, DAIR will obtain information from the protected step control block. If there is no unit information in the PSCB, then a default of all direct access devices is used. The specified unit information will be ignored if volume information is obtained from the catalog, unless the unit specification is a subset of that obtained from the catalog. In this case the specified unit information will override the returned information.
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. In this case, the following is done: If the data set is a new data set, the system determines the volume to be used for the data set based on the unit information. If the data set already exists, volume and unit information are obtained from the catalog. If the information is not found in the catalog, the allocation request is denied.
4	DA24BLK	This a four-byte field used as follows: If the data set is a new data set and CONTROL bit 0 is off and bit 1 is on (see below), this field is used with PRIMARY SPACE QUANTITY to determine the amount of direct access space to be allocated for the data set. If CONTROL bit 6 is off, the field is also used as a DCB blocksize specification. The value for BLOCKSIZE must be placed in the low-order two bytes. The high-order byte must be zero.
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. In this case for new direct access data sets primary and secondary space, and type of space will be defaulted. Directory quantity will be used if specified in DA24DQTY.

Figure 33. DAIR Parameter Block -- Entry Code X'24' (Part 2 of 4)

Number of Bytes	Field	Contents or Meaning
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 If this byte is zero, OLD is assumed.
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 If this byte is zero, it is defaulted as follows: if DA24DSP1 is new, DELETE is used; otherwise KEEP is used.
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 If this byte is omitted (set to zero), DA24DPS2 will be used.

Figure 33. DAIR Parameter Block -- Entry Code X'24' (Part 3 of 4)

Number of Bytes	Field	Contents or Meaning
1	DA24CTL	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 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 organization.
	.1..	Physical sequential organization.
	..1.	Direct organization.
	...1	BTAM or QTAM line group.
	.... 1...	QTAM direct access message queue.
	.... .1..	QTAM problem program message queue.
	.... ..1.	Partitioned organization.
	1	Unmovable.
8	DA24ALN	Attribute list name, or a ddname from which DCB attributes should be copied (as in a JCL DCB reference). If the name is less than eight characters, it should be padded to the right with blanks.

Figure 33. DAIR Parameter Block -- Entry Code X'24' (Part 4 of 4)

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

0, 4, 8, 12, 16, 20, 52

See "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 34 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 DAPBs, for each of the operations to be performed.

Figure 34. DAIR Parameter Block -- Entry Code X'28'

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

0, 4, 8, 12, 16, 20, 24, 28, 32, 44, 52

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 35 to request that DAIR mark data sets associated with a task control block as not in use. This allows data set 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 data sets 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'. DA2CFLG must be set to hex 0000.
8	DA2CDDN	Place in this field the DDNAME to be marked 'not in use'. DA2CFLG must be set to hex 0001.

Figure 35. DAIR Parameter Block -- Entry Code X'2C'

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

0, 4, 52

For return code meanings see "Return Codes from DAIR" later in this section.

Code X'30' - Allocate a SYSOUT Data Set to the Message Class

Build the DAIR parameter block shown in Figure 36 to request that DAIR allocate a SYSOUT data set to the message class. 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, or the DDNAME of a currently allocated data set from which DCB attributes can be copied (as in a JCL DCB reference).

To place a SYSOUT data set in a class other than the message class, use DAIR entry code X'30' and when the output has been written, specify the desired class either by using DAIR entry code X'18', or execute the FREE command, after the program has completed processing.

When setting disposition in a parameter list, only one bit should be on.

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 or zeros. The DSNAME buffer is a 46-byte field which must appear as follows: The first two bytes must contain 44 (X'2C'); the next 44 bytes contain 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 set is allocated.
8	DA30UNIT	This is an eight-byte field containing *an* esoteric group name, a generic group name, or a specific device address (in EBCDIC). If the unit information is less than eight characters, it must be padded to the right with blanks. If no information is to be provided, the field must be blank. In this case, DAIR will obtain unit information from the protected step control block. If there is no unit information in the PSCB, then a default of all direct access devices is used. The specified unit information will be ignored if volume information is obtained from the catalog, unless the unit specification is a subset of that obtained from the catalog. In this case the specified unit information will override the returned information.
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 no volume serial number is specified, the field must be blank. In this case, the following is done: If the data set is a new data set, the system determines the volume to be used for the data set based on the unit information. If the data set already exists, volume and unit information are obtained from the catalog. If the information is not found in the catalog, the allocation request is denied.

Figure 36. DAIR Parameter Block -- Entry Code X'30' (Part 1 of 2)

Number of Bytes	Field	Contents or Meaning
4	DA30BLK	Block size requested. This figure represents the average record length desired.
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 must be set to zero. In this case for new direct access data sets primary and secondary space, and type of space will be defaulted.
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. The data set will be allocated to the message class, regardless of the class you specify here. To place a SYSOUT data set in a class other than the message class, use DAIR entry code X'30' and when the output has been written, specify the desired class by using DAIR entry code X'18'.
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, or a ddname from which DCB attributes should be copied (as in a JCL DCB reference). If the name is less than eight characters, it should be padded to the right with blanks.

Figure 36. 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, 52

See "Return Codes from DAIR" later in this section for return code meanings.

Code X'34' - Associate DCB Parameters with a Specified Name

Build the DAIR parameter block shown in Figure 37 to request that DCB parameters to be used with subsequent allocations are associated with a specified name (attribute name). The following functions related to attribute names are available using code X'34':

1. Associate a set of DCB parameters to be used in subsequent allocations.
2. Search on the attribute name.
3. Delete the attribute name.

Note: When you request that DAIR associate DCB parameters with a specified name, 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. Set to zero.
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 for the attribute list name specified in field DA34NAME.
	DA34CHN	
	.1..	Build and chain an attribute list.
	DA34UNCH	
	..1.	Delete an attribute list name.
	...0 0000	Reserved bits. Set to zero.
1		Reserved. Set to zero.
8	DA34NAME	This field contains the name for the list of attributes. This field is required and if the name is less than 8 characters it must be padded to the right with blanks.
4	DA34ADDR	This field contains the address of the DAIR attribute control block (DAIRACB). This field need only be specified if bit 1 of DA34CTRL is on.

Figure 37. DAIR Parameter Block -- Entry Code X‘34’

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

0, 4, 12, 52

See “Return Codes from DAIR” below for return code meanings.

DAIRACB - DAIR Attribute Control Block

Build the DAIRACB shown in Figure 38 when you request that DAIR construct an attribute list. Place the address of the DAIRACB into the DA34ADDR field of the code X‘34’ DAIR parameter block shown in Figure 37.

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 or zeros. The DSNAME buffer is a 46-byte field which must appear as follows: The first two bytes must contain 44 (X‘2C’); the next 44 bytes contain 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 set is allocated.
8	DA30UNIT	This is an eight-byte field containing an esoteric group name, a generic group name, or a specific device address (in EBCDIC). If the unit information is less than eight characters, it must be padded to the right with blanks. If no information is to be provided, the field must be blank. In this case, DAIR will obtain unit information from the protected step control block. If there is no unit information in the PSCB, then a default of all direct access devices is used. The specified unit information will be ignored if volume information is obtained from the catalog, unless the unit specification is a subset of that obtained from the catalog. In this case the specified unit information will override the returned information.
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 no volume serial number is specified, the field must be blank. In this case, the following is done: If the data set is a new data set, the system determines the volume to be used for the data set based on the unit information. If the data set already exists, volume and unit information are obtained from the catalog. If the information is not found in the catalog, the allocation request is denied.

Figure 36. DAIR Parameter Block -- Entry Code X‘30’ (Part 1 of 2)

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. Set to zero.
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 for the attribute list name specified in field DA34NAME.
	DA34CHN	
	.1..	Build and chain an attribute list.
	DA34UNCH	
	..1.	Delete an attribute list name.
	...0 0000	Reserved bits. Set to zero.
1		Reserved. Set to zero.
8	DA34NAME	This field contains the name for the list of attributes. This field is required and if the name is less than 8 characters it must be padded to the right with blanks.
4	DA34ADDR	This field contains the address of the DAIR attribute control block (DAIRACB). This field need only be specified if bit 1 of DA34CTRL is on.

Figure 37. DAIR Parameter Block -- Entry Code X'34'

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

0, 4, 12, 52

See "Return Codes from DAIR" below for return code meanings.

DAIRACB - DAIR Attribute Control Block

Build the DAIRACB shown in Figure 38 when you request that DAIR construct an attribute list. Place the address of the DAIRACB into the DA34ADDR field of the code X'34' DAIR parameter block shown in Figure 37.

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.
	..00 0000	Reserved bits. Should be set to zero.
3		Reserved. Should be set to zero.
3	DAIEXPDT	This field contains a data set expiration date specified in binary.
	DAIYEAR	The first byte contains the expiration year.
	DAIDAY	The next 2 bytes contain the expiration day, left justified. For example, the date 99352 is specified '630160'B.
2		Reserved. Should be set to zero.
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).
	0... 00..	Reserved bits. Should be set to zero.
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 EOT.
	...0 0000	Reserved bits. Should be set to zero.
1	DAIEKYLE	This field contains the key length.
6		Reserved. Should be set to zero.

Figure 38. DAIR Attribute Control Block (DAIRACB) (Part 1 of 2)

DAIRFAIL Routine (IKJEFF18)

The DAIRFAIL routine analyzes return codes from SVC99 or DAIR, and performs one of the following functions, as requested:

- Issue an error message when appropriate.
- Issue an error message, as well as return the message to the caller.

This process of returning the message(s) to the caller is referred to as extracting the message.

DAIRFAIL must receive control in 24-bit addressing mode and be passed input that resides below 16 megabytes. If the program invoking DAIRFAIL is executing in 31-bit addressing mode, it may issue a LINK macro without switching addressing modes to invoke DAIRFAIL. (The LINK macro ensures that DAIRFAIL is invoked in 24-bit addressing mode.) When linking to IKJEFF18, provide the address of the following four-word parameter list in register 1:

Offset			
Dec	**Hex**	**Field Name**	**Contents**
DSECT - DFDSECTD			
0	0	DFS99RBP or DFDAPLP	Address of the failing SVC99 request block or address of the failing DAIR parameter list.
4	4	DFRCP	Address of a fullword containing either the SVC99 or DAIR return code.
8	8	DFJEFF02	Address of a fullword containing either the entry point address of IKJEFF02 (message writer routine) or zeros, if that address is unknown. This field (DFJEFF02) must always contain an address.
12	C	DFIDP	Address of a two-byte area containing: Byte 1 Switches Bit 1: 0 - PUTLINE issued 1 - WTP issued Bit 2: 1 - Caller wants message extracted only. Bit 3: 1 - Caller wants message extracted as well as issued as a PUTLINE or write-to-programmer (WTO). Byte 2 Caller identification number X'01' - DAIR X'32' - SVC99 X'33' - SVC99 invoked by the FREE command
16	10	DFCPPLP	Address of the CPPL. This is needed only when IKJEFF18 is called with an SVC99 error and the user is not requesting a write-to-programmer message.
20	14	DFBUFP	Address of DFBUFS buffer if bit 2 (DFBUFSW) or bit 3 (DFBUFS2) of DFIDP is on. This is required when the message is to be extracted and returned to the caller. If the DFBUFSW is on, the message(s) will only be extracted. If DFBUFS2 is on, the message(s) will be issued as well as extracted and returned to the caller. It will be possible to extract the first level and one second level message.

Offset Dec	Offset Hex	Field Name	Contents
DSECT - DFDSECT2			
		DFBUFS or	
0	0	DFBUFL1	A 2 byte field that will contain the total length of the first level message, plus 4 bytes for length and offset fields.
2	2	DFBUF01	A 2 byte field containing the offset field. It will be set to zero when a message is extracted.
4	4	DFBUFT1	A 251 byte buffer that will contain the text of the first level message extracted. If the message is greater than 251 bytes, the message will be truncated.
256	100	DFBUFL2	A 2 byte field containing the total length of the first second level message plus four bytes. If there is no second level message, this field will contain HEX zeros.
258	102	DFBUF02	A 2 byte field containing the offset. It will be set to zero when a message is extracted.
260	104	DFBUFT2	A 251 field that will contain the text of the first second level message extracted. If the message is greater than 251 bytes, the message will be truncated.

The IKJEFFDF macro may be used to map the fields in the parameter list. Specify DFDSECT=YES option to obtain DSECT DFDSECTD instead of storage. Specify the DFSECT2=YES option to obtain DSECT DFDSECT2 instead of storage. DFDSECT2 defines a storage area supplied by the caller. DAIRFAIL will return the requested informational message(s) in the associated buffers. It is not necessary to initialize these buffers. On return from DAIRFAIL, the buffers will contain the extracted message(s).

DAIRFAIL allows the user to specify that a write-to-programmer message should be issued rather than the default PUTLINE routine. This is especially useful for analyzing errors occurring in a batch invocation of SVC99. If the high-order bit of the caller identification area (pointed to by DFIDP) is on, a write-to-programmer message will be issued instead of a PUTLINE. When the write-to-programmer feature is used, the address of the CPPL (DFCPPLP) need not be specified.

The return code from DAIRFAIL is contained in register 15 as follows:

0	-	Message issued successfully
4	-	Invalid caller identification number
8	-	Message writer detected an error while attempting to issue a message
12	-	Extracted message buffer parameter list error

GNRLFAIL/VSAMFAIL Routine (IKJEFF19)

The GNRLFAIL/VSAMFAIL routine analyzes VSAM macro instruction failures, subsystem request (SSREQ) failures, parse service routine or PUTLINE failures, and ABEND codes, and issues an appropriate error message. It will insert the meaning of return codes from the VSAM/job entry subsystem interface. Other VSAM codes are explained in the *VSAM Programmer's Guide.*

GNRLFAIL/VSAMFAIL must receive control in 24-bit addressing mode and be passed input that resides below 16 megabytes. If the program invoking GNRLFAIL/VSAMFAIL is executing in 31-bit addressing mode, it may issue a LINK macro without switching addressing modes to invoke it. (The LINK macro ensures that GNRLFAIL/VSAMFAIL is invoked in

24-bit addressing mode.) When linking to IKJEFF19, provide the address of a pointer to the following parameter list in register 1:

Offset			
Dec	Hex	Field Name	Contents
0	0	GFCBPTR	Pointer to VSAM ACB if GFOPEN or GFCLOSE callerid. Pointer to VSAM RPL for other VSAM macro failures. Pointer to SSOB if GFSSREQ caller id.
4	4	GFRCODE	Error return code from register 15 or ABEND code if GFCALLID is GFABEND.
8	8	GF02PTR	Zero, or address of TSO message issuer routine (IKJEFF02) if already loaded.
12	C	GFCALLID	ID for caller's failing VSAM macro, or other failure.
		Hexadecimal	
GFCALLID =		01 (GFCHECK)	for VSAM CHECK macro error
		02 (GFCLOSE)	for VSAM CLOSE macro error
		03 (GFENDREQ)	for VSAM ENDREQ macro error
		04 (GFERASE)	for VSAM ERASE macro error
		05 (GFGET)	for VSAM GET macro error
		06 (GFOPEN)	for VSAM OPEN macro error
		07 (GFPOINT)	for VSAM POINT macro error
		08 (GFPUT)	for VSAM PUT macro error
		15 (GFPARSE)	for parse service routine error, other than a return code of 4 or 20.
		16 (GFPUTL)	for PUTLINE service routine error
		1F (GFABEND)	Issue ABEND message
		20 (GFSSREQ)	for Subsystem interface request (SSREQ) error
14	E	GFBITS	Special processing switches
		GFKEYN08	1... .... Caller not in key 0 or 8.
		GFSUBSYS	.1.. .... Caller used VS2 VSAM/job entry subsystem interface.
		GFWTPSW	..1. .... Issue error message as write-to-programmer instead of PUTLINE.
15	F	GFRESV1	Reserved.
16	10	GFCPPLP	Pointer to TMP's CPPL control block (needed if PUTLINE issued, or to have command name inserted in the failure message).
20	14	GFECBP	Pointer to ECB for PUTLINE (optional).
24	18	GFDSNLEN	Length of data set name.
26	1A	GFPGMNL	Length of program name.
28	1C	GFDSNP	Pointer to data set name to insert in VSAMFAIL error messages (optional; default is ddname).
32	20	GFPGMNP	Pointer to program name for insertion in all error messages (optional; default is ddname).
36	24	GFRESV2	Reserved.
40	28	GFRESV3	Reserved.

The return code from GNRLFAIL is contained in register 15 as follows:

0 - Message issued successfully

80 - Invalid input parameter list (GFPARMS) for IKJEFF19. A message is also issued.

Other- PUTLINE/PUTGET/IKJEFF02 message issuer error return code.

The IKJEFFGF macro may be used to map the input parameter list. Specify GFDSECT=YES option to obtain DSECT GFDSECTD instead of storage.

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 TSO. 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 relevant 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.* The BSAM/QSAM macro instructions must be issued in 24-bit addressing mode. Figure 39 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 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
REOV	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 virtual 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 storage area provided by the control program.
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 39. BSAM/QSAM Macro Functions under TSO

SAM Terminal Routines

The GET, PUT, PUTX, READ, WRITE, and CHECK macro instructions perform differently in terminal I/O than 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 completes.

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 (that is, specially-indicated characters are deleted from the message). Lowercase characters are folded to uppercase characters.

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 storage. This transfer of data is done via a TGET macro instruction which does not return control before the transfer is completed. The data is folded to uppercase.

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 until 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 four 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.

Control characters (ASCII or machine) are not supported under TSO. On output, they are removed before the data is sent to the terminal. On input, they are ignored.

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 displayed 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

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 *JCL*.

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 IBM-supplied terminal monitor program with one of your 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 or data set 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 display or inhibit the displaying of message identifiers.
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.

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 service routines.

Note: See the section "Interfacing with TSO Service Routines" for information on the CALLTSSR macro interface to TSO service routines and a list of the DSECTS provided for TSO control blocks.

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). The IOPL and the fields in the IOPL must reside below 16 megabytes. 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 40 describes the contents of the IOPL.

Number of Bytes	Field	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 40. 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 41, an extension of Figure 38, summarizes the control block interfaces established between the terminal monitor program and an I/O service routine.

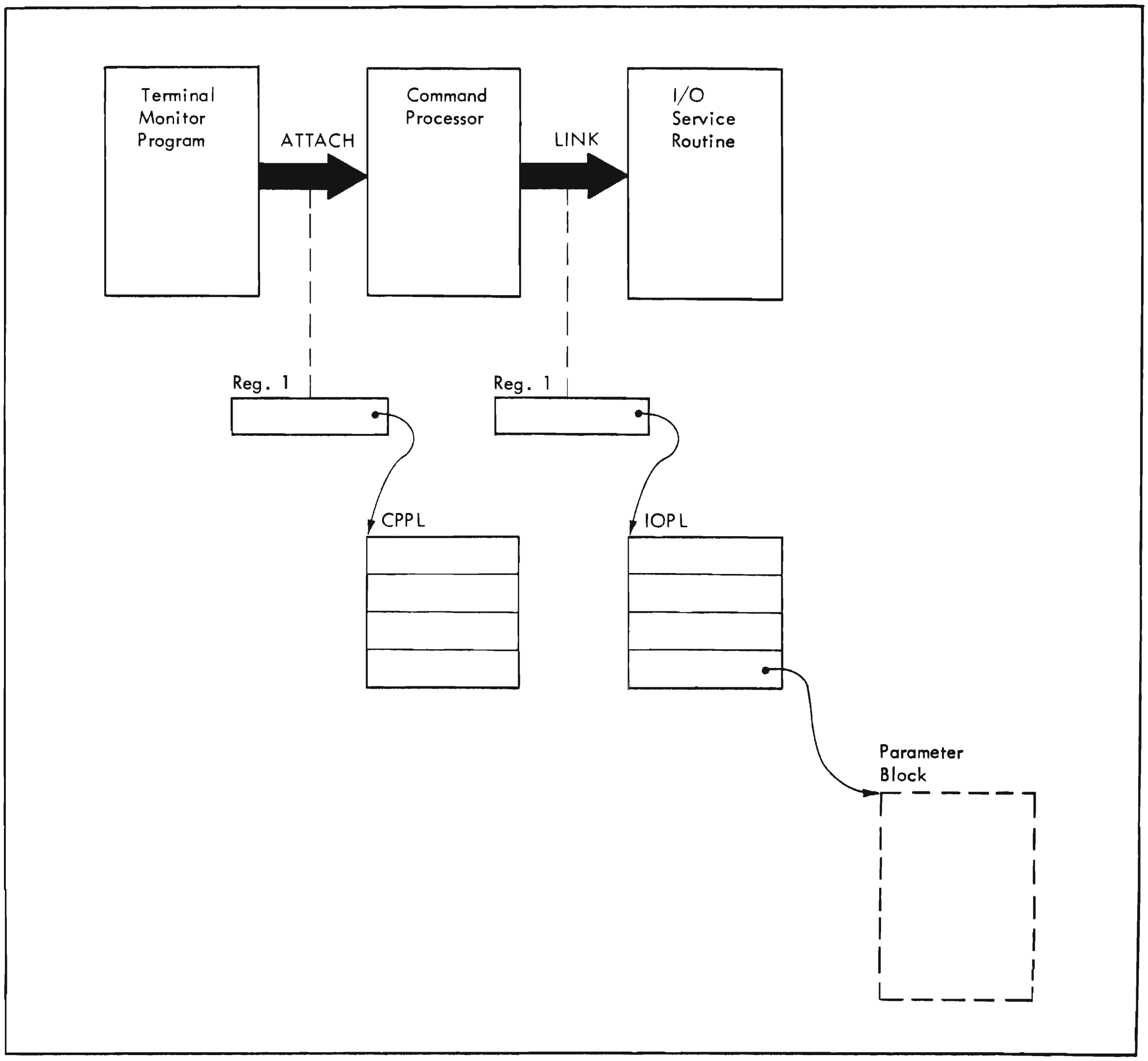


Figure 41. Control Block Interface between TMP and I/O Service Routine

Passing Control to the I/O Service Routines

Pass control to an I/O service routing using the corresponding I/O macro instruction:

Service Routine	Macro Instruction
• STACK	STACK
• GETLINE	GETLINE
• PUTLINE	PUTLINE
• PUTGET	PUTGET

You can use the DELETE macro instruction to release the storage area occupied by the load module when you have finished with your terminal I/O.

All of the TSO terminal I/O service routines are contained in the IKJPTGT load module. The IKJPTGT load module has the AMODE=24 and RMODE=24 attributes and is loaded below 16 megabytes. The TSO I/O service routines must receive control in 24-bit addressing mode.

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 "TSO Service Routines -- Their Uses and Interfaces," 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 STACK, GETLINE, PUTLINE, and PUTGET macros must be issued in 24-bit addressing mode. All input must reside below 16 megabytes.

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 referenced by IKJSTPB
- The GETLINE parameter block referenced by IKJGTPB
- The PUTLINE parameter block referenced by IKJPTPB
- The PUTGET parameter block referenced by 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 IBM-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 42 shows the list form of the STACK macro instruction; each of the operands is explained following the figure. Appendix A describes the notation used to define macro instructions.

```
[symbol]   STACK   ┌          ⎧TOP ⎫                                        ┐
                   │   DELETE=⎨PROC⎬                                        │
                   │          ⎩ALL ⎭                                        │
                   │                                                        │
                   │                              ⎧PROCN,PROMPT⎫            │
                   │   STORAGE=(element address,⎨PROCL,PROMPT⎬)            │ ,MF=L
                   │                              ⎩SOURCE      ⎭            │
                   │                                                        │
                   │           ⎧ *                        ⎫                 │
                   │   DATASET=⎪INDD=addr1,PROMPT,LIST     ⎪                 │
                   │           ⎨ MEMBER=addr3             ⎬                 │
                   │           ⎪ OUTDD=addr2,CNTL,SEQ     ⎪                 │
                   └           ⎩ CLOSE                    ⎭                 ┘
```

Figure 42. The List Form of the STACK Macro Instruction

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).

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.

PROMPT
: Specifies prompting by commands within a command procedure. PROMPT is used with the keywords PROCN and PROCL, which specify that the element to be added to the input stack is a command procedure.

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.

SOURCE

The element to be added to the input stack is an in-storage source data set.

MF=L

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

DATASET

Supports the use of ACCOUNT in the background by expanding the facilities of dataset I/O for TSO commands to include reading from a SYSIN data set and writing to a SYSOUT dataset. To use the dataset function, the input and output files passed to the STACK service routine must be preallocated, either by a previously issued ALLOCATE command, a command processor going to dynamic allocation, a DD statement specified in the logon procedure, or, in the background, a user-supplied DD statement.

*****

Specifies that STACK use the bottom element in the input stack for I/O operations. This operand is the functional equivalent of TERM=*, which is still supported for compatibility.

INDD=addr1

Specifies the input file name.

PROMPT

Allows prompting if prompting is also allowed on the bottom element of the input stack.

LIST

Lists the input from the input stream.

MEMBER=addr3

Specifies an 8-character member name for a partitioned data set which was specified as the input file with the INDD operand.

OUTDD=addr2

Specifies the output file name.

CNTL

The output line has its own control character.

CLOSE

Closes the data control blocks (DCBs) of the input stack.

SEQ

Tells dataset I/O not to remove sequence numbers.

Note: In the list form of the macro instruction, only

```
      STACK MF=L
```

is required. When only STACK MF=L is specified, the STPB is zeroed. The other operands and their sublists are optional because they may be supplied by the execute form of the macro instruction.

The operands you specify in the list form of the STACK macro instruction set up control information used by the STACK service routine. The DATASET, 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. To set up the input output parameter list (IOPL).
2. To initialize 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. To pass control to the STACK service routine which modifies the input stack.

Figure 43 shows the execute form of the STACK macro instruction; each of the operands is explained following the figure. Appendix A describes the notation used to define macro instructions.

```
[symbol]   STACK   [PARM=parm addr.][,UPT=upt addr.]

                   [,ECT=ect addr.][,ECB=ecb addr.]

                   ┌           (TOP )                                 ┐
                   │  DELETE=  {PROC}                                 │
                   │           (ALL )                                 │
                   │                                                  │
                   │                            (PROCN,PROMPT)        │
                   │  STORAGE=(element addr.,   {PROCL,PROMPT})       │
                   │                            (SOURCE      )        │
                   │                                                  │
                   │            (*                          )         │
                   │            (INDD=addr1,PROMPT,LIST     )         │
                   │  DATASET=  {MEMBER=addr3               }         │
                   │            (OUTDD=addr2,CNTL,SEQ       )         │
                   └            (CLOSE                      )         ┘

                   [,ENTRY={entry addr.}] ,MF=(E,{list addr.})
                           {   (15)    }         {   (1)    }
```

Figure 43. The Execute form of the STACK Macro Instruction

Note: TERM=* will be allowed by STACK to provide compatibility with existing modules when they are recompiled.

SEQ

Tells dataset I/O not to remove sequence numbers.

CLOSE

Closes the data control blocks (DCBs) of the bottom element of the input stack.

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 four-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.

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 DATASET, 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 TSO 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.

An in-storage list and its processing are specified by setting the STORAGE operand type to PROCN, PROCL, or SOURCE.

- PROCN or PROCL - Indicates that the in-storage list is a command procedure, a list of commands to be executed in the order specified. If you specify PROCN, 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 PROCL option is specified, the command is displayed at the terminal as it is read from the list.
- 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 PROCN or PROCL. 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 five 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.

For example, if you coded the list form of the STACK macro instruction as follows:

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

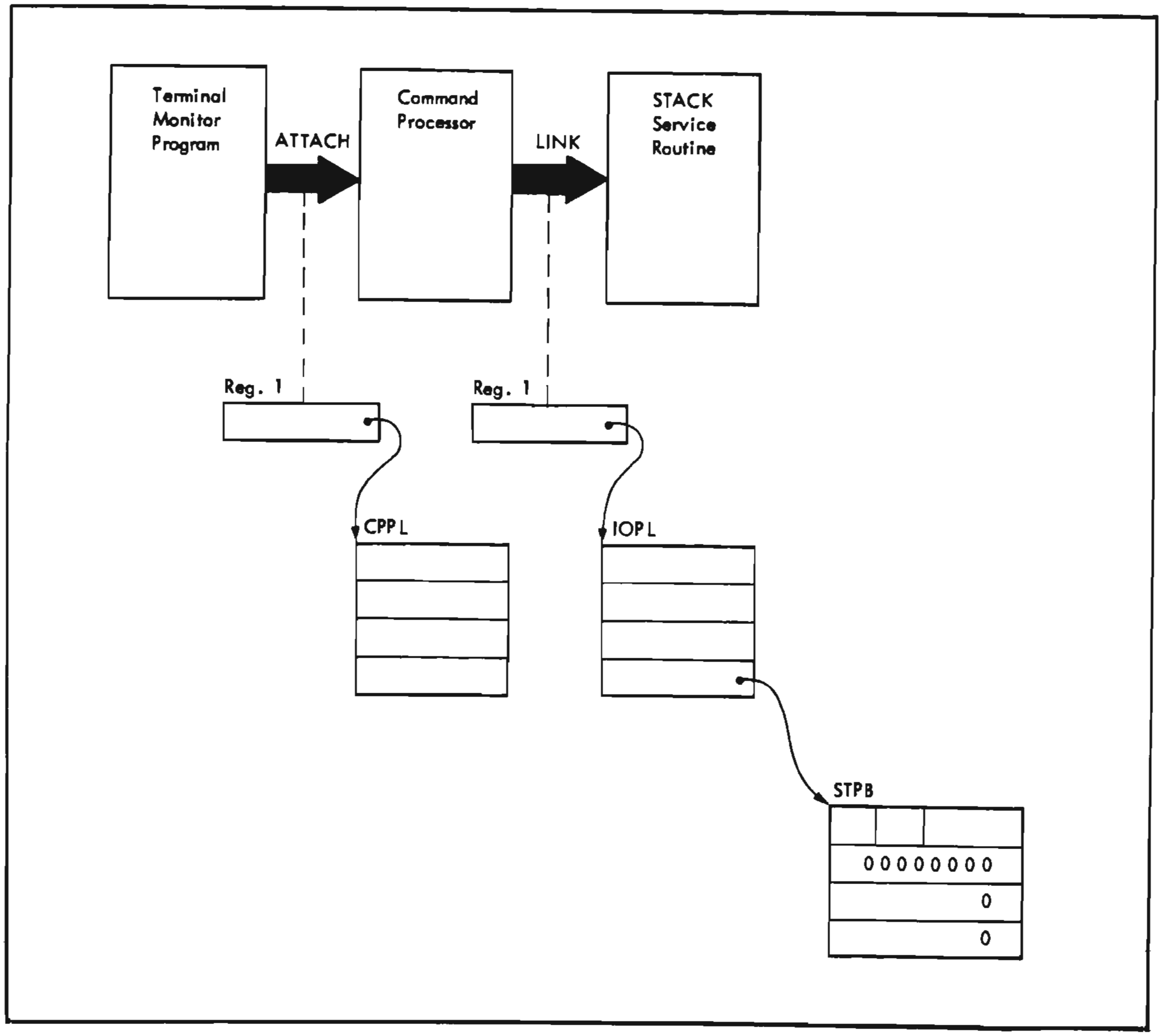
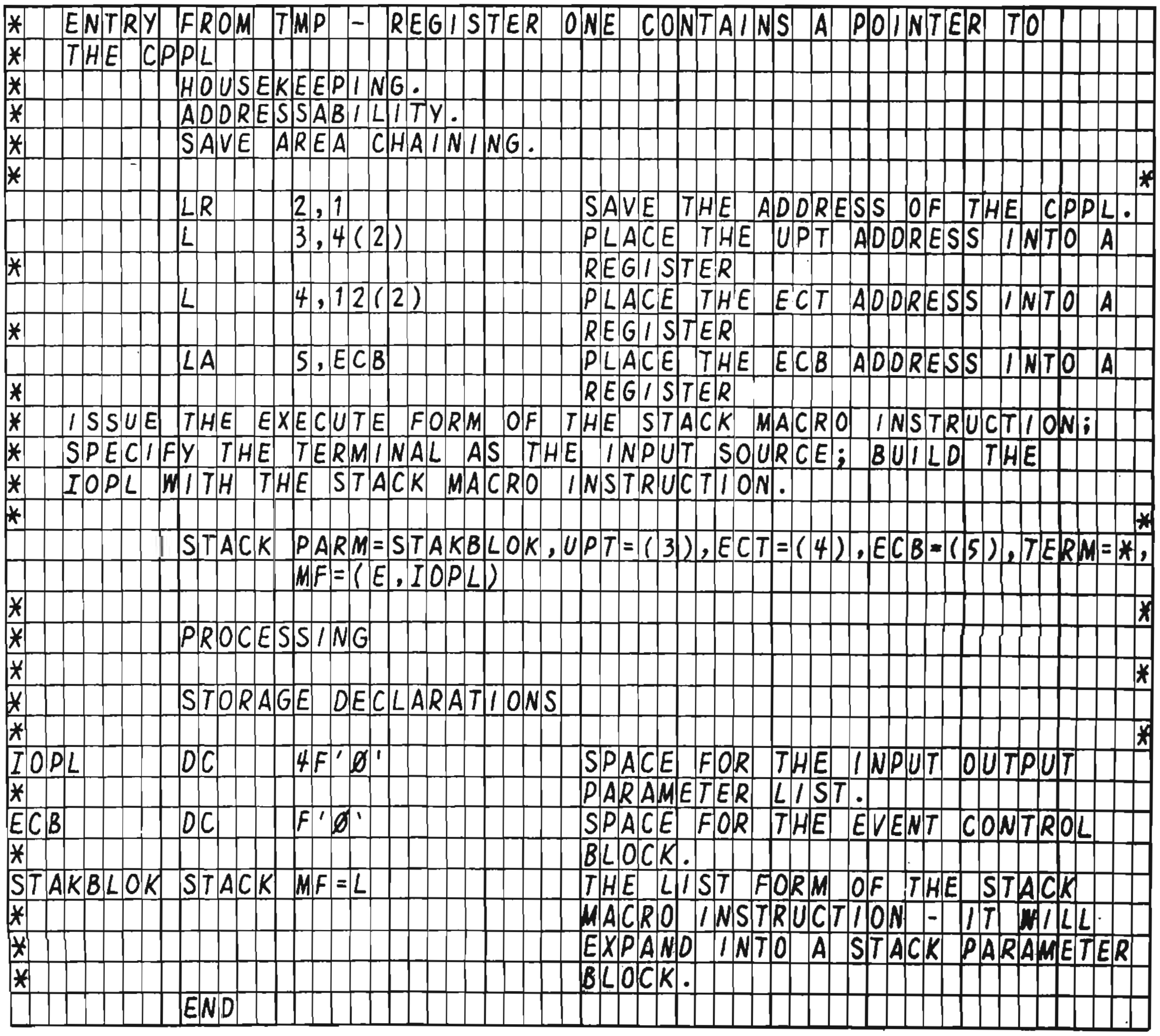


Figure 45. STACK Control Blocks: No In-Storage List

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 46 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 46. 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 doubleword 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 47 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 47. 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 48.

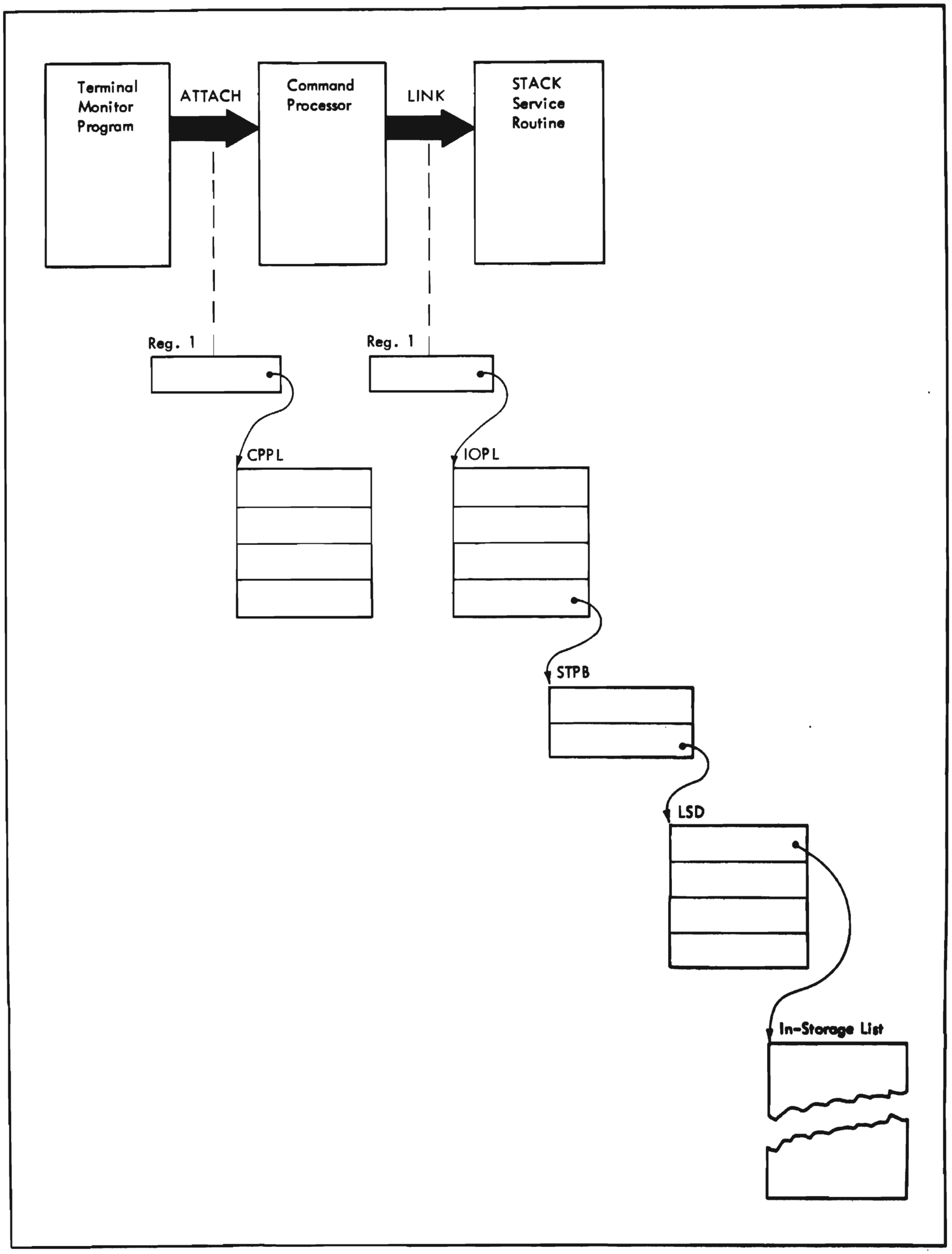


Figure 48. STACK Control Blocks: In-Storage List Specified

Figure 49 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.

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 carrier 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.

Note: 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.

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 51 shows the execute form of the GETLINE macro instruction; each of the operands is explained following the figure. Appendix A 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 51. 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. 1his 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 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.

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. A logical line is a line that has had additional processing by the GETLINE service routine before it is returned to the requesting program.

PHYSICAL

The input line to be obtained is a physical line. A physical line is a line that is returned to the requesting program exactly as it is received from the input source.

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. All station control characters are suppressed from data.

ASIS

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

a. Station control characters remain in the data.

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 carrier 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 four-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, ECB, 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.

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 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.

1. ***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 '-' or a plus '+'). 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.

2. ***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 52 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 trailing blanks to the next doubleword 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 52. 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 doubleword 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 CP relinquishes control. You may 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 fullword header followed by the text returned by GETLINE. Figure 53 shows the format of the input buffer returned by the GETLINE service routine.

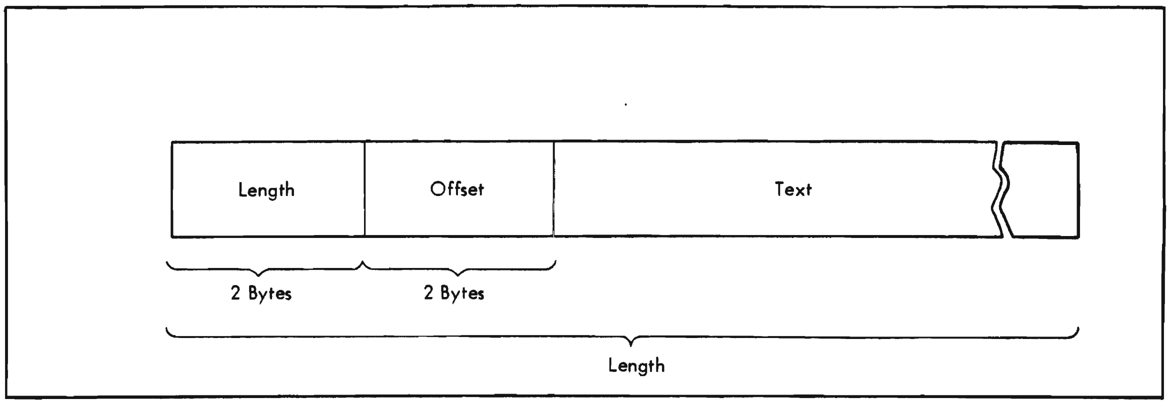


Figure 53. Format of the GETLINE Input Buffer

The two-byte length field contains the length of the input line including the header length (4 bytes). You can use the 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 54 shows the GETLINE control block structure after the GETLINE service routine has returned an input line.

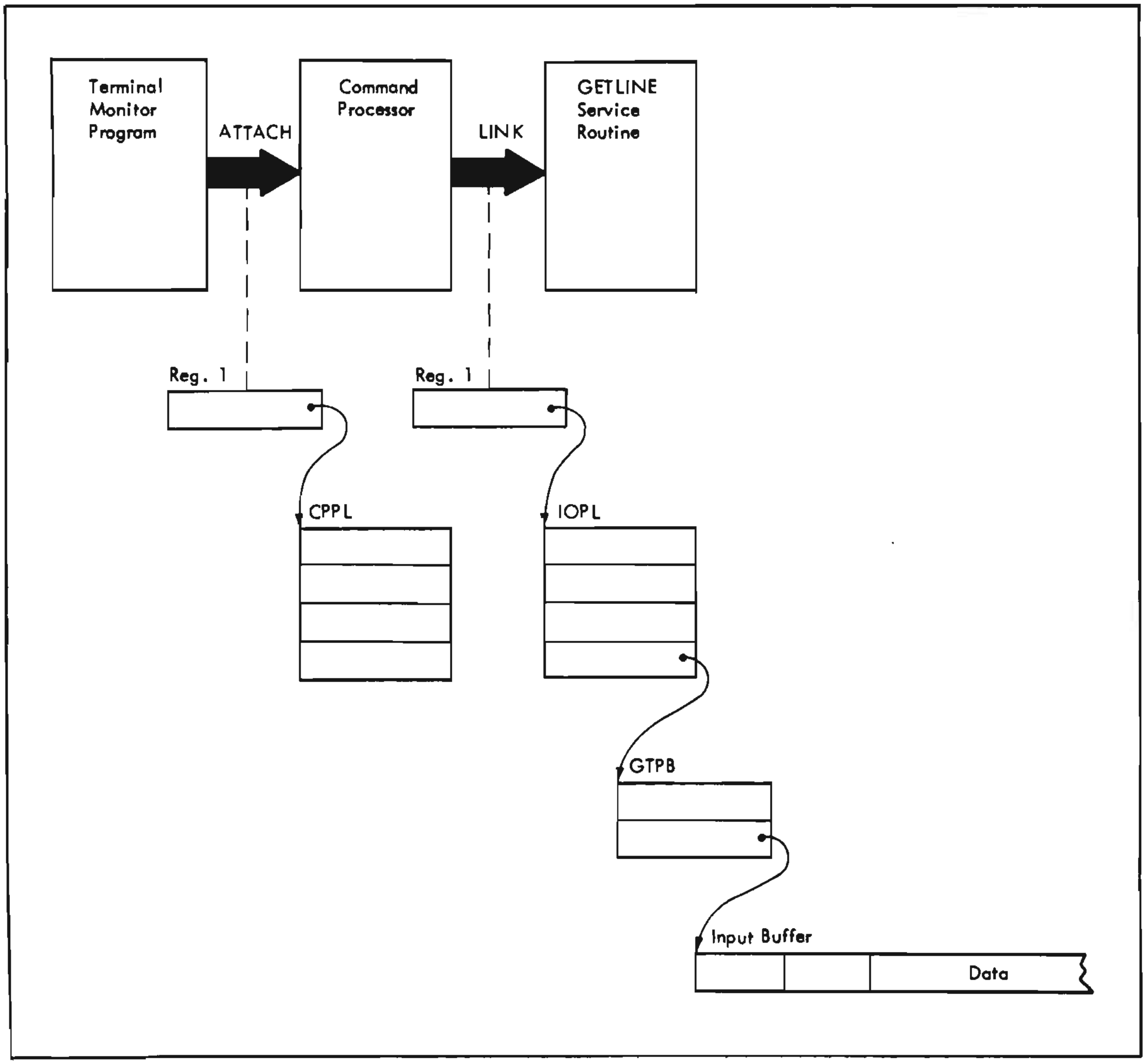


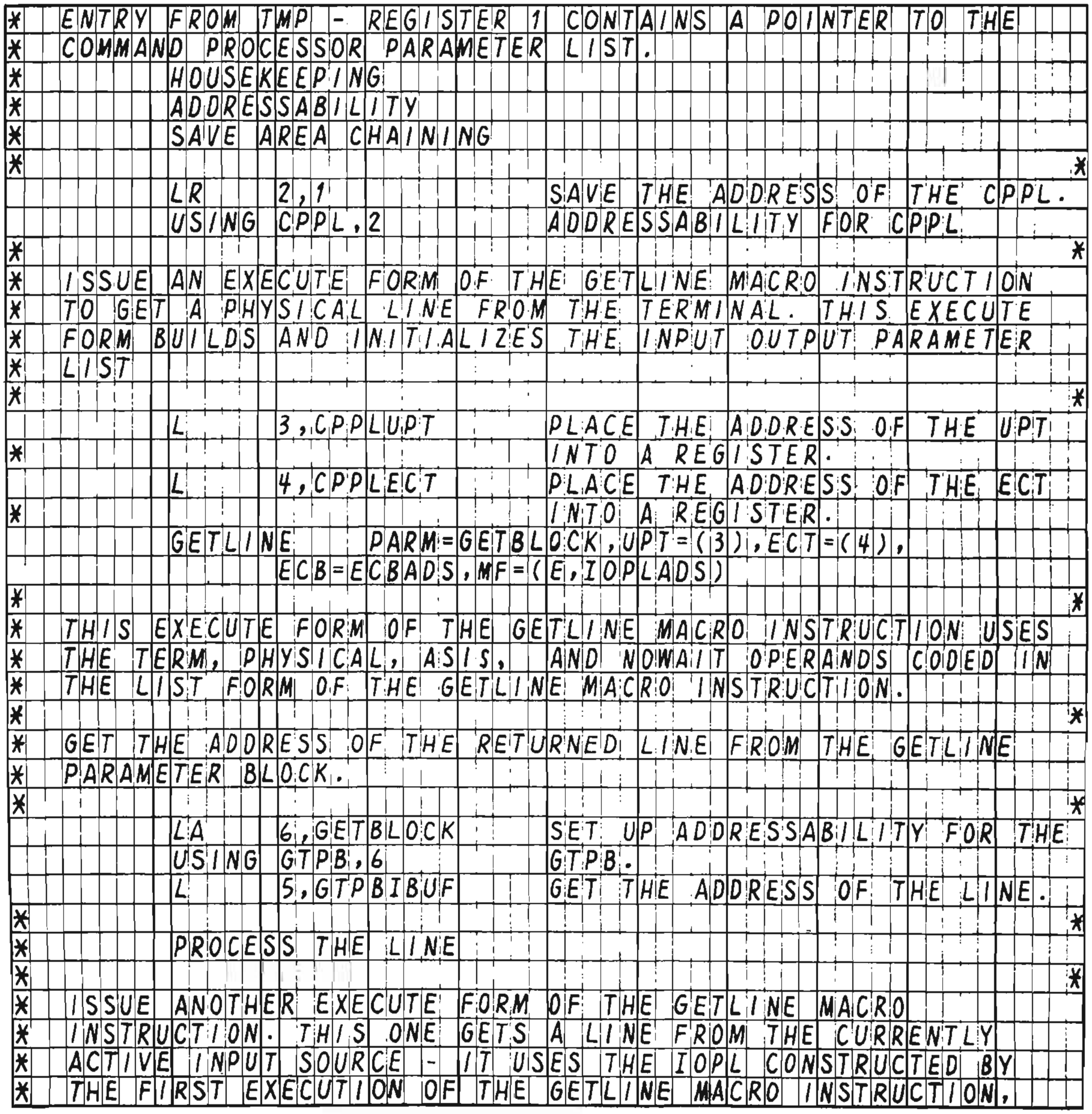
Figure 54. GETLINE Control Blocks - Input Line Returned

Examples of GETLINE

Figure 55 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.

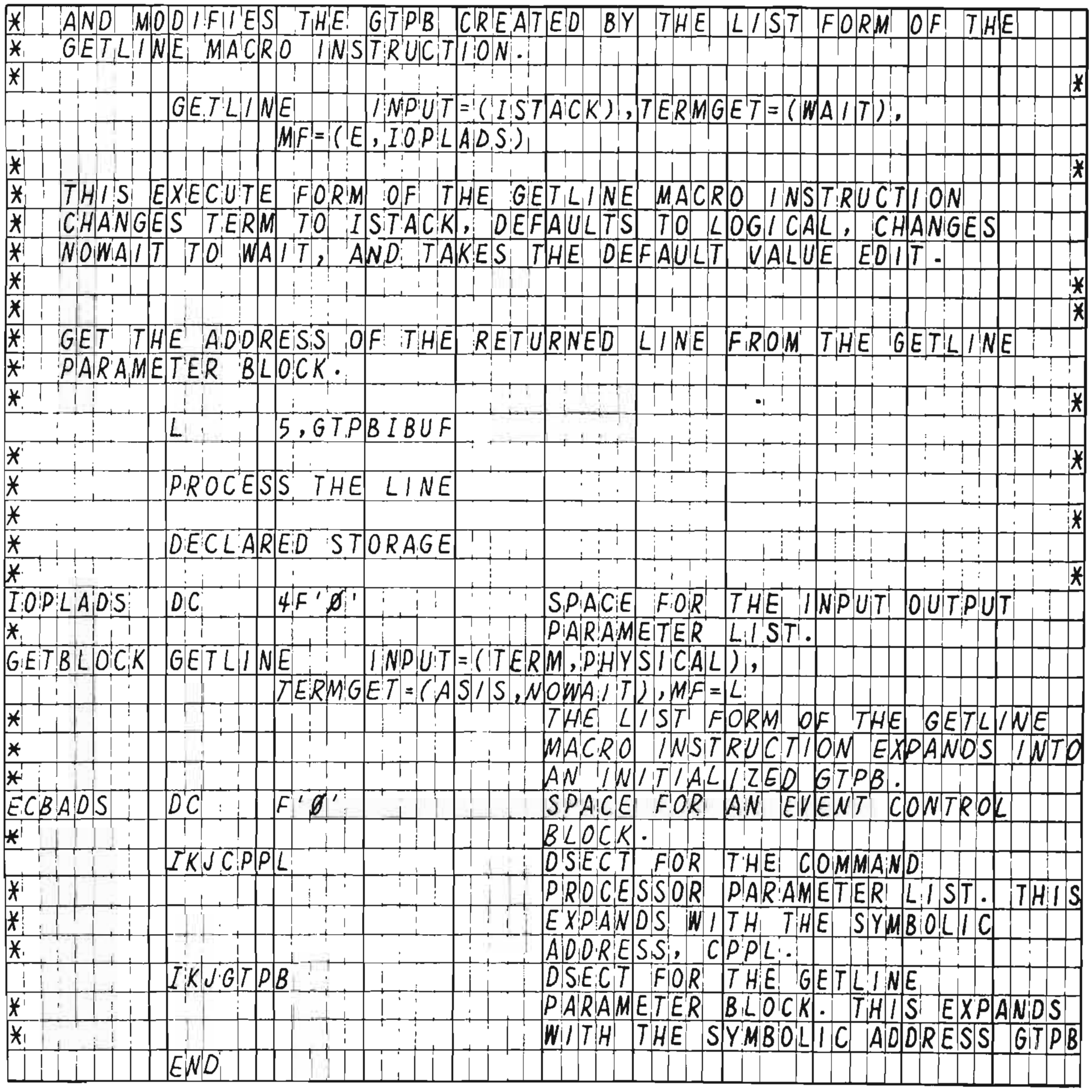
In the second execution of the GETLINE macro instruction, the same IOPL is used, but the GETLINE options are changed from TERM to ISTACK, and from NOWAIT to WAIT explicitly, and from PHYSICAL to LOGICAL and from ASIS to EDIT by default.

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 55. 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 55. 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 15:

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.
28	The terminal has been disconnected.
36	End of data was received when a continuation condition was expected.

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 appropriate.

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 56 shows the list form of the PUTLINE macro instruction; each of the operands is explained following the figure. Appendix A describes the notation used to define macro instructions.

```
[symbol]   PUTLINE    ┌                           ┌,SINGLE ┐          ┐
                      │OUTPUT=(output address{,TERM  }{,MULTLVL}{,INFOR})│
                      └                      {,FORMAT}{,MULTLIN}{,DATA }  ┘

                      ┌            ┌EDIT   ┐                               ┐
                      │,TERMPUT=( {ASIS   }{,WAIT  }{,NOHOLD}{,NOBREAK}) │
                      └            └CONTROL┘{,NOWAIT}{,HOLD  }{,BREAKIN}   ┘

                      ,MF=L
```

Figure 56. 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 fullword 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 fullword 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 cursor 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 item 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 cursor 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 cursor 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 by item c. under EDIT.

   If the NL is embedded in your message, it is sent to the terminal as a carrier return. No idle characters are added (see item 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

```
     PUTLINE MF=L
```

is required. 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.

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.

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 57 shows the execute form of the PUTLINE macro instruction; each of the operands is explained following the figure. Appendix A 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 }

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

                     [,ENTRY={entry address}],MF=(E,{list address})
                     [       {  (15)       }]       {(1)         }
```

Figure 57. The Execute Form of the PUTLINE Macro Instruction

PARM=parameter address
Specifies the address of the 3-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 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.

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 fullword 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 fullword 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 messages 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 cursor 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 item 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 cursor 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 cursor 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 item f. below). This may cause overprinting, particularly on terminals that require a line-feed character to position the cursor 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 display or move the cursor 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 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 displayed 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 the output line is sent. Any data that was received before the interruption is kept and dis*played* 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 four-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.

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 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 58 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 multiline.
	.... .1..	The output is multilevel.
	.... ..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.
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 58. The PUTLINE Parameter Block

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).

1. 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 59.

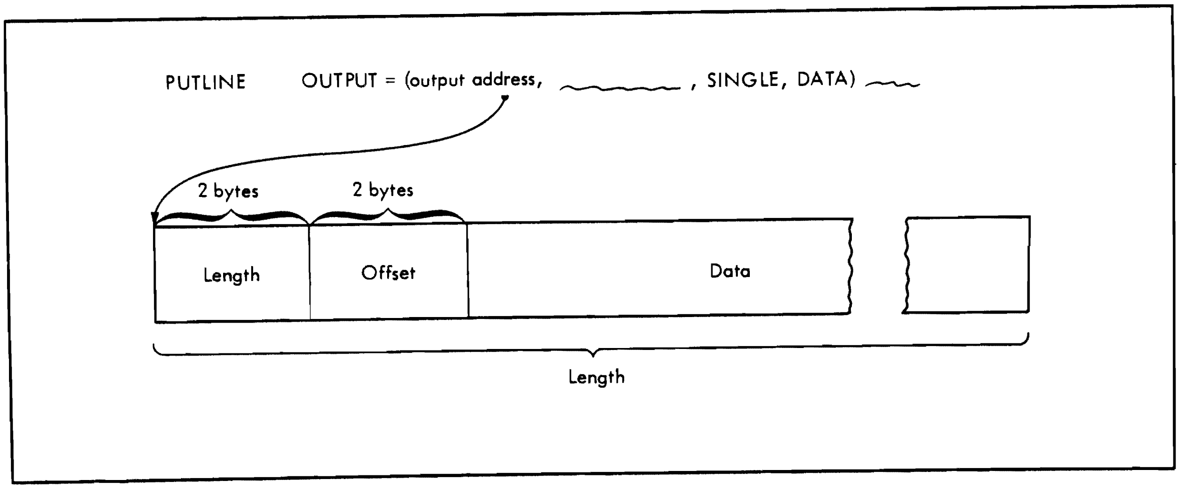


Figure 59. PUTLINE Single Line Data Format

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 60 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.

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	14	TEXT
13	17	CONTINUE

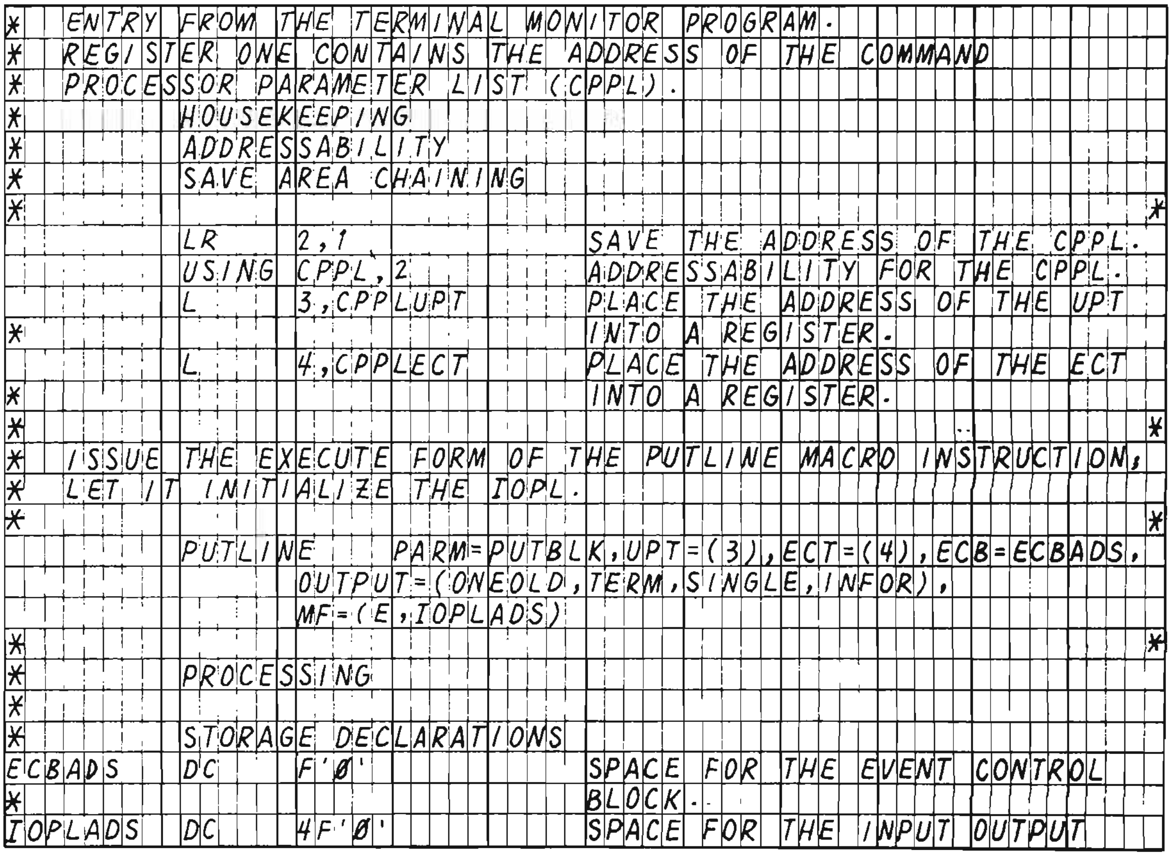
PUTLINE would place the first insert, TEXT, after the 13th character, and the second insert, CONTINUE, after the 17th character of the text field of the primary segment. After PUTLINE inserts the two text segments, the message would read:

```
    PLEASE ENTER TEXT TO CONTINUE PROCESSING
```

The leading and trailing blanks are automatically stripped off before the message is written to the terminal.

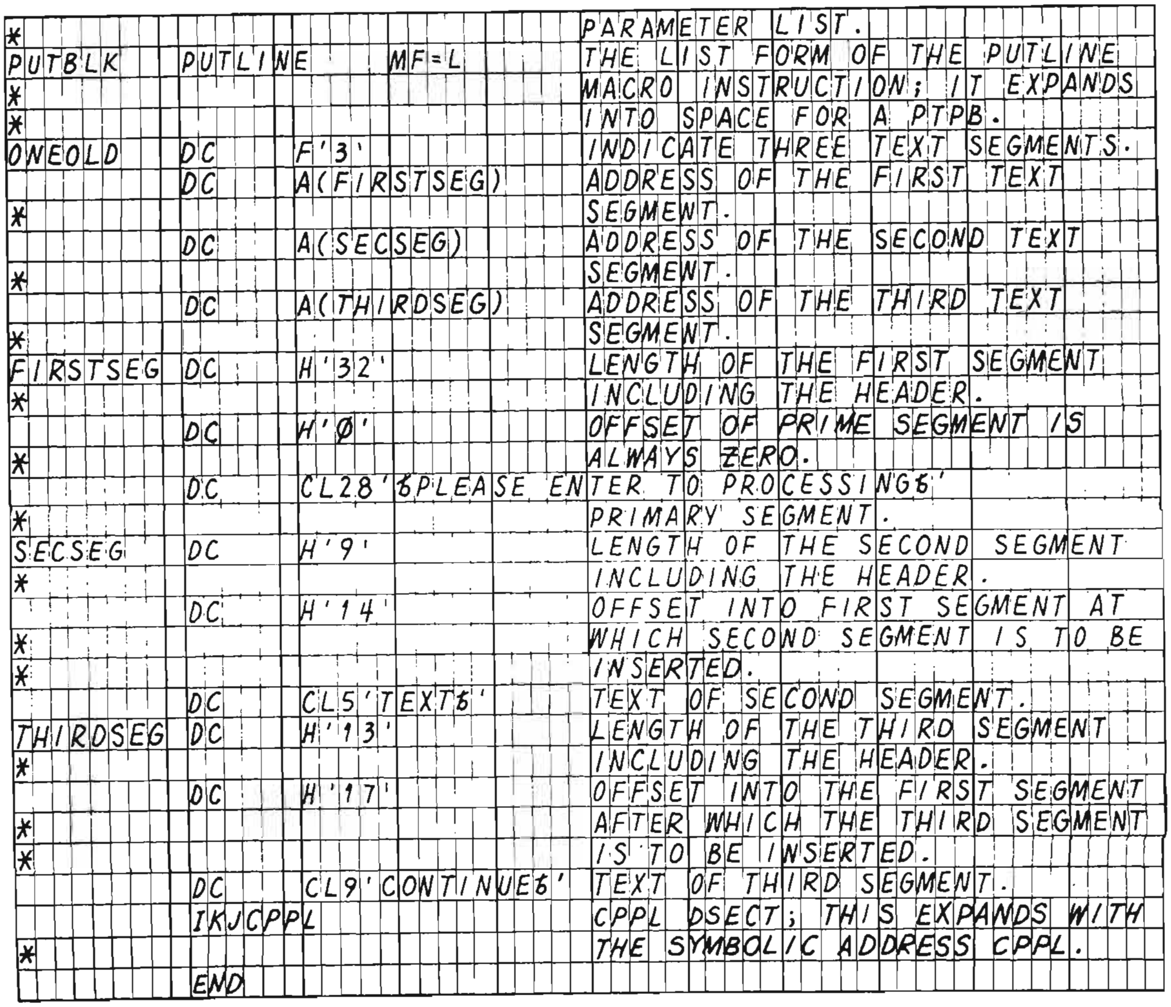
Figure 66 is an example of the code required to make use of the text insertion feature of the PUTLINE service routine; it uses the text segments shown above.

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'Ø'           SPACE FOR THE EVENT CONTROL
*                             BLOCK.
IOPLADS  DC    4F'Ø'          SPACE FOR THE INPUT OUTPUT
```

Figure 66. 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 66. 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 real storage occupied by the message with the R form of the FREEMAIN macro instruction.

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).

*1. **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 59.

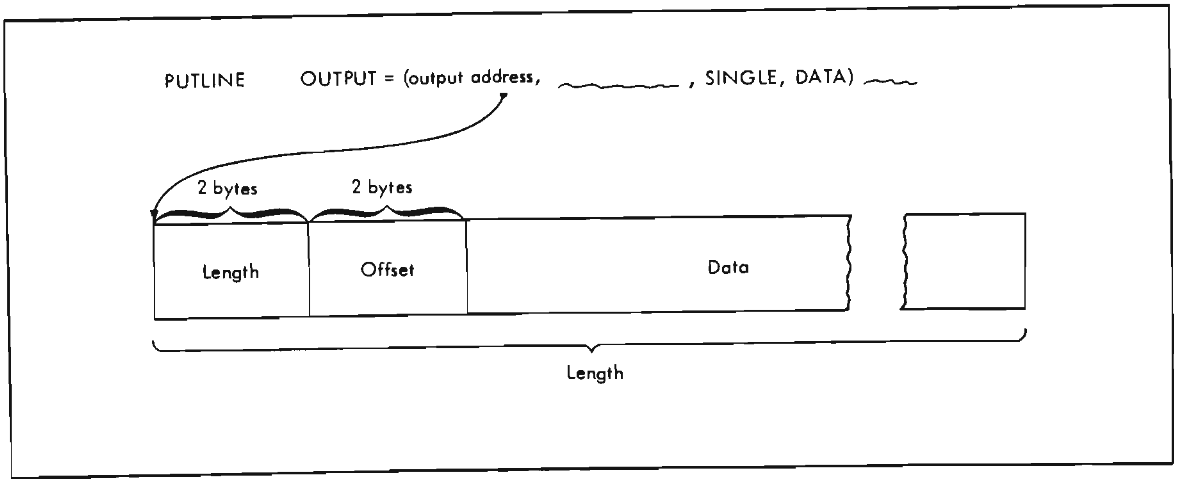
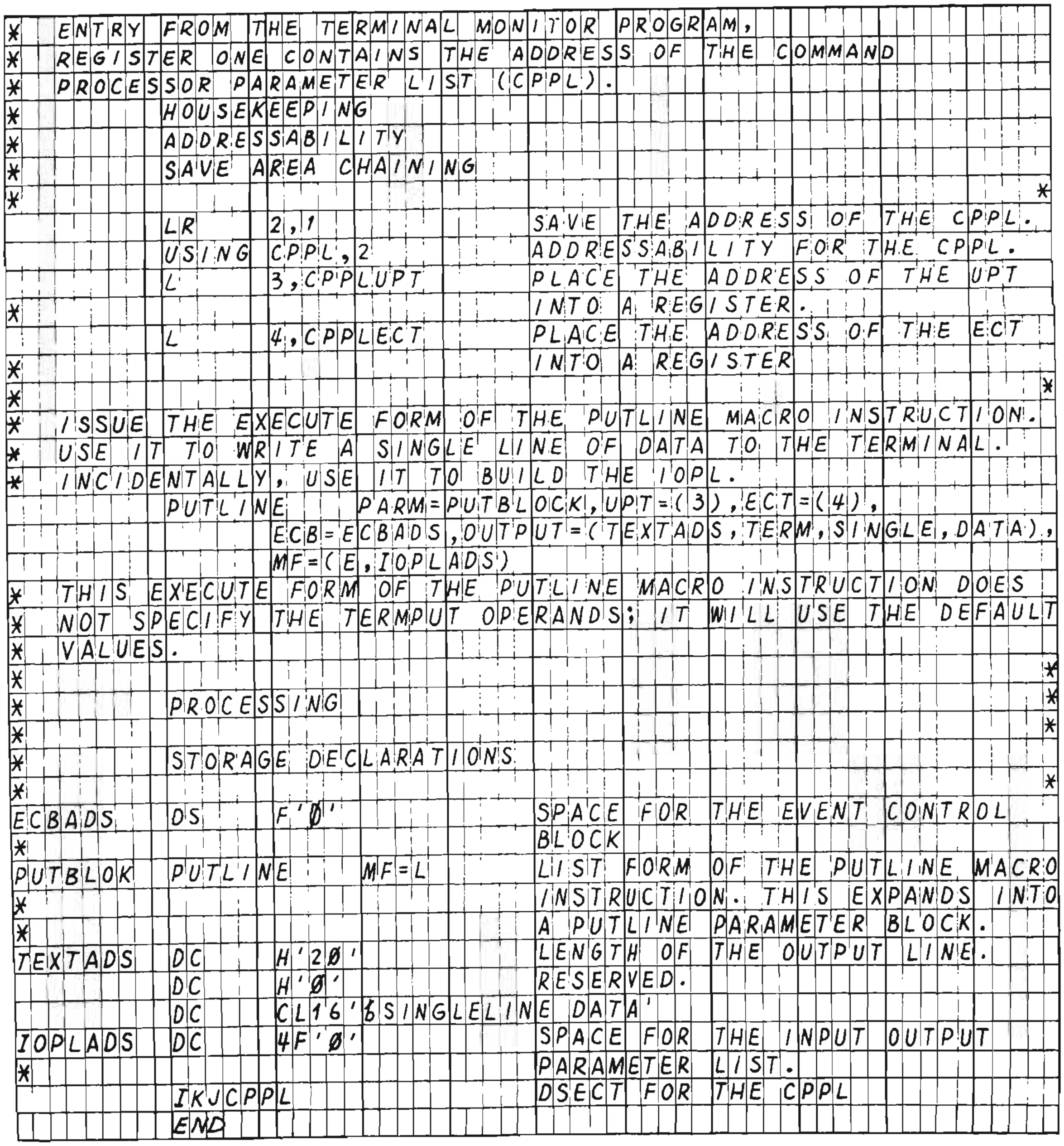


Figure 59. PUTLINE Single Line Data Format

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 60 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 60. 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 61 shows the format of PUTLINE multiline data.

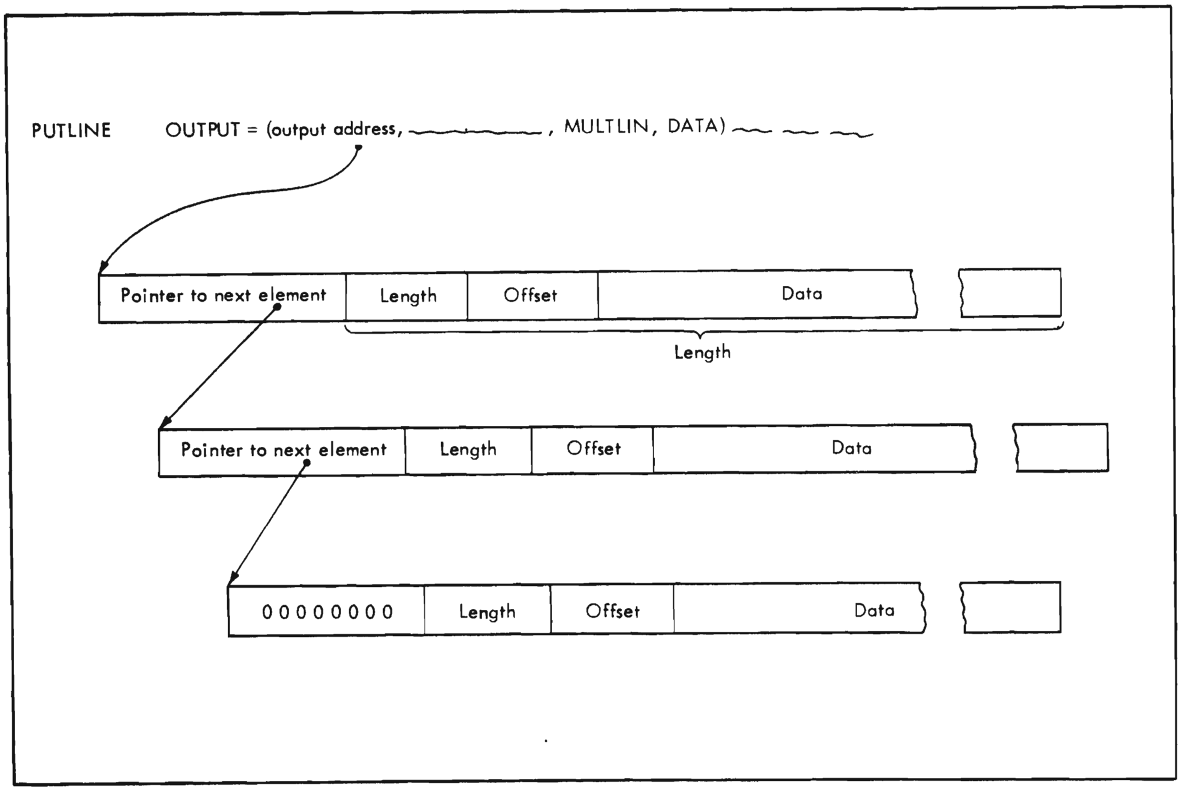
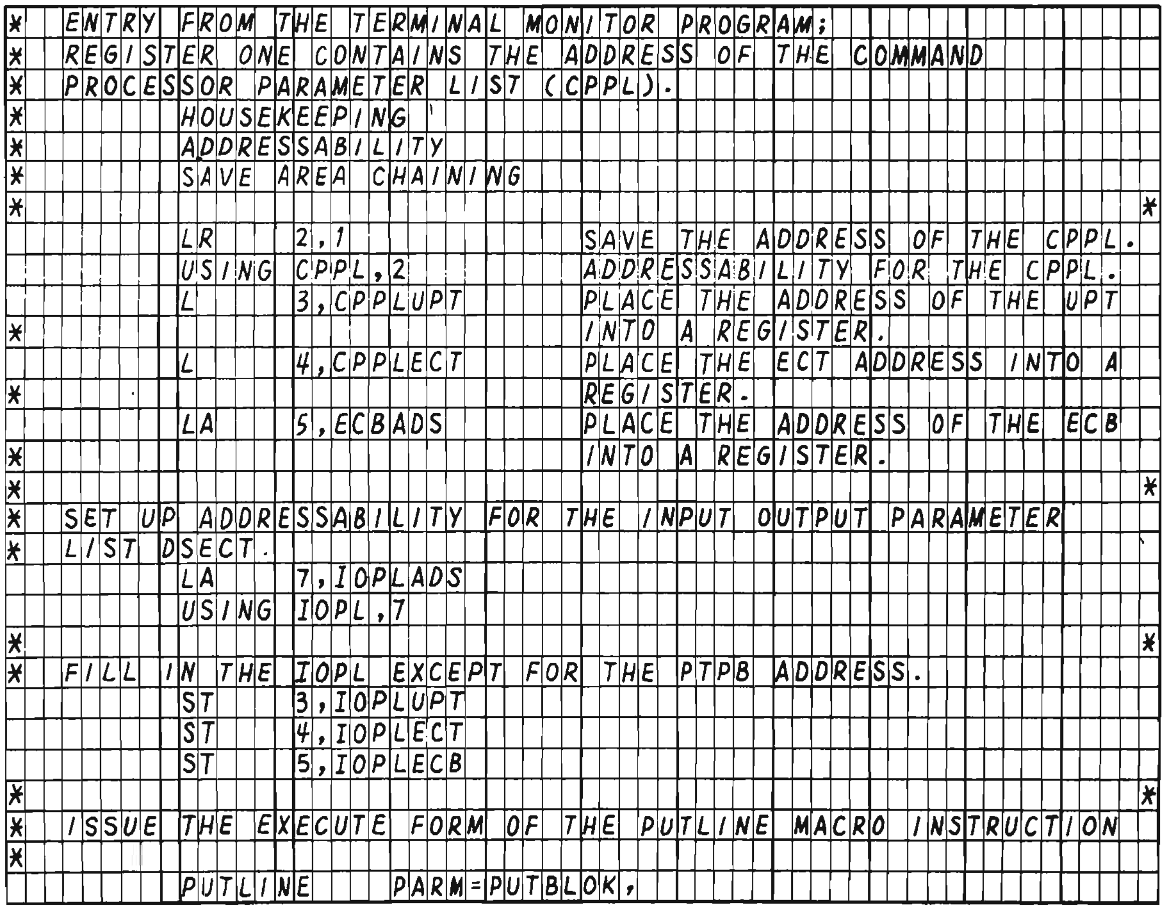


Figure 61. PUTLINE Multiline Data Format

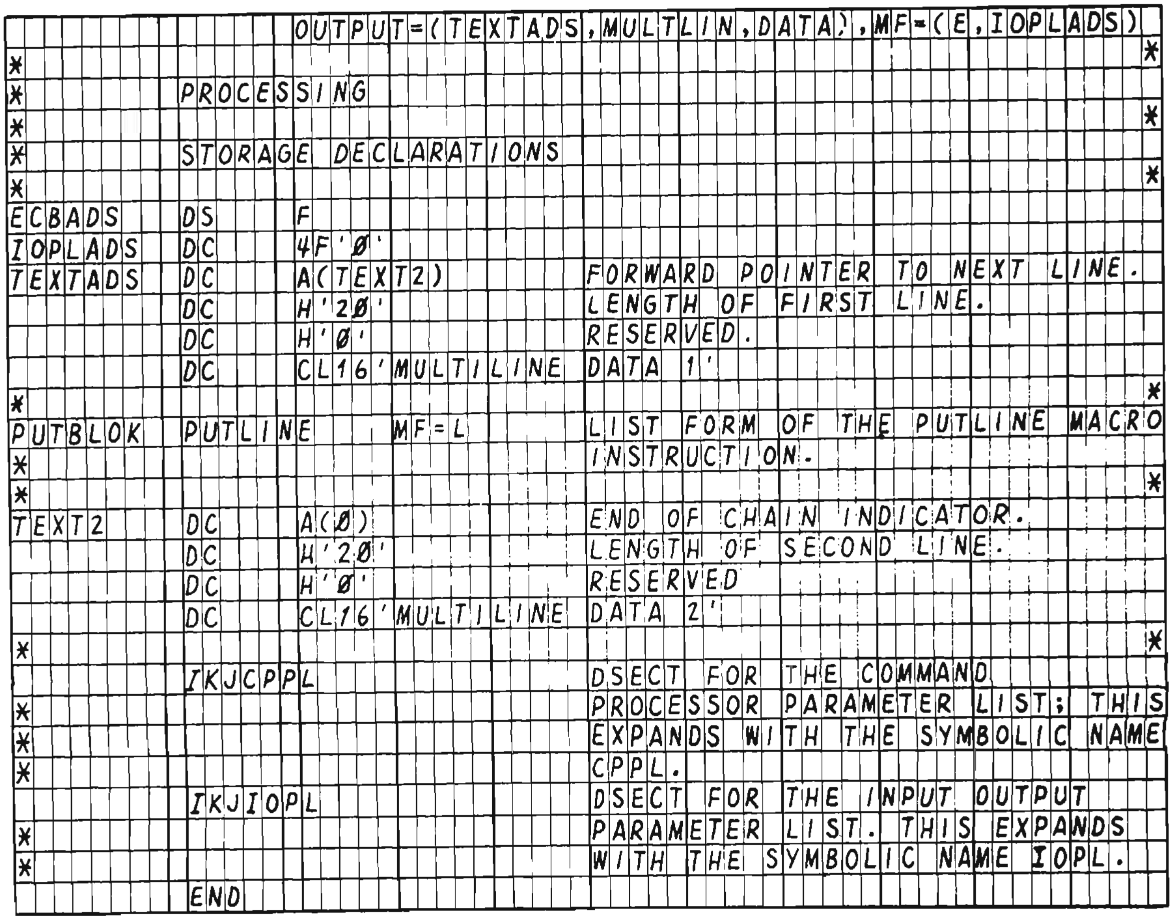
Each of the forward-chain pointers points to the next data line to be written to the terminal. The forward-chain pointer in the last data line contains zeros. In the case of multiline data, you pass the address of the output line to the PUTLINE service routine by coding the beginning address of the first 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 multiline data address into the second word of the PUTLINE parameter block.

Figure 62 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 IOPL and CPPL DSECTs (generated by the IKJIOPL and IKJCPPL macro instructions). 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 62. Coding Example - PUTLINE Multiline 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 62. Coding Example - PUTLINE Multiline Data (Part 2 of 2)

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)
- Multilevel 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 multilevel 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 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 63 shows the format of the output line descriptor.

Number of Bytes	Field	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 multilevel 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.

Figure 63. 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 64 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.

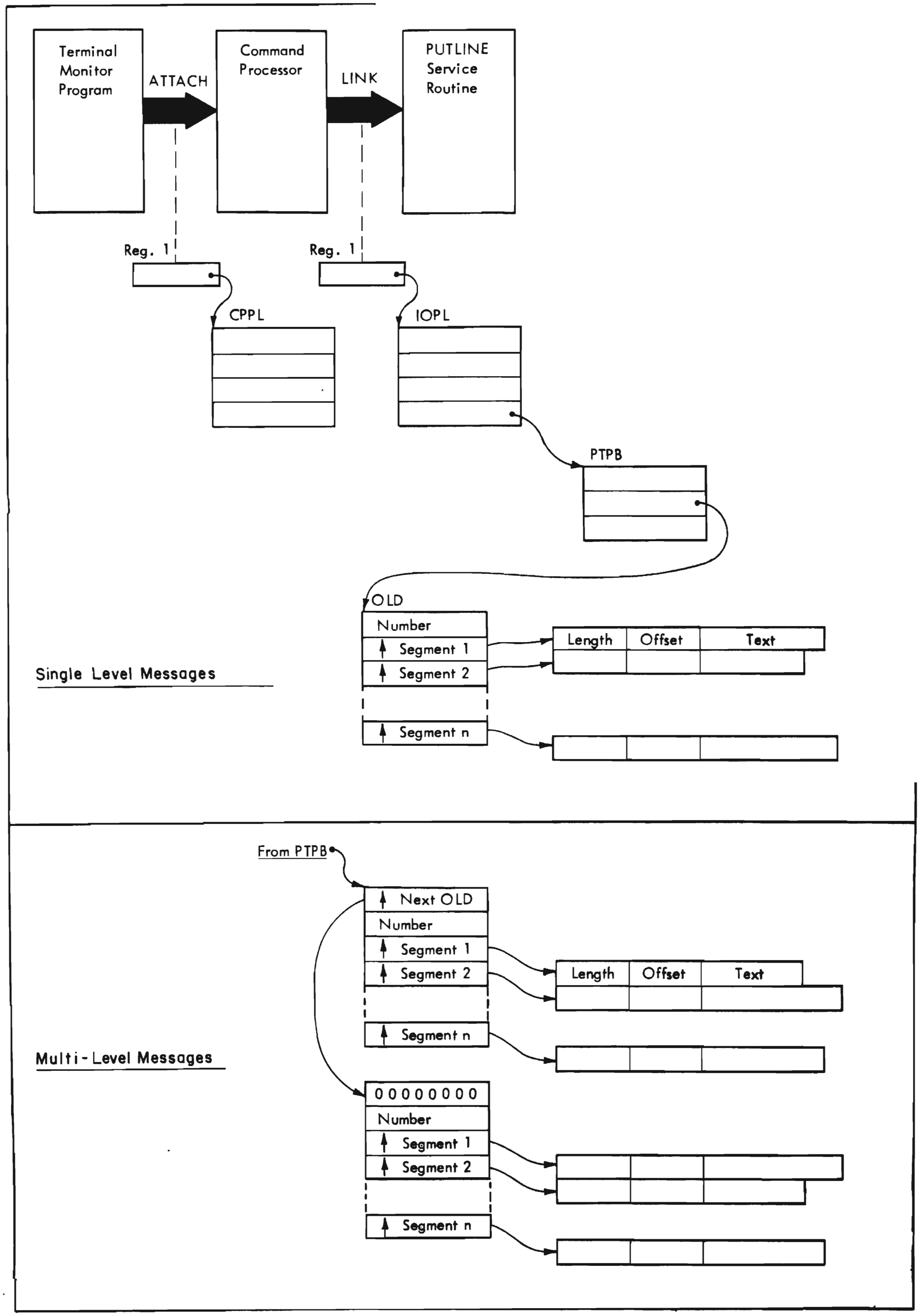


Figure 64. Control Block Structures for PUTLINE Messages

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 65 shows the distribution of these PUTLINE service routine functions over the two output message types.

	Message Types	
Functions	**Single Level**	**Multilevel**
Message ID Stripping	x	x
Text Insertion	x	x
Formatting Only	x	
Second Level Informational Chaining		x

Figure 65. 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 *TSO Command Language Reference* and *TSO 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. If the message segment does not contain at least one blank, PUTLINE will return a code of 12 (invalid parameters) in register 15, even if message ID stripping is not 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 ID 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 ID 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. A message beginning with a blank is unaffected by a terminal user's request for message ID stripping and will appear as you wrote it, minus the blank.

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 an error code of 12 (invalid parameters) in register 15. If the secondary segments do not appear in the OLD with their offsets in ascending order, PUTLINE returns an error 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 on 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	14	TEXT
13	17	CONTINUE

PUTLINE would place the first insert, TEXT, after the 13th 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 66 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.

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.

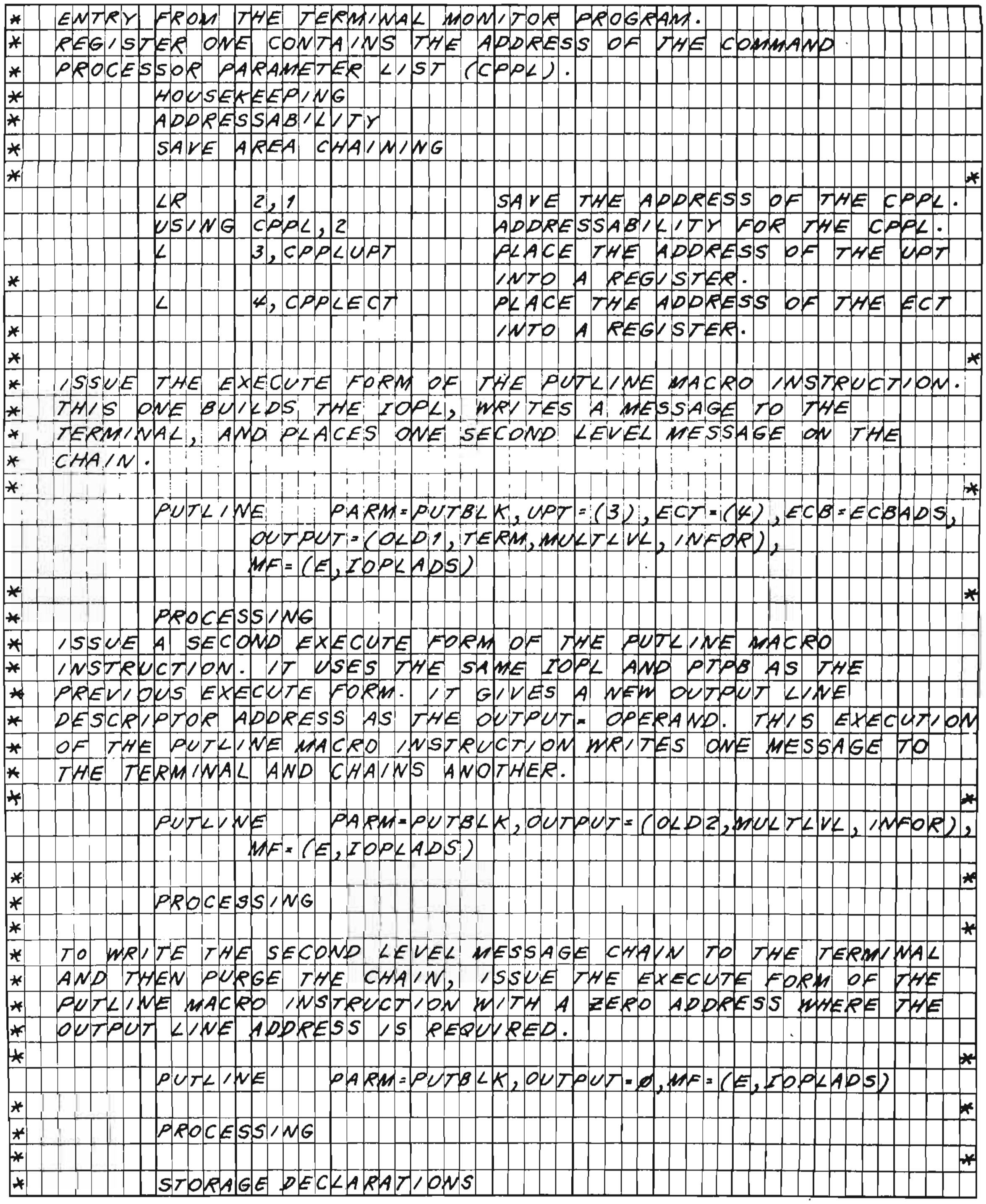
To purge the chain without putting it out to the terminal, you must turn on the high order bit in the first byte (ECTMSGF) of the third word of the environment control table (ECT). The ECT is pointed to by the second word of the input/output parameter list, and may be mapped by the IKJECT DSECT. The 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 67 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.

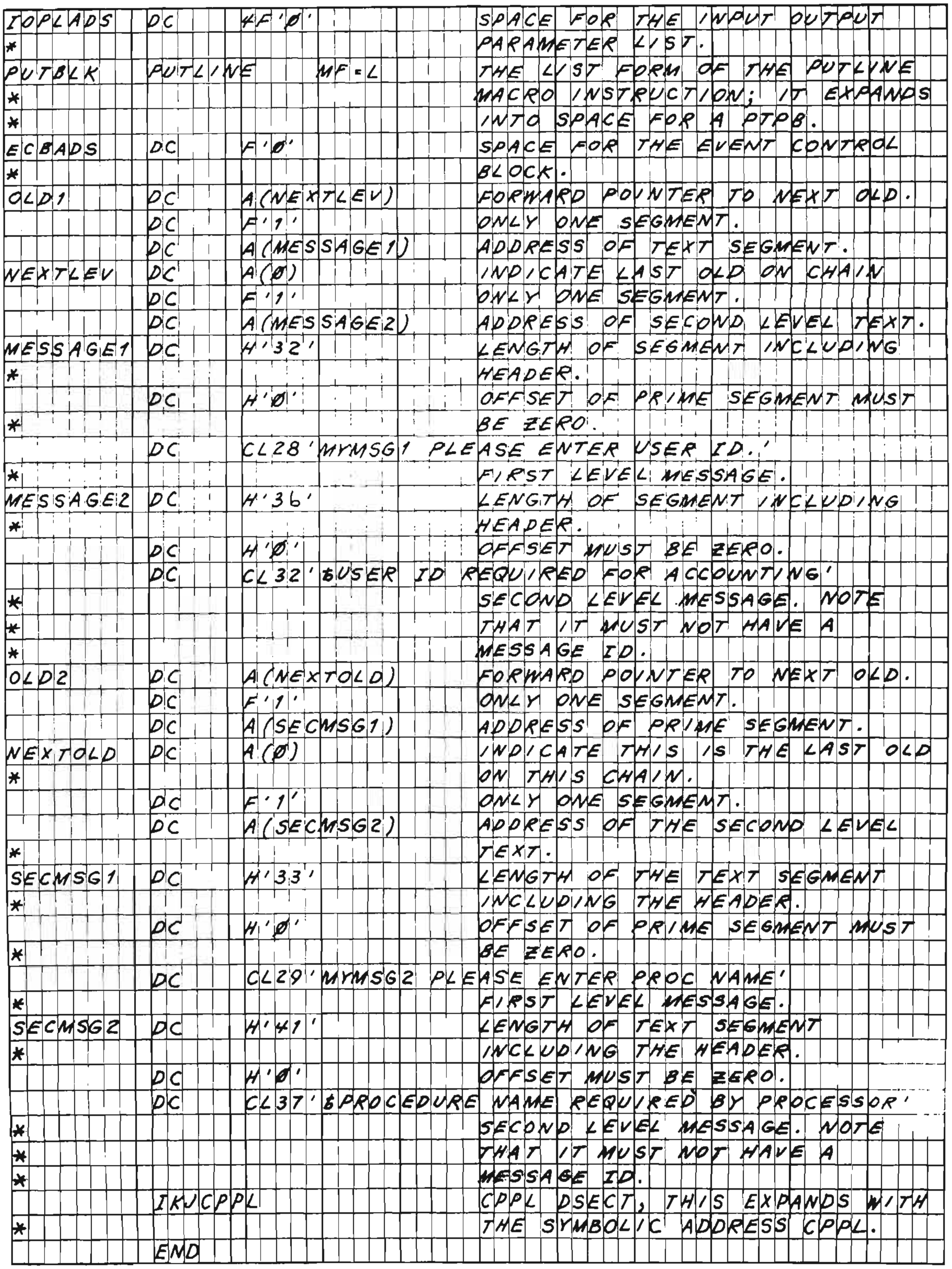
The code shown puts two messages out to the terminal and places two second level messages on an internal chain. It then executes a third execute form of the PUTLINE macro instruction with a zero OUTPUT address to put the second level chain out to the terminal.

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=Ø,MF=(E,IOPLADS)
*                                                                       *
*          PROCESSING
*                                                                       *
*          STORAGE DECLARATIONS
```

Figure 67. 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'36'               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 67. 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 15:

Code	Meaning
decimal	
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 real storage to satisfy the request.
20	The terminal has been disconnected.

Note: The GNRLFAIL service routine described in this book can be invoked to issue a meaningful error message for a PUTLINE error code.

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 68 shows the list form of the PUTGET macro instruction; each of the operands is explained following the figure. Appendix A describes the notation used to define macro *instructions.*

```
[symbol]   PUTGET   [                                        ( ,PROMPT )  ]
                    [ 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 68. 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 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 second 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 be displayed at those terminals that support the print inhibit feature. A terminal user can override bypass processing by hitting an attention followed by hitting the ENTER key 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 cursor 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 item 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 printable characters 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 cursor 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 cursor 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 item c. under EDIT.

If the NL is embedded in your message, it is sent to the terminal as a cursor return. No idle characters are added (see item f. below). This may cause overprinting, particularly on terminals that require a line-feed character to position the cursor 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 display or move the cursor 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 16 (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 displayed 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 cursor 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.

The other operands and their sublists 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.

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 initialize 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 69 shows the execute form of the PUTGET macro instruction; each of the operands is explained following the figure. Appendix A 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 69. The Execute Form of the PUTGET Macro Instruction

PARM=parameter address
Specifies the address of the four-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 parameter 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 1 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 it in one of the general registers 2-12, and use that register number, enclosed in parentheses, as the ECB address.

OUTPUT=output address

Specifies 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 function 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 second level messages are displayed 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.

PTBPYS

The output line is a prompt message and the terminal user's response will not display at those terminals that support the print inhibit feature. A terminal user can override bypass processing by hitting an attention followed by hitting the ENTER key before entering 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 cursor 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 item 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 cursor 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 cursor 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 item c. under EDIT.

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.

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 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 initialize 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 70 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.
	.... .0..	NOBREAK processing has been requested. The output line will be displayed 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 70. The PUTGET Parameter Block (Part 1 of 2)

Number of Bytes	Field	Contents or Meaning
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 70. 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.

Multilevel Message: 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 written 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 *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 multilevel 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. The type depends 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 71 shows the format of the output line descriptor.

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 "PUTLINE Message Line Processing."

The PUTGET service routine provides other processing capabilities dependent upon whether the message is a mode or a prompt message.

1. 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 can 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 second level message (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 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 message 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 message 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 and reissuing the PUTGET. The second level message 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 second level messages available to him, even if the current input source is not the terminal.

PAUSE processing works as follows. If a second level message 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 second level messages put out to the terminal, or press the ENTER key to indicate that the information is not needed. If the user presses the ENTER key, the second level message is eliminated. If he enters any response other than a question mark or hitting the ENTER key, PUTGET prompts him for a correct response.

2. ***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 second 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. You may free the input buffer with the FREEMAIN macro instruction after you have processed or copied the input line.

Regardless of the source of input, 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 73 shows the format of the input buffer returned by the PUTGET service routine.

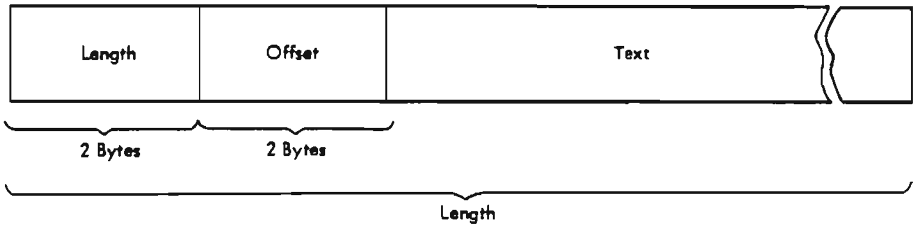


Figure 73. Format of the PUTGET Input Buffer

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 74 shows the PUTGET control block structure for a multilevel PROMPT message after the PUTGET service routine has returned an input line.

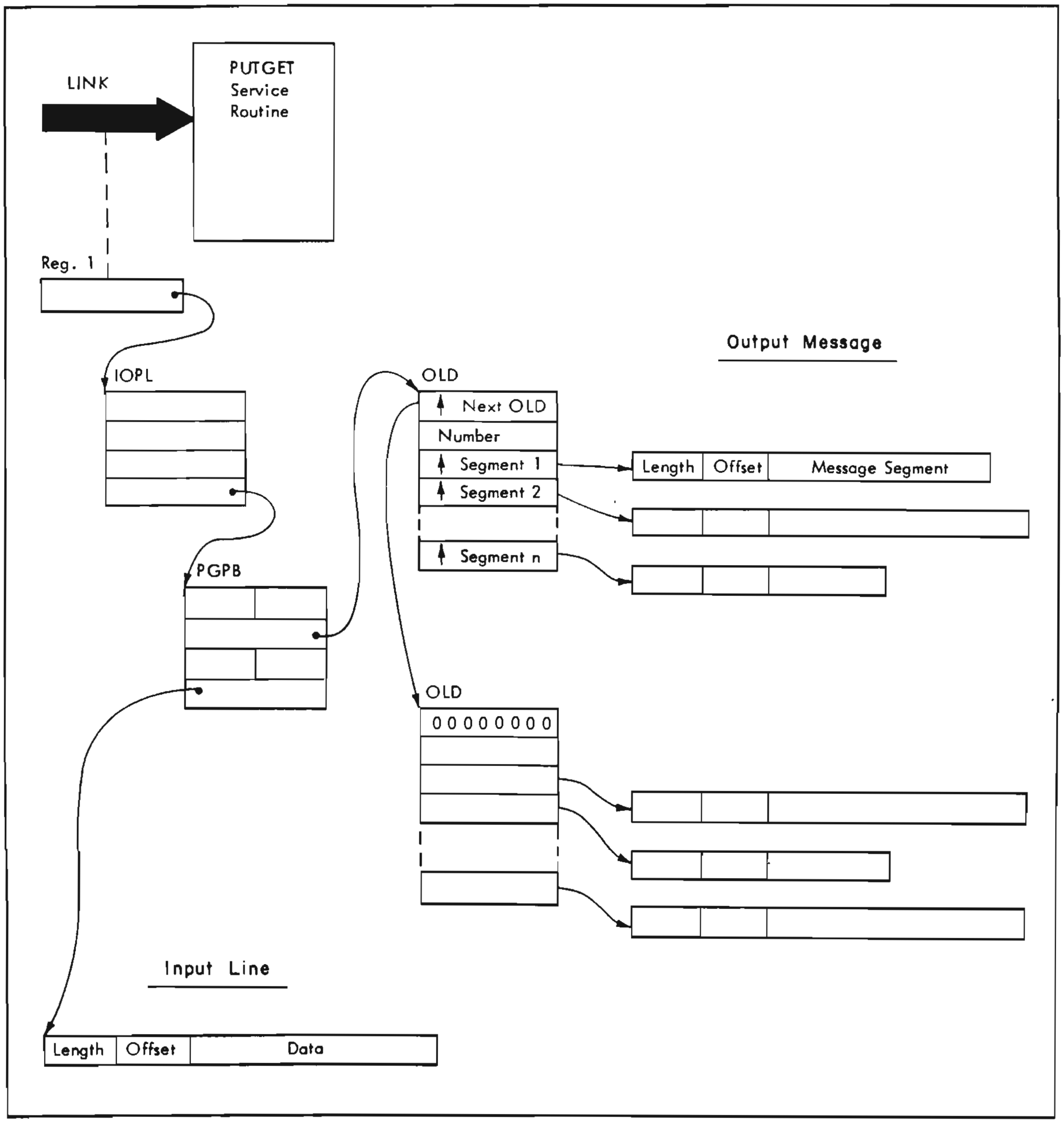


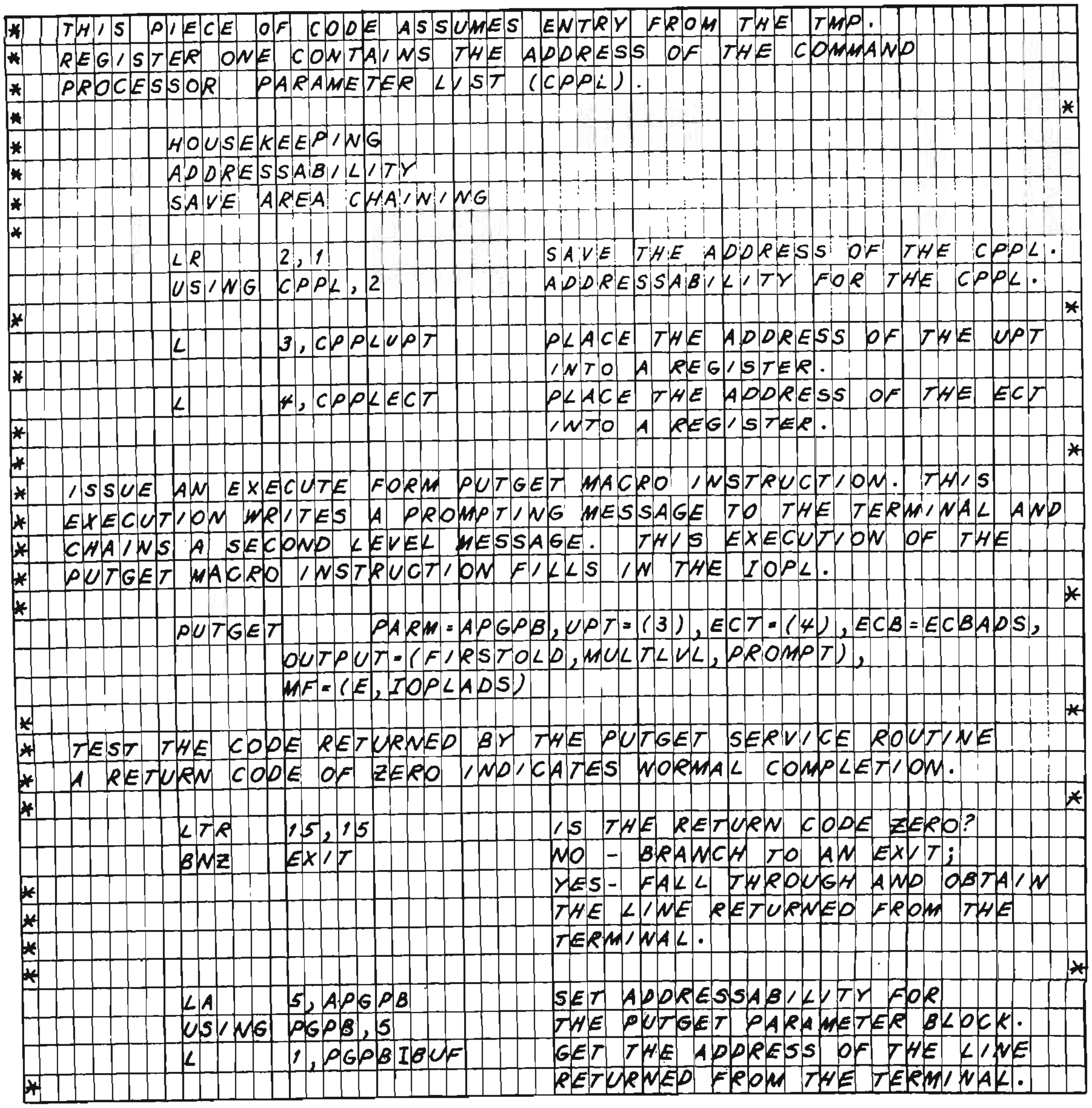
Figure 74. PUTGET Control Block Structure - Input Line Returned

An Example of PUTGET

Figure 75 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).

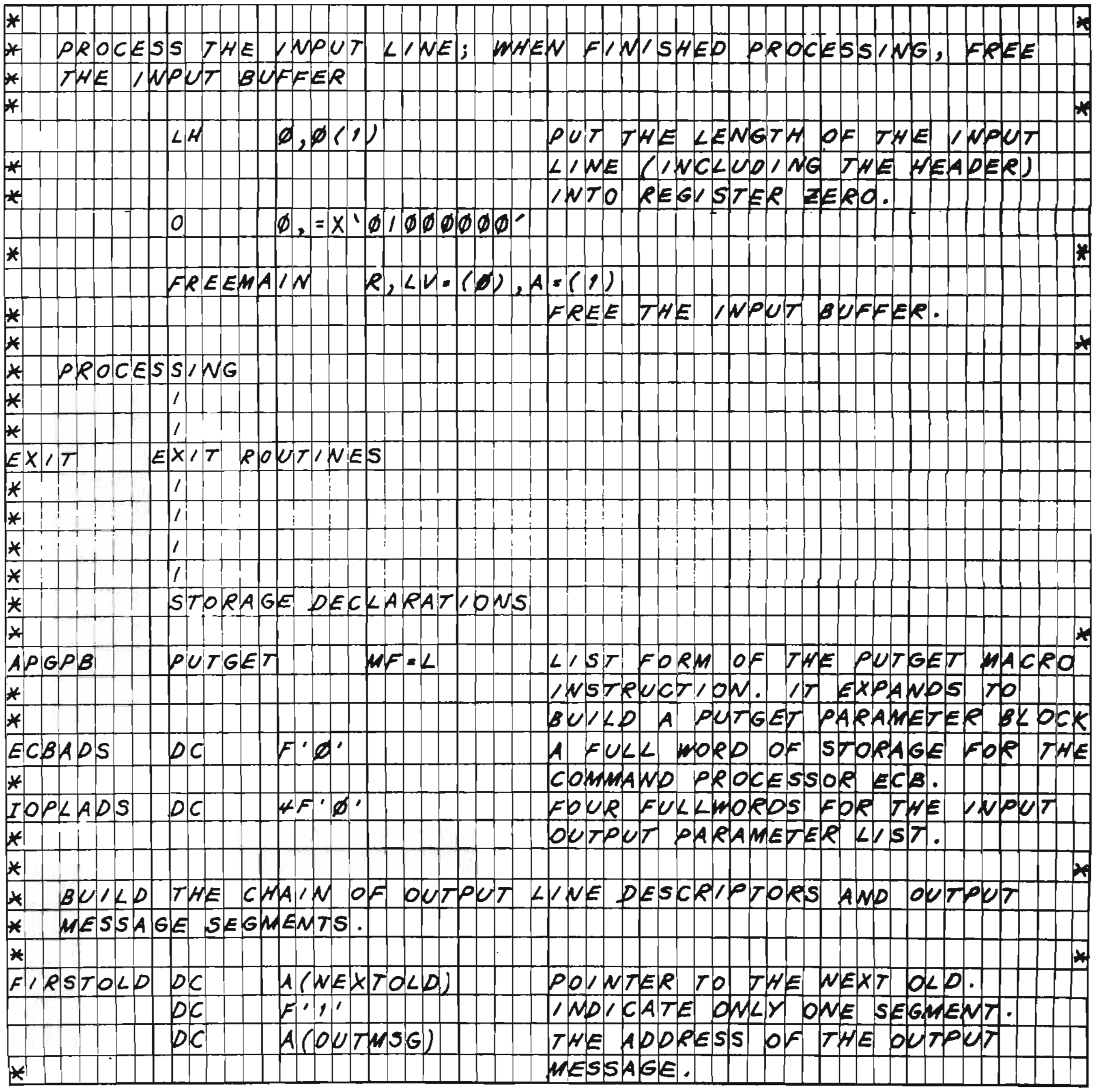
The execute form of the PUTGET macro instruction builds the I/O parameter list, using the addresses of the user profile table and the environment control table supplied in the command processor parameter list. In addition, the I/O parameter list contains the address of an ECB built by the code, and the 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 terminal as the current source of input.



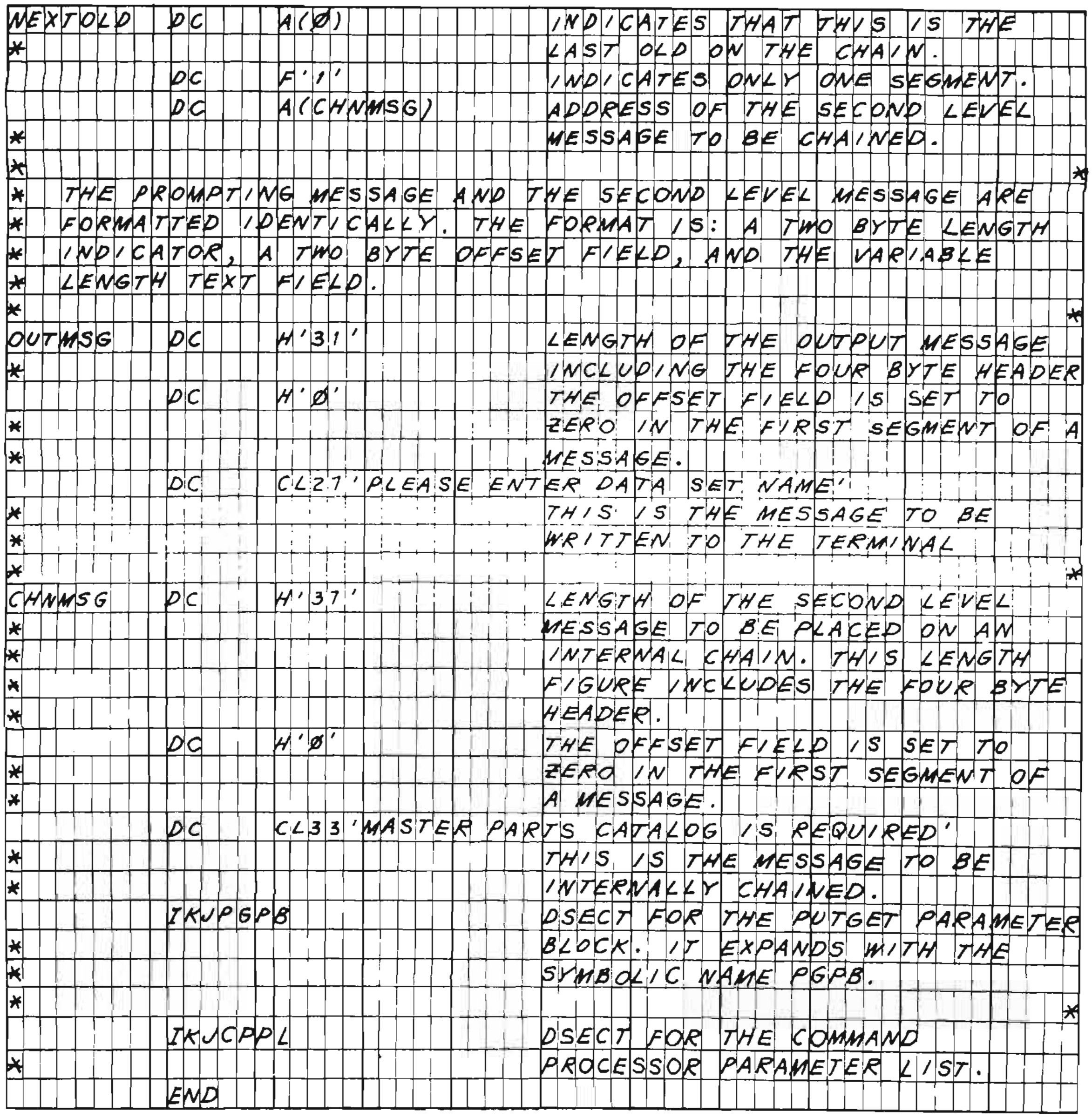
```
*  THIS PIECE OF CODE ASSUMES ENTRY FROM THE TMP.
*  REGISTER ONE CONTAINS THE ADDRESS OF THE COMMAND
*  PROCESSOR  PARAMETER LIST (CPPL).
*                                                                      *
*          HOUSEKEEPING
*          ADDRESSABILITY
*          SAVE AREA CHAINING
*
         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 75. Coding Example - PUTGET Multilevel 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.
       O     Ø,=X'Ø1ØØØØØØ'
*                                            *
       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 75. Coding Example - PUTGET Multilevel PROMPT Message (Part 2 of 3)



```
NEXTOLD DC   A(Ø)        INDICATES 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'31'       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   CL27'PLEASE ENTER DATA SET NAME'
*                        THIS IS THE MESSAGE TO BE
*                        WRITTEN TO THE TERMINAL
*                                                              *
CHNMSG  DC   H'37'       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   CL33'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 75. Coding Example - PUTGET Multilevel 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 interruption 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 other than an EXEC command procedure.
12	A line could not be obtained after a MODE request. Second level messages exist, and the current stack element is not a 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.
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.
32	The terminal has been disconnected.

Note: For a discussion of register contents and parameter list expansions for TPUT, see "TGET/TPUT/TPG Parameter Formats" later in this section.

```
[symbol]     TPUT     [buffer address,buffer size                                              ]
                      [ [,EDIT    ]                                                            ]
                      [ [,NOEDIT  ]                                                            ]
                      [ [,ASIS    ]  [,WAIT  ]  [,NOHOLD]  [,NOBREAK]                          ]
                      [ [,CONTROL ]  [,NOWAIT]  [,HOLD  ]  [,BREAKIN]                          ]
                      [ [,FULLSCR ]                                                            ]
                      [                         [,HIGHP]   [,ASID=id         ]                 ]
                      [                         [,LOWP ]   [,ASIDLOC=address ]                 ]
                      [                                    [,USERIDL=address ]                 ]
                      [ [,R                ]                                                   ]
                      [ [,MF={L           }]                                                   ]
                      [ [     (E,ctrladdr)}]                                                   ]
```

Figure 76. The TPUT Macro Instruction -- Standard, Register, List, and Execute Forms

buffer address

Standard form: The address of the buffer that holds your line of output. You may specify any label valid in an RX instruction, or place the address of the label in one of the general registers 1-12, and then specify that register within parentheses.

Register form: The register that 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.

buffer size

Standard form: The size of the output buffer in bytes. The allowable range is from 0 through 32,767 bytes. A buffer size of 0 results in no data being transmitted to the terminal. 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 that 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.

Notes:

1. If the registers you specify as the first and second operands in the register form of TPUT are registers 1 and 0 respectively, the TPUT macro instruction will expand directly into the TGET/TPUT/TPG SVC. However, if you use registers 2-12, the macro expansion will load registers 1 and 0 from the registers you specify before issuing the SVC. Therefore, you might find it advantageous to use registers 1 and 0. (The expansion destroys the contents of registers 1 and 0.)
2. If QSAM is used for terminal I/O and a data set is defined with BLKSIZE=80 and RECFM=U, each line will be truncated by 1 character. This byte (the last byte) is reserved for an attribute character.

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 macro instruction.

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.

MF=

Indicates the form of the TPUT macro instruction.

L

Specifies the list form.

(E,ctrl addr)

Specifies the execute form and the address of the list form.

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 cursor to the beginning of the next line.

c. All terminal control characters (except backspace) are replaced with a printable character.

EDIT is the default value for the EDIT, ASIS, CONTROL, FULLSCR, and NOEDIT operands.

NOEDIT

Indicates that, if the terminal is an IBM 3270 display, the message is transmitted completely unedited. It is assumed that the command processor using this option has structured the data stream with the necessary commands to perform the display function. For LU__T1 terminals, this option is converted to ASIS.

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 are eliminated. For example, restore, uppercase, lower case, bypass, and bell ring might be valid but unprintable characters at some terminals. (See CONTROL.)

b. Transmission control characters are added.

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

If you specify EDIT, NL is handled as described in item c under EDIT.

If the NL is embedded in your message, a semicolon or colon may be substituted for NL and sent to the terminal. No idle characters are added (see item 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. If the output line is longer than the terminal line size, control characters are added as needed to cause the message to display on several lines. 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 cursor is being returned to the left-hand margin.

g. Including a bypass character, bypass carriage return, or bypass new-line character in the TPUT macro data suppresses printing of the next input entered by the user at the 3270 terminal. VTAM moves the cursor to the next available line, unlocking the keyboard. No more data is sent to the terminal until the terminal user enters data or presses the ENTER key. The data entered by the user is not printed at the terminal.

CONTROL

Indicates that this line is composed of terminal control characters and does not display or move the cursor on the terminal. This option should be used for transmission of characters such as bypass, restore, or bell ring. See item g under ASIS.

FULLSCR

Indicates that, for IBM 3270 display terminals, the message will be transmitted essentially unedited. The FULLSCR option is designed to allow you to use special features of the 3270 system. For any other terminal type, this option is treated exactly as ASIS. With the FULLSCR option, only the following editing is performed:

a. If the first character in your message is an escape control character (X‘27’), the two following characters are treated as a command code and as a write control character by the 3270. Note that the command code should always be for a remote 3270. If necessary, TPUT will convert the code to that for a local 3270. If the first character is not an escape character, a default write command and a write control character are added to the beginning of the message. Any attachment-dependent characters required for correct transmission of the data stream are provided by the access method.

b. Transmission control characters (SOH, STX, ETX, ETB, EOT, and NAK) and characters having no 3270 equivalent (X‘04’, X‘06’, X‘14’. through X‘17’, and X‘24’) are converted to printable colons to prevent program-caused I/O errors.

Lines are not counted when you use this option.

If the OWAITHI value specified in your TSO parameters is not large enough to contain your entire message, or if the BUFFERS and BUFFERSIZE parameters are specified so that your message does not fit into all of the system's buffers, the TPUT operation does not proceed, and code X'10' is returned. For a description of OWAITHI, refer to *SPL: Initialization and Tuning.* Without the FULLSCR option, your TPUT proceeds buffer-by-buffer as buffers become available.

If FULLSCR is specified for a message destined for another terminal, ASIS will be used instead.

WAIT

Specifies that control is not be returned to the program that issues the TPUT macro instruction until the output line is placed into a terminal output buffer. If no buffers are available, the issuing program is 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 is returned to the program that issues 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 in register 15.

NOHOLD

Indicates that control is returned to the program that issues the TPUT macro instruction as soon as the output line is placed in terminal output buffers.
NOHOLD is the default value for the NOHOLD and HOLD operands.

HOLD

Specifies that the program that issues the TPUT macro instruction cannot continue its processing until this output line is written to the terminal or deleted.

NOBREAK

Specifies that if the user starts to enter input, he is not interrupted. The output message is placed on the output queue and displayed after the user completes the line.
NOBREAK is the default value for the NOBREAK and BREAKIN operands.

BREAKIN

Specifies that output has precedence over input. If the user starts to enter input, he is interrupted, and this output line is displayed. Any data received before the interruption is displayed following this output line.

HIGHP

Specifies that this message must be sent to the terminal, even though the destination terminal does not display messages from other terminals. This operand counters the effect of the interterminal communication bit when the bit is set by the TERMINAL command. (The HIGHP operand is used by the SEND subcommand of OPERATOR and the SEND operator command.) The operand is recognized only if the issuing task is authorized (via system key, supervisor state, or APF). The ASID keyword must also be specified. HIGHP is the default if neither HIGHP nor LOWP is specified, and if the issuing program is authorized.

LOWP

Specifies that if the user of the destination terminal allows interterminal messages, this TPUT will be sent to the terminal. (TPUT tests the interterminal communication bit in the terminal status block). If such messages are not allowed, the message is not displayed, and a code of X'0C' is returned, indicating that the message was not displayed. The LOWP operand is recognized only when ASID is specified. The issuer must be authorized (via system key, supervisor state, or APF).

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.

ASID, ASIDLOC, or USERIDL

Specifies the ASID (address space identifier) of the target terminal, the address of that ASID, or the address of a field that contains a user ID. This facility is used for supervisor communication with the terminal and for inter-user communication among terminals (the SEND command). If you specify ASID, you must supply an ASID number. The ASID is located in the two-byte JSCBTJID field of the job step control block. If you use ASIDLOC, you must supply the address of the halfword that contains the ASID. If you use USERIDL, you must supply the address of the eight-byte field that contains the user ID. The user ID must be left-justified and, if necessary, padded with blanks. ASID, ASIDLOC, or USERIDL can be specified in a register (2-12), and must be right-justified. The register number must be enclosed in parentheses. If USERIDL is used, the NOHOLD option is both required and the default if not specified.

Note: Normally, a program invokes TPUT to issue a message to the user running the program -- that is, ASID, ASIDLOC, and USERIDL are not specified. If that program is run in the background, the TPUT has no effect.

If the TPUT specifies an ASID or user ID, the message is sent to the target terminal. ASID and USERID TPUTs from programs not in supervisor state or not authorized under APF are prefixed with a plus sign (+) by SVC 93 to prevent possible counterfeiting of system messages to an operator console.

Return Codes from TPUT

When it returns control to the program (foreground or background) 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. The message was not sent.
0C	A TPUT macro instruction with an ASID operand was issued but the user, indicated by the ASID, requested that interterminal messages not be printed on his terminal. The message was not sent.
10	Invalid parameters were passed to the TPUT SVC.
14	The terminal was disconnected and could not be reached.

The TPG Macro Instruction -- Writing a Line Causing Immediate Response

Use the TPG macro instruction (SVC 93) to transmit a line of output to the terminal if that line of output will cause the device to respond immediately with input. The main use of TPG is to perform the Query function for a user who has included a Read Partition Structured field. TPG NOEDIT creates an outbound request unit with an associated change direction indicator to allow the device to go into send state. This data is not inspected. A TGET macro must be issued to retrieve the query response.

You can use the TPG macro instruction in any TSO routines you write, and in any application programs to be run under TSO.

Note: The TPG macro instruction is designed specifically to allow you to use the special features of the IBM 3279.

The TPG macro instruction must be issued in 24-bit addressing mode. All input specified on the macro must reside below 16 megabytes.

Figure 77 shows the format of the TPG macro instruction. Each of the operands is explained following the figure. Appendix A describes the notation used to define macro instructions. For a discussion of parameter list expansions for TPG, see "TGET/TPUT/TPG Parameter Formats" later in this section.

[symbol]	TPG	buffer address,buffer size [[,NOEDIT] [,WAIT ,NOWAIT] [,NOHOLD ,HOLD] [,MF={L (E,ctrladdr)}]]

Figure 77. The TPG Macro Instruction -- Standard, List, and Execute Forms

Return Codes from TPUT

When it returns control to the program (foreground or background) 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. The message was not sent.
0C	A TPUT macro instruction with an ASID operand was issued but the user, indicated by the ASID, requested that interterminal messages not be printed on his terminal. The message was not sent.
10	Invalid parameters were passed to the TPUT SVC.
14	The terminal was disconnected and could not be reached.

The TPG Macro Instruction -- Writing a Line Causing Immediate Response

Use the TPG macro instruction (SVC 93) to transmit a line of output to the terminal if that line of output will cause the device to respond immediately with input. The main use of TPG is to perform the Query function for a user who has included a Read Partition Structured field. TPG NOEDIT creates an outbound request unit with an associated change direction indicator to allow the device to go into send state. This data is not inspected. A TGET macro must be issued to retrieve the query response.

You can use the TPG macro instruction in any TSO routines you write, and in any application programs to be run under TSO.

Note: The TPG macro instruction is designed specifically to allow you to use the special features of the IBM 3279.

The TPG macro instruction must be issued in 24-bit addressing mode. All input specified on the macro must reside below 16 megabytes.

Figure 77 shows the format of the TPG macro instruction. Each of the operands is explained following the figure. Appendix A describes the notation used to define macro instructions. For a discussion of parameter list expansions for TPG, see "TGET/TPUT/TPG Parameter Formats" later in this section.

```
[symbol]  TPG   buffer address,buffer size
                [ [,NOEDIT] [,WAIT   ] [,NOHOLD] ]
                [           [,NOWAIT ] [,HOLD  ] ]
                [ [,MF= {L             } ]       ]
                [ [     {(E,ctrladdr)  } ]       ]
```

Figure 77. The TPG Macro Instruction -- Standard, List, and Execute Forms

buffer address

Standard form: The address of the buffer that holds your output data. You may specify any address valid in an RX instruction, or place the address in one of the general registers 1-12, and then specify that register within parentheses.

buffer size

Standard form: The size of the output buffer in bytes. The allowable range is from 0 through 32,767 bytes. A buffer size of 0 results in no data being transmitted to the terminal. 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.

Note: The R format may not be used for the TPG macro.

NOEDIT

Indicates that, if the terminal is an IBM 3270 display, the message is transmitted completely unedited. The command processor using this option must structure the data stream with the necessary commands to perform the display function (by including the command, write control character, structured fields,...). The command processor should supply only the data stream. Any attachment-dependent characters (such as X'27' for bisynchronous devices) are provided by the access method. For LU_T1 terminals, this option is treated exactly like the ASIS option of the TPUT macro.

WAIT

Specifies that control is not returned to the program that issued the TPUT macro instruction until the output line is placed into a terminal output buffer. If no buffers are available, the issuing program is 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 is returned to the program that issued the TPG macro instruction, whether or not a terminal output buffer is available for the output line. If no buffer is available, the TPG SVC returns a code of 04 in register 15.

NOHOLD

Indicates that control is returned to the program that issued the TPUT macro instruction as soon as the output line is placed in terminal output buffers.
NOHOLD is the default value for the NOHOLD and HOLD operands.

HOLD

Specifies that the program that issued the TPG macro instruction cannot continue its processing until this output line is written to the terminal or deleted.

MF=

Indicates the form of the TPG macro instruction.

L

Specifies the list form.

(E,ctrl addr)

Specifies the execute form and the address of the list form.

Note: If a TPG macro is coded in a background program, the TPG is ignored.

Return Codes from TPG

When it returns control to the program (foreground or background) that invoked it, the TPG SVC supplies one of the following return codes in general register 15:

Code (hexadecimal)	**Meaning**
00	TPG completed successfully.
04	NOWAIT was specified and no terminal output buffer was available.
08	An attention interruption occurred while the TPG SVC routine was processing. The message was not sent.
10	Invalid parameters were passed to the TPG SVC.
14	The terminal was 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 cursor 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 routine, and in any application program that is 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 X'0C' indicates that only part of the line was obtained by TGET. You must then issue as many TGET macro instructions as are required to get the rest of the line of input.

The TGET macro instruction must be issued in 24-bit addressing mode. All input specified on the macro must reside below 16 megabytes.

Figure 78 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 A describes the notation used to define macro instructions.

For a discussion of register contents and parameter list expansions for TGET, see "TGET/TPUT/TPG Parameter Formats" later in this section.

```
[symbol]   TGET   buffer address,buffer size [[,EDIT] [,WAIT  ]                 ]
                                             [[,ASIS] [,NOWAIT]                 ]
                                             [ [,R                       ]      ]
                                             [ [,MF={L                }  ]      ]
                                             [       {(E,ctrladdr)}             ]
```

Figure 78. The TGET Macro Instruction -- Standard, Register, List, and Execute Forms

buffer address

Standard form: The address of the buffer that is to receive the input line. This can be any address valid 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 that 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.

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. A TGET with a 0-length buffer size will successfully get a null line.

Register form: The register that 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.

Note: If the registers you specify as the first and second operands in the register form of TGET are registers 1 and 0 respectively, the TGET macro instruction will expand directly into the TGET/TPUT/TPG SVC. However, if you use registers 2-12, the macro expansion will load registers 1 and 0 from the registers you specify before issuing the SVC. Therefore, you might find it advantageous to use registers 1 and 0.

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 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 (nongraphic characters such as bypass, line feed, restore, prefix and the character immediately following it) are removed from the data.

b. When backspace is not used for character deletion, the horizontal tab (HT) and the backspace (BS) characters, remain in the data.

c. If the returned input line is shorter than the input buffer length, the buffer is padded with blanks. 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), cursor 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 1.

e. After the input message is received, the cursor is returned to the left-hand margin of the next line before any output to the terminal is displayed.

WAIT

Specifies that control is not returned to the program that issues the TGET macro instruction until the input line is 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, whether or not an input line is available from the terminal, control is returned to the program that issues the TGET macro instruction. If no line is returned, the TGET SVC returns a code of X'04' in register 15.

MF=

Indicates the form of the TGET macro instruction.

L

Specifies the list form.

(E,ctrladdr)

Specifies the execute form and the address of the list form.

Return Codes from TGET

When it returns control to the program that invokes it, the TGET SVC supplies, in register 1, the length of the message moved into your buffer, and, in register 15, one of the following return codes:

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. The message was not received.
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 was disconnected and could not be reached.
18	TGET completed successfully. Register 1 contains the length of the input line read into your input buffer. The data was received in NOEDIT mode.
1C	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. The data was received in NOEDIT mode.

TGET/TPUT/TPG Parameter Formats

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/TPG SVC into two registers. 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.

Note: For TPUT, the expansion destroys the contents of registers 0 and 1.

If the registers you specify as the first and second operands 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 R format may not be used for the TPG macro.

For the TPUT macro, the registers must be formatted as shown in Figure 79.

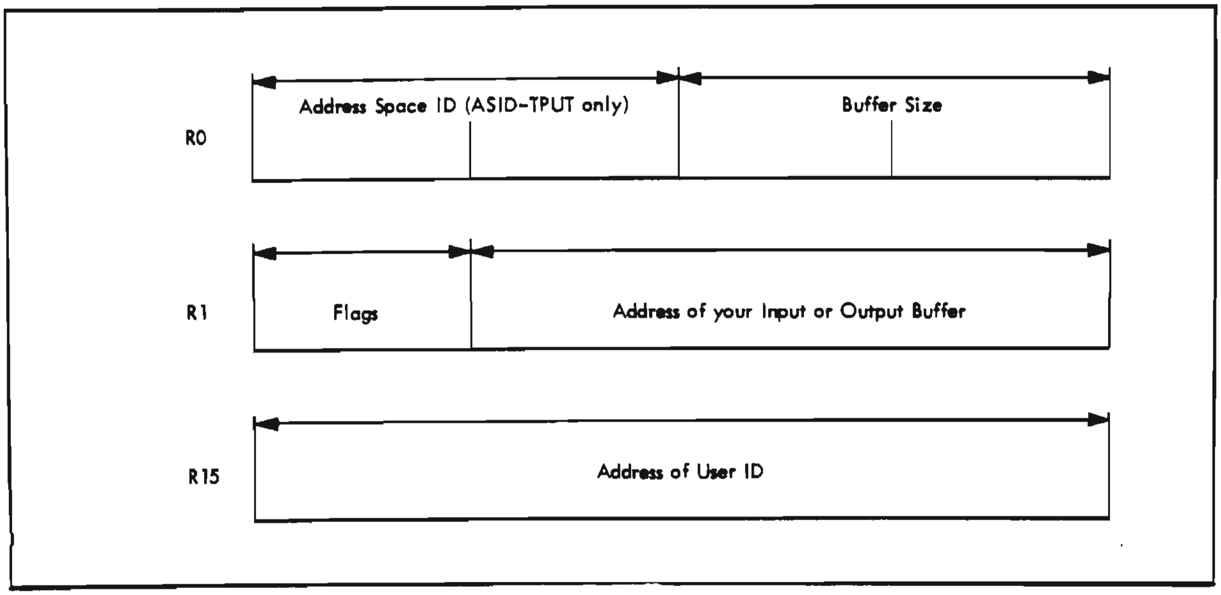

R0
Address Space ID (ASID-TPUT only)
Buffer Size
R1
Flags
Address of your Input or Output Buffer
R15
Address of User ID

Figure 79. TPUT Parameter Registers

For the TGET macro, the registers must be formatted as shown in Figure 80.

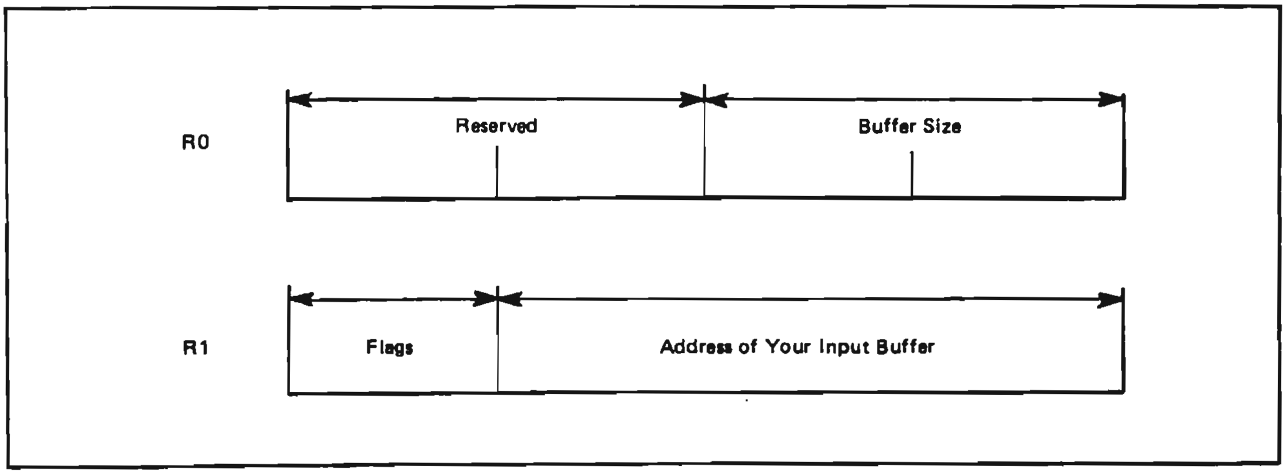


Figure 80. TGET Parameter Registers

Flags/Flag1
One Byte

0...	Always set to 0 for TPUT.
1...	Always set to 1 for TGET.
.0..	No user ID.
.1..	Register 15 contains address of user ID.
..0.	HIGHP processing is requested.
..1.	LOWP processing is requested.
...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.
.... ..11	FULLSCR processing is requested.

If you use the execute form of the TPUT macro, the coded parameters expand into the parameter list shown in Figure 81.

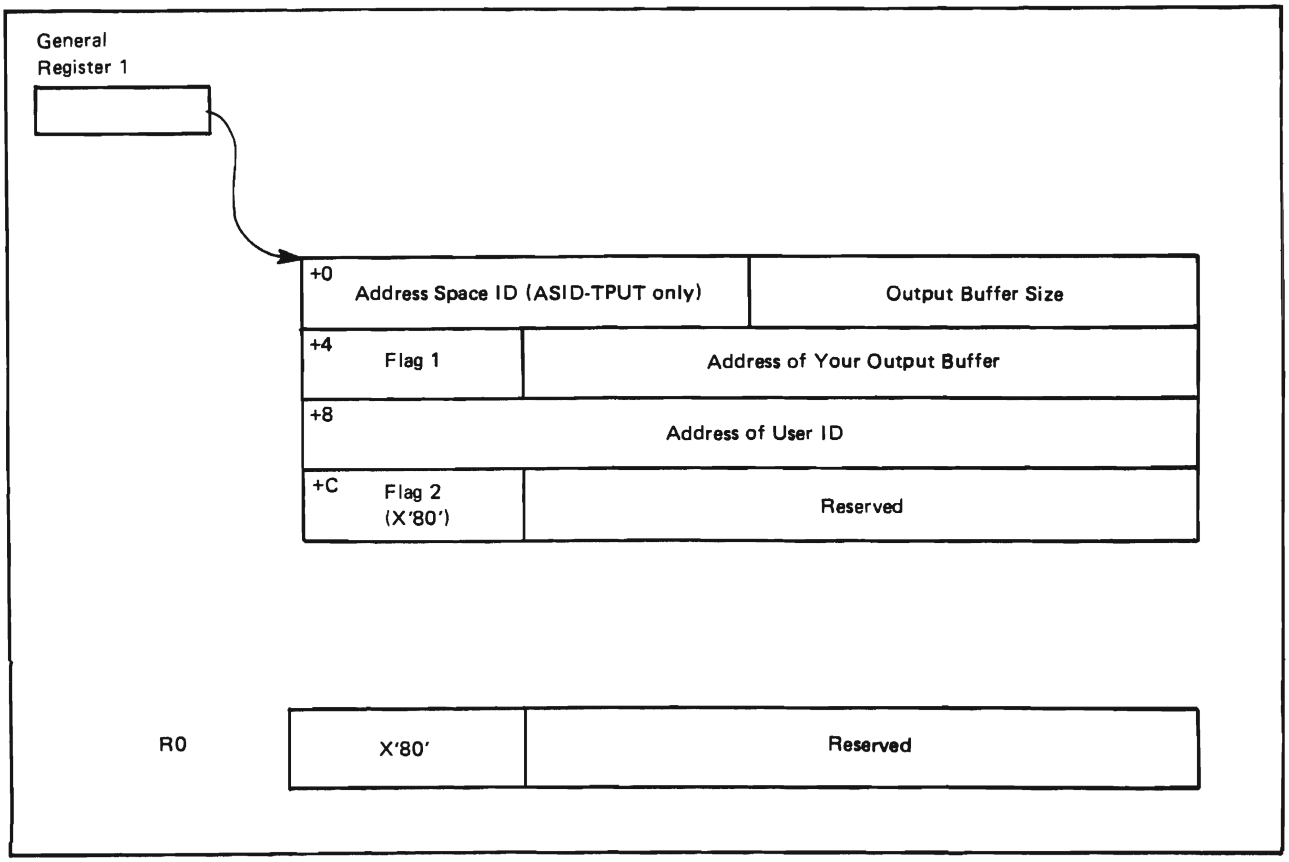


Figure 81. Parameter List Expansion for the Execute Form of TPUT

If you use the standard form of the TPUT macro, you can code your parameters using registers or symbols. In this case, the TPUT macro expands to load the parameters into registers 0, 1, and 15 in the format illustrated in Figure 79.

If you use the list form of the TPUT macro, the coded parameters expand into the parameter list shown in Figure 82.

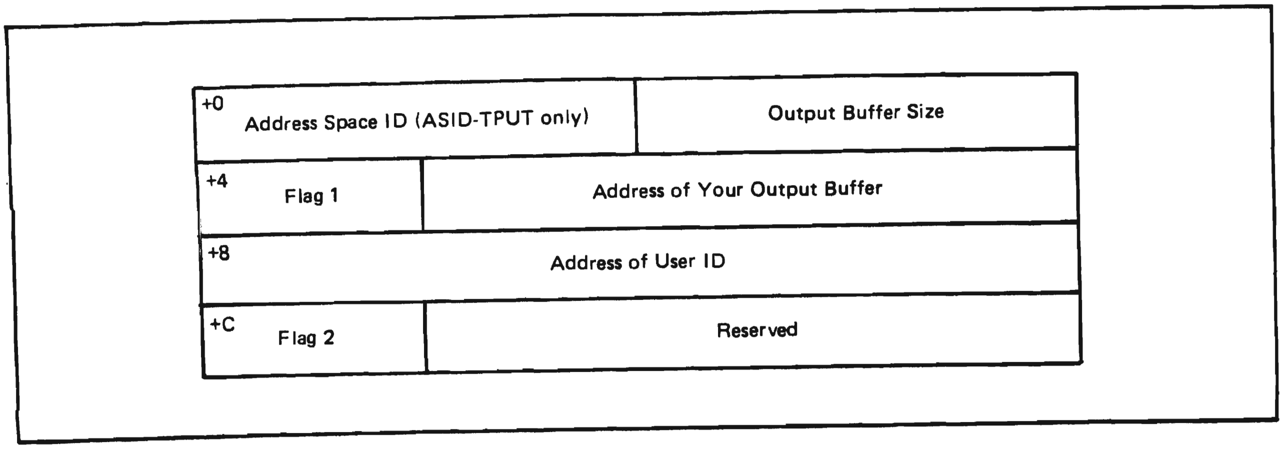


Figure 82. Parameter List Expansion for the List Form of TPUT

If you use the execute form of the TPG macro, the coded parameters expand into the parameter list shown in Figure 83.

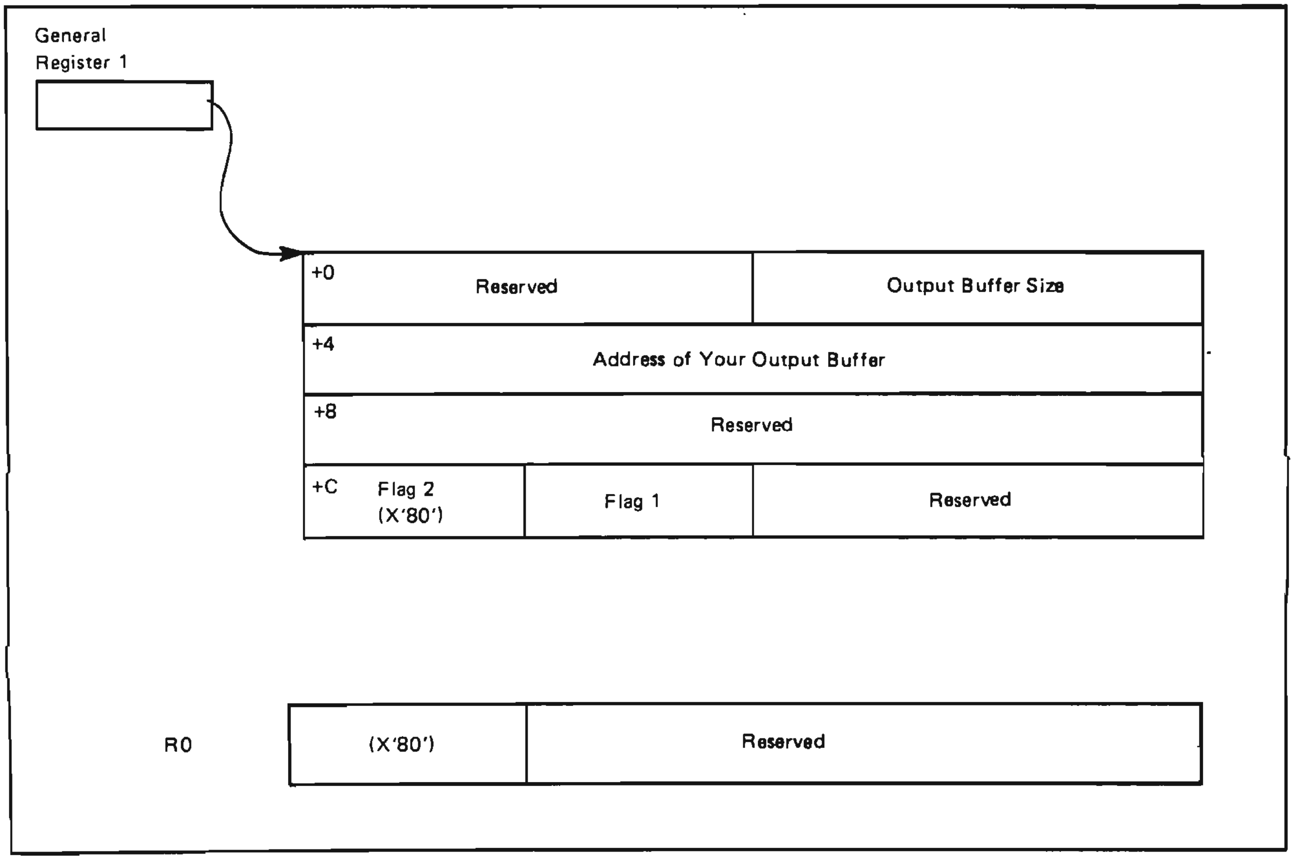


Figure 83. Parameter List Expansion for the Execute Form of TPG

If you use the list form of the TPG macro, the coded parameters expand into the parameter list shown in Figure 84.

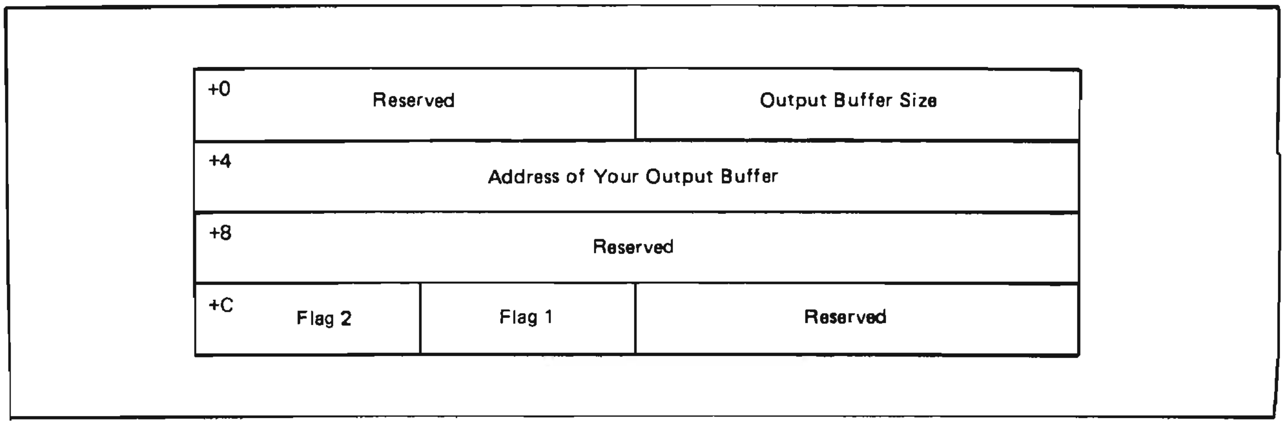


Figure 84. Parameter List Expansion for the List Form of TPG

For Figure 81 - Figure 84, Flag1 is the same as that for Flags in Figure 79 and Figure 80. Flag2 is X'01' for the NOEDIT option and X'02' for the TPG macro.

In Figure 85, Flags is the same as that for Flags in Figure 79 and Figure 80.

If you use the standard, list, or execute form of the TGET macro, the coded parameters expand into the parameter list shown in Figure 85.

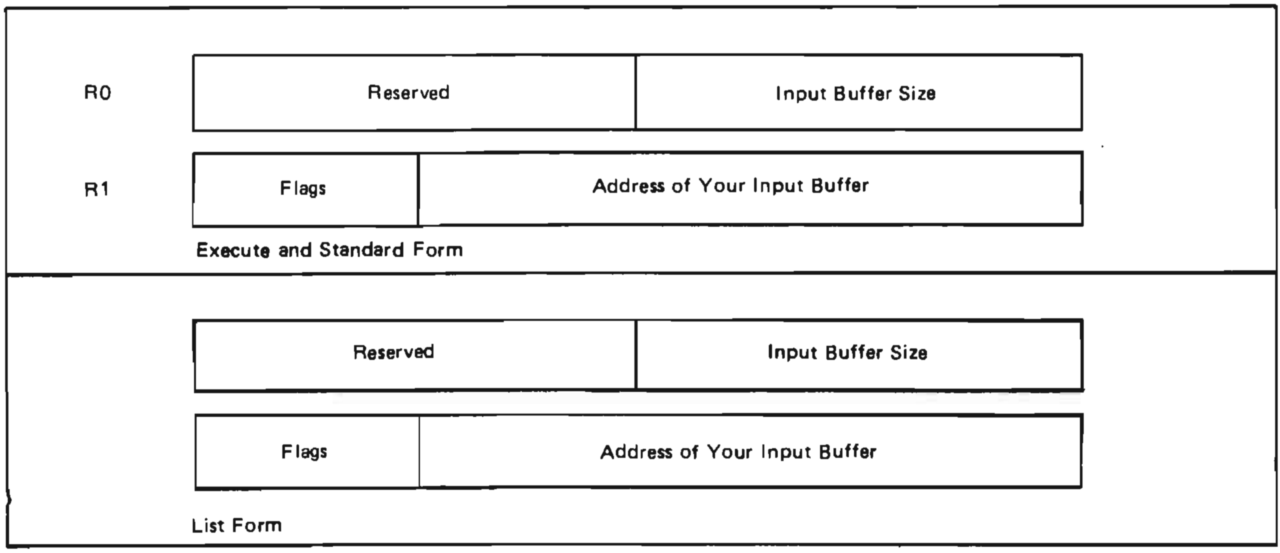

R0
Reserved
Input Buffer Size
R1
Flags
Address of Your Input Buffer
Execute and Standard Form
Reserved
Input Buffer Size
Flags
Address of Your Input Buffer
List Form

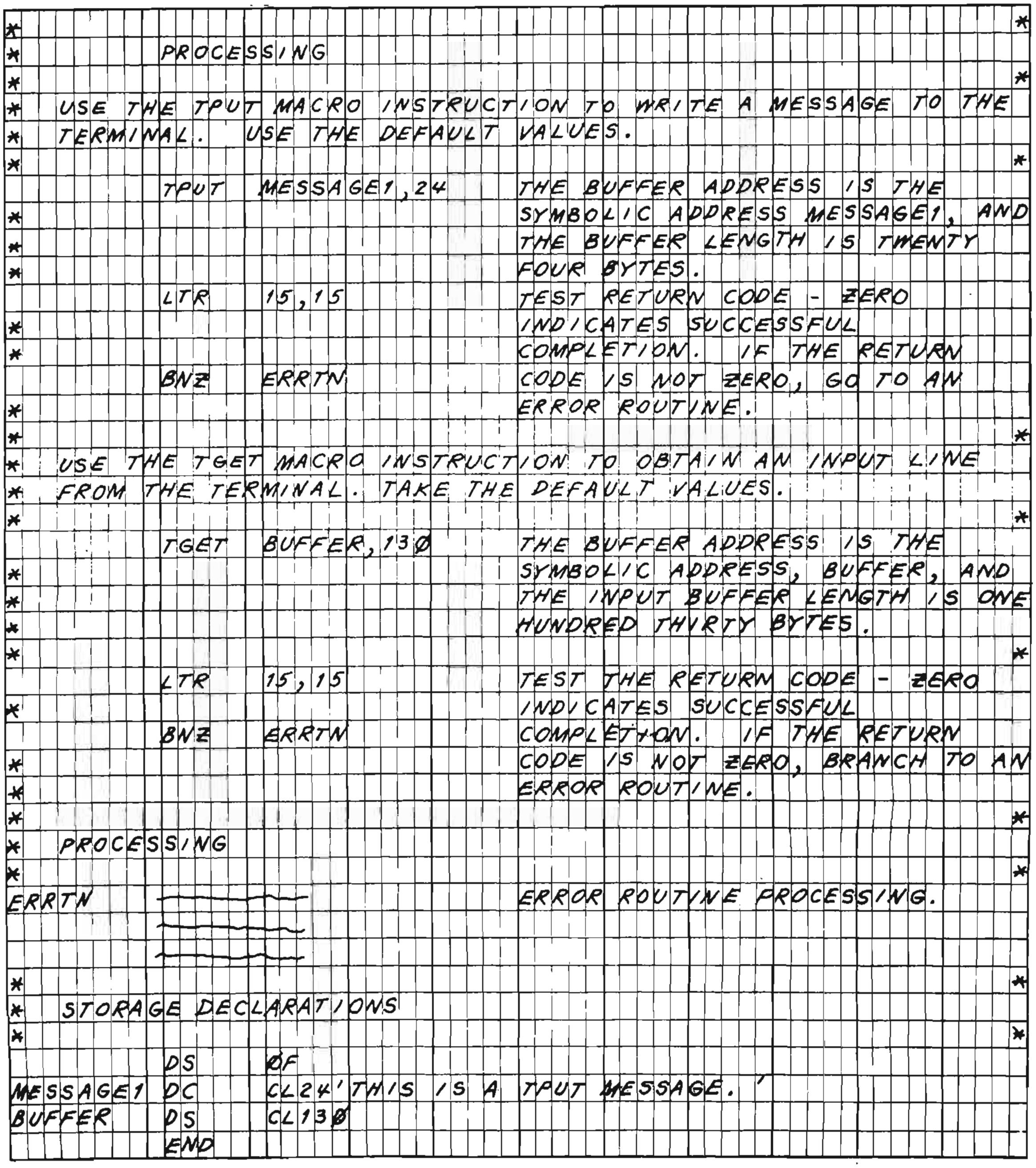
Figure 85. Parameter List Expansion for the Standard, List, and Execute Forms of TGET

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 TPUT and TGET Using the Default Values

Figure 86 shows a TPUT and a TGET macro instruction. They both use 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
```

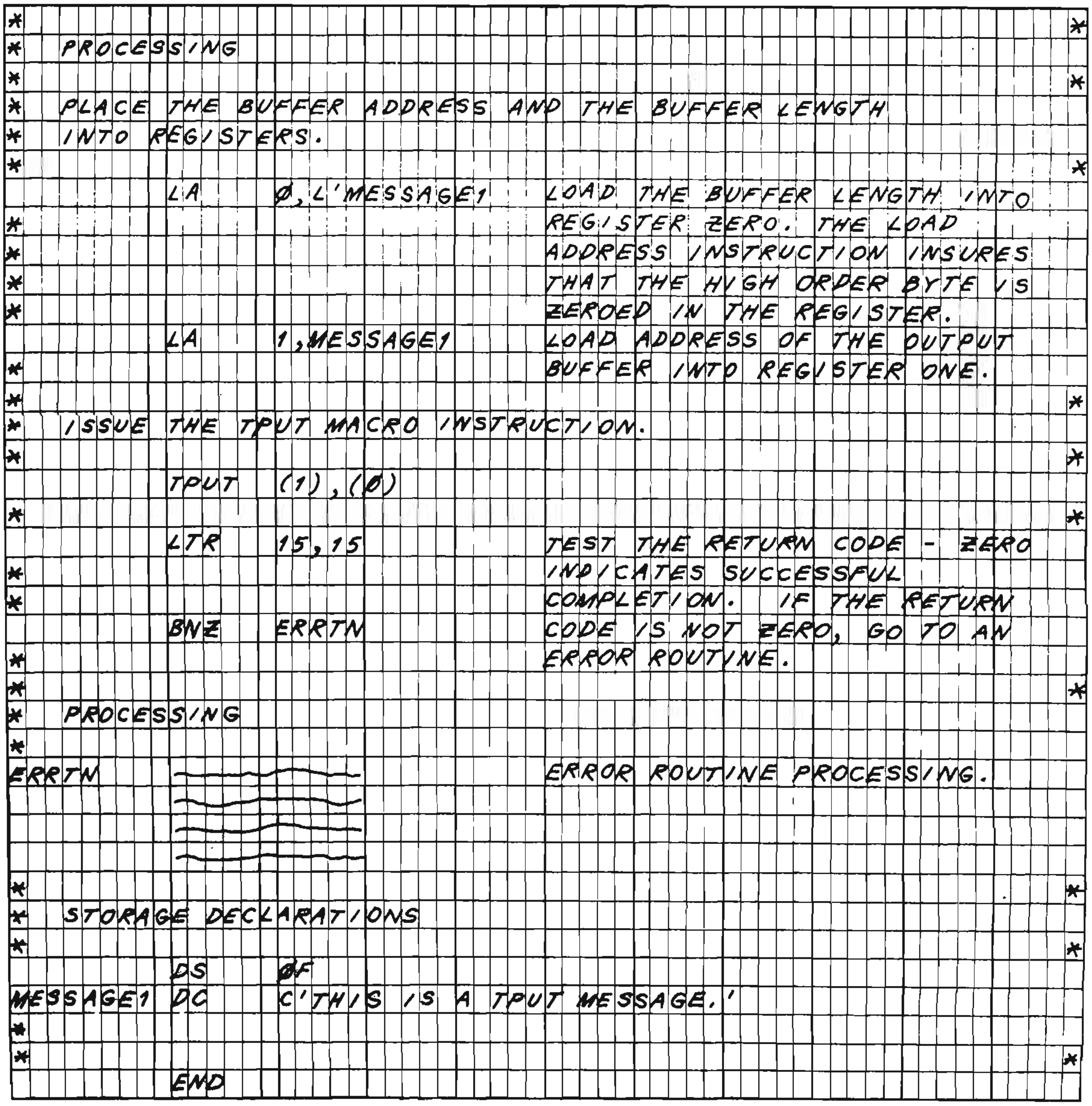
Figure 86. Coding Example: TPUT and TGET Macro Instructions Using the Default Values

The program issuing the TGET macro instruction is not given control until a line of data is returned. The default value is WAIT. If less then 130 characters are entered, the input buffer will be padded with blanks. 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 87 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.

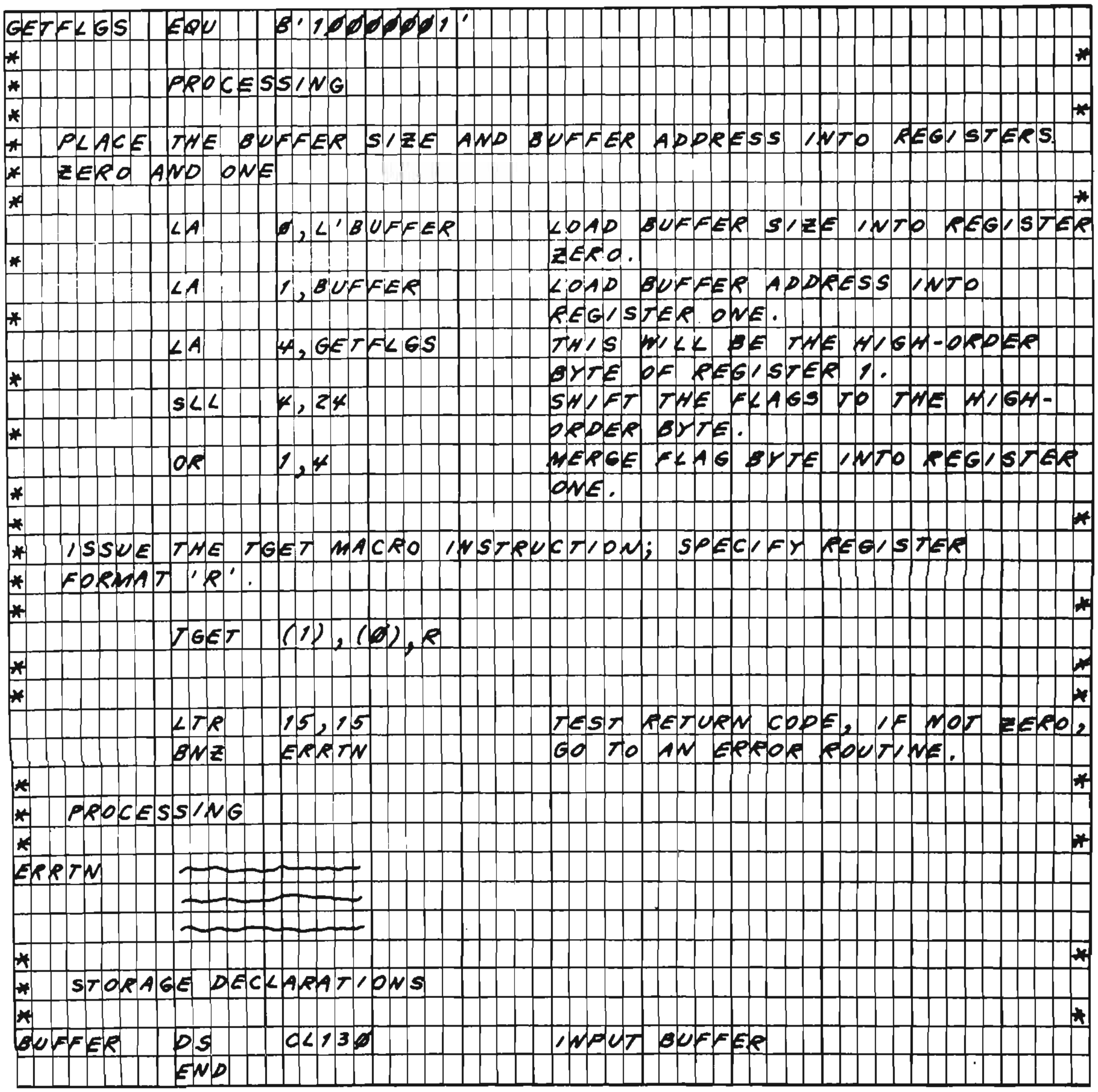


```
*                                                                      *
*  PROCESSING
*                                                                      *
*  PLACE THE BUFFER ADDRESS AND THE BUFFER LENGTH
*  INTO REGISTERS.
*                                                                      *
         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 87. Coding Example: TPUT Macro Instruction Buffer Address and Buffer Length in Registers

Example of the TGET Macro Instruction -- Register Format

Figure 88 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 is set to binary 10000001, indicating that this is a TGET macro instruction requesting ASIS processing. This means that only minimal editing is 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 88. 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.

Macro Instruction	Function
GTSIZE	Get terminal line size
GTTERM	Get terminal attributes
RTAUTOPT	Restart automatic line numbering or character prompting
SPAUTOPT	Stop automatic line numbering or character prompting
STATTN	Set attention simulation
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 interterminal communication
STFSMODE	Set full screen mode
STLINENO	Set line number
STSIZE	Set terminal line size
STTIMEOU	Set timeout feature
STTMPMD	Set terminal display manager options
STTRAN	Set character translation
TCLEARQ	Clear buffers

Some of the terminal control macro instructions may be safely coded in a user-written command processor. They are:

GTSIZE
GTTERM
RTAUTOPT
SPAUTOPT
STAUTOCP
STAUTOLN
STFSMODE
STLINENO
STTMPMD
STSIZE
TCLEARQ

The other macro instructions are intended for system use and 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
STTRAN

Except for the GTSIZE, RTAUTOPT, SPAUTOPT, and STAUTOCP macros, all terminal control macros must be issued in 24-bit addressing mode. Note that all services invoked by the terminal control macros execute in 24-bit addressing mode.

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 89 shows the format of the GTSIZE macro instruction.

[symbol]	GTSIZE

Figure 89. 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 specified to the SVC. No parameter should be specified.

GTTERM -- Get Terminal Attributes

Use the GTTERM macro instruction to determine the primary (default) and alternate screen sizes specified for a 3270 display terminal.

Use the ERASE/WRITE command (X'F5') to erase the screen, to set the screen size mode to primary mode, and optionally to write data to the screen. Use the ERASE/WRITE ALTERNATE command (X'7E') to erase the screen, to set the screen size mode to the alternate mode, and optionally to write data to the screen.

Figure 90 shows the format of the GTTERM macro instruction.

```
[symbol] | GTTERM | PRMSZE=addr[,ALTSZE=addr][,ATTRIB=addr][,MF={L           }]
         |        |                                         [     {(E,ctrl-addr)}]
```

Figure 90. The GTTERM Macro Instruction

PRMSZE=addr
specifies the address of a 2-byte area into which GTTERM returns the primary row value in the high-order byte and the primary column value in the low-order byte.

ALTSZE=addr

specifies the address of a 2-byte area into which GTTERM returns the alternate row value in the high-order byte and the alternate column value in the low-order byte.

ATTRIB=addr

specifies the address of a 1-word field into which GTTERM returns terminal attributes. The contents of this field are described below:

Byte	Bits	Values	Meaning
0-2			Reserved
3	0-5		Reserved
	6	0	The device supports EBCDIC code.
		1	The device supports ASCII code.
	7	0	The Read Partition (Query) is not supported.
		1	The Read Partition (Query) is supported.

Note: If ATTRIB is specified, you do not have to code PRMSZE or ALTSZE.

MF=

Indicates the form of the GTTERM macro instruction.

L

Specifies the list form.

(E,ctrladdr)

Specifies the execute form and the address of the list form.

When control is returned to the user, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Successful.
08	Terminal in use is not a display terminal.
0C	Required PRMSZE parameter was not specified.

If you use the list form of the GTTERM macro, the coded parameters expand into the parameter list shown in Figure 91.

+0	Address of halfword to receive primary screen size
+4	Address of halfword to receive alternate screen size
+8	Address of word to receive Device Query supported flag

Figure 91. Parameter List Expansion for List Form of GTTERM

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 92 shows the format of the RTAUTOPT macro instruction.

[symbol]	RTAUTOPT

Figure 92. 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 specified to the SVC. No parameter should be specified.
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 interruption 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 93 shows the format of the SPAUTOPT macro instruction.

[symbol]	SPAUTOPT

Figure 93. 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 specified to the SVC. No parameter should be specified.
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 with terminals that use TSO through TCAM. It is ignored if TSO is not active when the macro instruction is issued.

Figure 94 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.

```
[symbol]     STATTN     [LINES=   {integer} [,TENS=   {integer}]]
                                  {   0   }           {   0   }
                        [,INPUT=  {address}]
                                  {   0   }
```

Figure 94. 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. The LINES operand applies only to terminals that are not display stations. However, the display user may cause a simulated attention interruption at the bottom of the screen (that is, 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, 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
08	Invalid terminal type. This macro instruction should not be issued for terminals that use TSO/VTAM.

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 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 EDIT.

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 (that is, 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 issued by a batch task.

Figure 95 shows the format of the STAUTOCP macro instruction.

[symbol]	STAUTOCP

Figure 95. 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 specified to the SVC. No parameter should be specified.

STAUTOLN -- Start Automatic Line Numbering

Use the STAUTOLN macro instruction to start automatic line numbering. Automatic line numbering displays 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 (that is, issuing successive TGET macros).

The system displays 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 issued by a batch task.

Figure 96 shows the format of the STAUTOLN macro instruction. Each of the operands is explained following the figure.

[symbol]	STAUTOLN	S=address, I=address

Figure 96. 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 specified - values out of range.

STBREAK -- Set Break

Use the STBREAK macro instruction to indicate whether the transmit interrupt feature on an IBM 1050, 2741, 3270, 3767, or 3770 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 transmit interrupt feature is a special feature on 1050 and 2741 terminals; it is a standard feature on the 3767, 3770, and 3270 display terminals. Specifying STBREAK YES for a 1050 without the transmit interrupt feature could result in loss of output or a permanent error at the terminal.

When the transmit interrupt feature is being used by the system, the terminal user can enter the next line while the previous one is being processed. All 33/35 Teletypes and IBM 3270, 3767, and 3770 terminals are handled this way. 1050s and 2741s that have been defined in the TCAM message control program as having the transmit interrupt feature will be handled this way (unless STBREAK NO is specified).

Note: For 2741s, 3767s, 3770s, TWX, and WTTY devices supported by VTAM, the keyboard will remain unlocked when STBREAK NO is specified.

When the feature is in use, terminal handling of input and output 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 from a 1050 or a 2741 terminal 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. If the application does not issue a TCLEARQ macro to flush the input buffer queue, the interrupted input from a 3767, 3270, or a 3770 terminal is received by the application program but is not printed at the terminal.

When the transmit interrupt feature is not being used by the system, a 1050 or 2741 terminal keyboard is unlocked only after the user's program has issued a TGET request for input. (A 3270, 3767, or 3770 terminal keyboard's normal state is unlocked.) 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 display stations are handled in this way. All 1050s and 2741s which have been defined in the TCAM message control program as not having the transmit interrupt feature will be handled this way.

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 carrier 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 an invalid character 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 was in effect for line deletion prior to the issuance of the STCC macro.

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 to the SVC.
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.
0C	Invalid terminal type.

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 99 shows the format of the STCLEAR macro instruction. Each of the operands is explained following the figure.

```
                                     {address}
   [symbol]      STCLEAR     STRING= {   0   }
```

Figure 99. 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 100 shows the format of the STCOM macro instruction.

[symbol	STCOM	[YES NO]

Figure 100. 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 to the SVC.

STFSMODE -- Set Full Screen Mode

Use the STFSMODE macro instruction under VTAM to specify whether an IBM 3270 display terminal is to operate in full-screen mode. Operating in full-screen mode provides screen protection by preventing the screen from being overlaid by non-full-screen messages, and allowing the terminal user to read non-full-screen messages before they are overlaid by full-screen messages. If full-screen mode is set off, full-screen TPUT requests (that is, TPUT requests that specify the FULLSCR operand) can result in certain problems at the terminal. A message not expected by the terminal user or the command processor, such as a broadcast message or password request, might not be noticed by the terminal user and might be quickly overlaid by a full-screen display. An unexpected message might overlay part of a full-screen display, which could result in invalid input to the command processor.

The STFSMODE macro instruction may be used only in a VTAM time-sharing environment. It is ignored if VTAM is not active when the macro instruction is issued.

See Appendix B for additional information about the use of the STFSMODE macro and the full-screen environment.

Figure 101 shows the format of the STFSMODE macro instruction.

```
[symbol]     STFSMODE    [ON ]   [,INITIAL=YES]    [,RSHWKEY=n]
                         [OFF]   [,INITIAL=NO ]
                                 [,NOEDIT=YES ]
                                 [,NOEDIT=NO  ]
```

Figure 101. The STFSMODE Macro Instruction

ON
indicates that full screen mode is in operation. If neither ON nor OFF is specified, ON is assumed. When a terminal operating in full-screen mode is to receive a non-full-screen message (TPUT without FULLSCR), the display screen is cleared, the alarm is sounded (if the Alarm special feature is installed), and the message is displayed on the screen. If several such messages occur one after the other, the screen is cleared once, the alarm is sounded, and the messages are dispalyed in sequence. When the next full screen TPUT message (TPUT with FULLSCR) is issued by the application, the terminal user will be required to acknowledge the messages on the screen before the TPUT FULLSCR can be displayed. Three asterisks (***) displayed at the current line indicate that acknowledgement is required. To continue, the user must press the ENTER key.

OFF
indicates that full screen mode is not in operation. When a terminal that is not operating in full-screen mode receives a message, the RSHWKEY value is reset to the default, and the message is sent to the terminal according to the options specified in the TPUT macro, possibly overlaying the current screen contents.

INITIAL=YES
indicates that this is the first time during the execution of a command processor that the command processor has entered full screen mode. This operand prevents the first TPUT FULLSCR issued by the command processor from forcing a paging condition when the last transaction at the terminal was input. For example, after a user logs on and the READY message is displayed and the user types in the name of a command processor, a paging condition is not forced if INITIAL=YES was specified. INITIAL=YES is ignored if OFF is specified.

INITIAL=NO

indicates that forced paging is to occur normally whenever a TPUT with FULLSCR follows a TPUT without FULLSCR. If neither INITIAL=YES nor INITIAL=NO is specified, INITIAL=NO is assumed.

NOEDIT=YES

indicates that input from the terminal will be added to the input queue without being modified, regardless of the options specified on the TGET macro instruction.

NOEDIT=NO

indicates that input from the terminal will be handled according to the options specified on the TGET macro instruction before it is added to the input queue. If neither NOEDIT=NO or NOEDIT=YES is specified, NOEDIT=NO is assumed.

RSHWKEY

specifies as a decimal digit the program function (PF) key to be used as the reshow key. If RSHWKEY is not specified, the default value for the PA2 key (X'6E') is used.

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 to the SVC.
08	Invalid terminal type. This macro instruction is valid only for IBM 3270 display terminals that use VTAM.

STLINENO -- Set Line Number

Use the STLINENO macro instruction under VTAM to specify the number of the screen line on an IBM 3270 display terminal on which the next non-full-screen message should appear. (A non-full-screen message results from issuing a TPUT macro instruction without the FULLSCR operand.) The STLINENO macro instruction may also be used to specify whether the 3270 terminal is to operate in full screen mode.

The STLINENO macro instruction may be used only in a TSO/VTAM time-sharing environment. It is ignored if TSO/VTAM is not active when the macro instruction is issued.

See Appendix B for additional information about the use of the STLINENO macro and the full-screen environment.

Figure 102 shows the format of the STLINENO macro instruction.

[symbol]	STLINENO	{LINE=number {LINELOC=address} [,MODE=ON [,<u>MODE=OFF</u>]

Figure 102. The STLINENO Macro Instruction

LINE=

specifies in decimal the line number on which the next non-full-screen message is to appear. The line number must be a value from 1 to n where **n** is the maximum number of lines allowed for the terminal in use. Either the actual line number or a register (2-12, enclosed in parentheses) containing the line number in the low-order byte may be specified.

LINELOC=

specifies the address of a fullword whose low-order byte contains the number of the screen line on which the next non-full-screen message is to appear. The line must number be a value from 1 to n where **n** is the maximum number of lines allowed for the terminal in use. Either an actual address (RX-type) or a register (2-12, enclosed in parentheses) containing the address may be specified.

MODE=

specifies whether full screen mode is to be set ON or OFF. If MODE is not specified, MODE=OFF is assumed.

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 to the SVC.
08	Invalid terminal type. This macro instruction is valid only for IBM 3270 display terminals that use TSO/VTAM.
0C	The line number specified was 0 or it was greater than the maximum number of lines allowed for the terminal in use.

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 103 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 103. 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 unpredicatable 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[2]	72
2260,2265	12x80, 12x40, 6x40, 15x64 - as specified by the installation in the TCAM message control program.
3270	12x40, 12x80, 24x80, 32x80, or 43x80
3767	132
3770	132

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 to the SVC.
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 Time Out Feature

Use the STTIMEOU macro instruction to specify whether the 1050 terminal has the optional text time out suppression feature. The macro instruction allows 1050s, 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.

[2] Trademark of the Teletype Corporation.

A 1050 without the text time out suppression feature operates as follows: When the PROCEED light is on and the keyboard is unlocked, the terminal will time out; that is, the keyboard will lock if the user does not type input for approximately 20 seconds. The system subsequently responds to the time out by restoring the keyboard so that the user may continue. The user can prevent the time out by periodically pressing the SHIFT key.

A 1050 with the text time out 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 time out within 28 seconds, in contrast to the read channel command that does time out. (Note: If the system is directed to use the read inhibit channel command for a 1050 that does time out, 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 TCAM message control program. If the currently connected terminal has the text time out 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 time out 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 TIME OUT 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 104 shows the format of the STTIMEOU macro instruction.

[symbol]	STTIMEOU	[<u>YES</u> NO]

Figure 104. The STTIMEOU Macro Instruction

YES

indicates that IBM 1050 terminal does time out. It does not have the text time out suppression feature. If the operand is omitted, the default is YES.

NO

indicates that the IBM 1050 terminal does not time out. The 1050 does have the text time out 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 to the SVC.
08	Invalid terminal type. This macro instruction applies to the IBM 1050 terminal only.

STTMPMD -- Set Terminal Display Manager Options

Use the STTMPMD macro instruction to specify whether a Display Terminal Manager is active or whether the PA1 and CLEAR key indications are to be passed through to the application program.

The STTMPMD macro instruction is issued only in a time-sharing environment. It is ignored if issued for a non-TSO task. The STTMPMD macro is valid for display terminals operating in both the TCAM and VTAM environments.

Figure 105 shows the format of the STTMPMD instruction. Each of the operands is explained following the figure.

[symbol]	STTMPMD	⎡ON ⎤ ⎣OFF⎦ ⎡,KEYS= ⎧NO ⎫⎤ ⎣ ⎩ALL⎭⎦

Figure 105. The STTMPMD Macro Instruction

ON
indicates that a Display Terminal Manager is in control. If neither ON nor OFF is specified, ON is the default.

OFF
indicates that a Display Terminal Manager is not in control.

KEYS=NO
indicates that the PA1 and CLEAR key indications are not to be returned to the application program. This is the default if the KEYS operand is omitted.

KEYS=ALL
indicates that the PA1 and CLEAR key indications are to be returned to the application program.

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 is not a display terminal.

STTRAN -- Set Character Translation

Use the STTRAN macro instruction to initiate the use of user-specified translation tables, to modify specific character translations in active translation tables, to remove character modifications made to user-specified translation tables, and to terminate the use of user-specified translation tables. Translation tables allow characters entered at the terminal to be interpreted as other characters when they are received by TSO, and characters sent by TSO to be interpreted as other characters when they are received at the terminal.

The TERMINAL command issues this macro instruction when a terminal user specifies the TRAN, NOTRAN, CHAR, or NOCHAR operand of the command.

Translation tables are built and used in pairs: one for input and one for output. Each pair is a control section consisting of a fullword containing the address of the output table, followed by a 256-byte EBCDIC table for translating the inbound characters, followed by a 256-byte EBCDIC table for translating the outbound characters. Each character in an input table must have a counterpart in its companion output table, and the characters must have the same relative position in both tables. Refer to *SPL: TSO* for instructions on building translation tables.

A translation table translates inbound data after the system translates the line code to EBCDIC characters. A translation table translates outbound data before the system translates EBCDIC characters to line code.

The STTRAN macro instruction is used only in a VTAM time-sharing environment. It is ignored if VTAM is not active when the macro instruction is issued.

Figure 106 shows the format of the STTRAN macro instruction. Each of the operands is explained following the figure.

[symbol]	STTRAN	[{ TABLE=address,NAME=address NOTRAN } { TCHAR=address,SCHAR=address NOCHAR,NAME=address }] [MF= { L (E,ctrl addr) }]

Figure 106. The STTRAN Macro Instruction

TABLE=address
specifies the address of a pair of user-written translation tables.

NAME=address

specifies the address of an 8-byte area containing an EBCDIC character string. (The string is left-justified and padded to the right with blanks if it is less than eight characters long.) The character string consists of the name of a member in a load module that contains user-written translation tables.
When NAME is used with NOCHAR, the STTRAN macro instruction causes the command processor to store the member name in the 8-byte area.

NOTRAN

specifies that the use of user-written translation tables be discontinued.

TCHAR=address

specifies the address of a 1-byte area containing the EBCDIC representation of a character as it appears at the terminal.

SCHAR=address

specifies the address of a 1-byte area containing the EBCDIC representation of a character as it appears to the system.

NOCHAR

specifies that current TCHAR and SCHAR values are no longer in effect.

MF=

indicates the form of the STTRAN macro instruction.

L

specifies the list form.

(E,ctrl addr)

specifies the execute form and the address of the list form.

When control is returned to the user, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Successful.
04	NOTRAN or NOCHAR was specified but translation was not in effect.
08	TABLE or NOCHAR was specified but the NAME operand did not specify an address.
0C	Internal error - unidentifiable flag set in input register 0.

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 ensures 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

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.

The following sections discusses:

- Using the command scan service routine
- Using the parse service routine

Using the Command Scan Service Routine (IKJSCAN)

In general, a terminal monitor program links to command scan to verify command names. It may also be invoked by any command processors that process subcommands. It can also be used to scan the reply to a prompt message.

Command scan examines a command in a command buffer and performs the following functions:

1. It translates all lowercase characters in the command name to uppercase.

2. If a valid parameter is present, it resets the offset to the number of text bytes preceding the first non-blank character in the operand field. If a valid operand is not present, the offset equals the length of the text portion 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 checks the syntax of the command name.

5. It recognizes an implicit EXEC command that has a percent sign as the first character.

6. It has logic to handle leading blanks and embedded comments.

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 invoking the command scan service routine via the CALLTSSR macro, you must create the parameter structure shown in Figure 108. You then place the address of the command scan parameter list (CSPL) into general register 1, set the flags in the flag word, and issue CALLTSSR specifying IKJSCAN, the command scan service routine.

IKJSCAN must receive control in 24-bit addressing mode. All input passed to IKJSCAN must reside below the 16-megabyte virtual storage line.

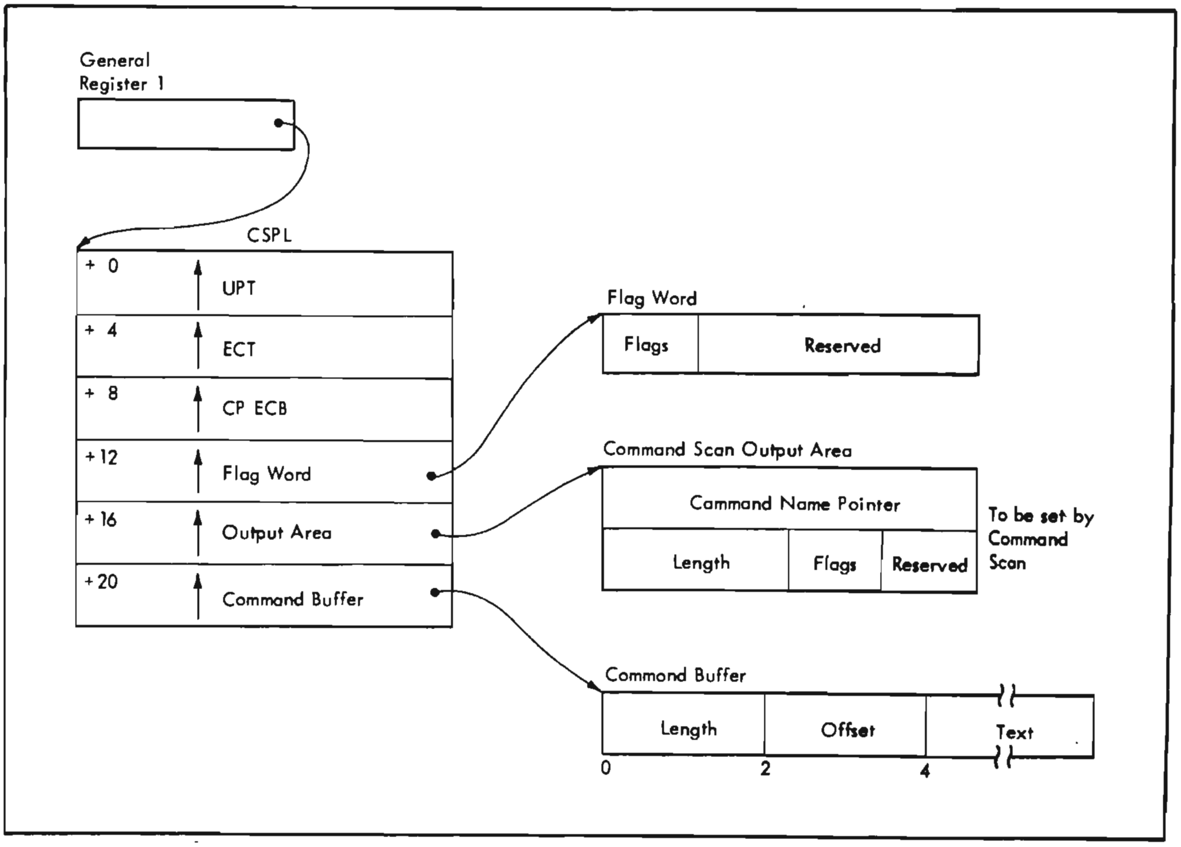


Figure 108. The Parameter List Structure Passed to Command Scan

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.

CHARACTER		CHARACTER TYPE						
		Separator	National	Alphabetic	Numeric	Command Delimiter	Delimiter	Special
Comment	/*	x						
Horizontal Tab	HT	x				x		
Blank	b̸	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
Less than	<							x
Greater than	>							x
Plus sign	+							x
Logical OR	|							x
Exclamation point	!							x
Logical NOT	¬							x
Percent sign	%							x
Dash	-							x
Question mark	?							x
Colon	:							x
Quotation Mark	"							x

Figure 111. 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 in the command scan output area, and by updating the offset field in the command buffer. Figure 112 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.
X'04'	The command is an implicit EXEC command.	Length of command name.	The first non-separator following the command name.

Figure 112. 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:

Hexadecimal Code	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 invokes the parse service routine via the CALLTSSR or LINK macro, passing it a parse parameter list (PPL) which contains the address of the PCL. The parse service routine scans and checks each operand against the entries (called PCEs: parameter control entries) in the PCL. In turn, the parse service routine builds and returns results of the scan to the command processor in a parameter descriptor list (PDL), whose 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.

IKJPARS must receive control in 24-bit addressing mode. All input passed to IKJPARS must reside below the 16-megabyte virtual storage line.

Figure 113 depicts a command processor's use of the parse macro instructions, the parse service routine, and the IKJPARMD DSECT.

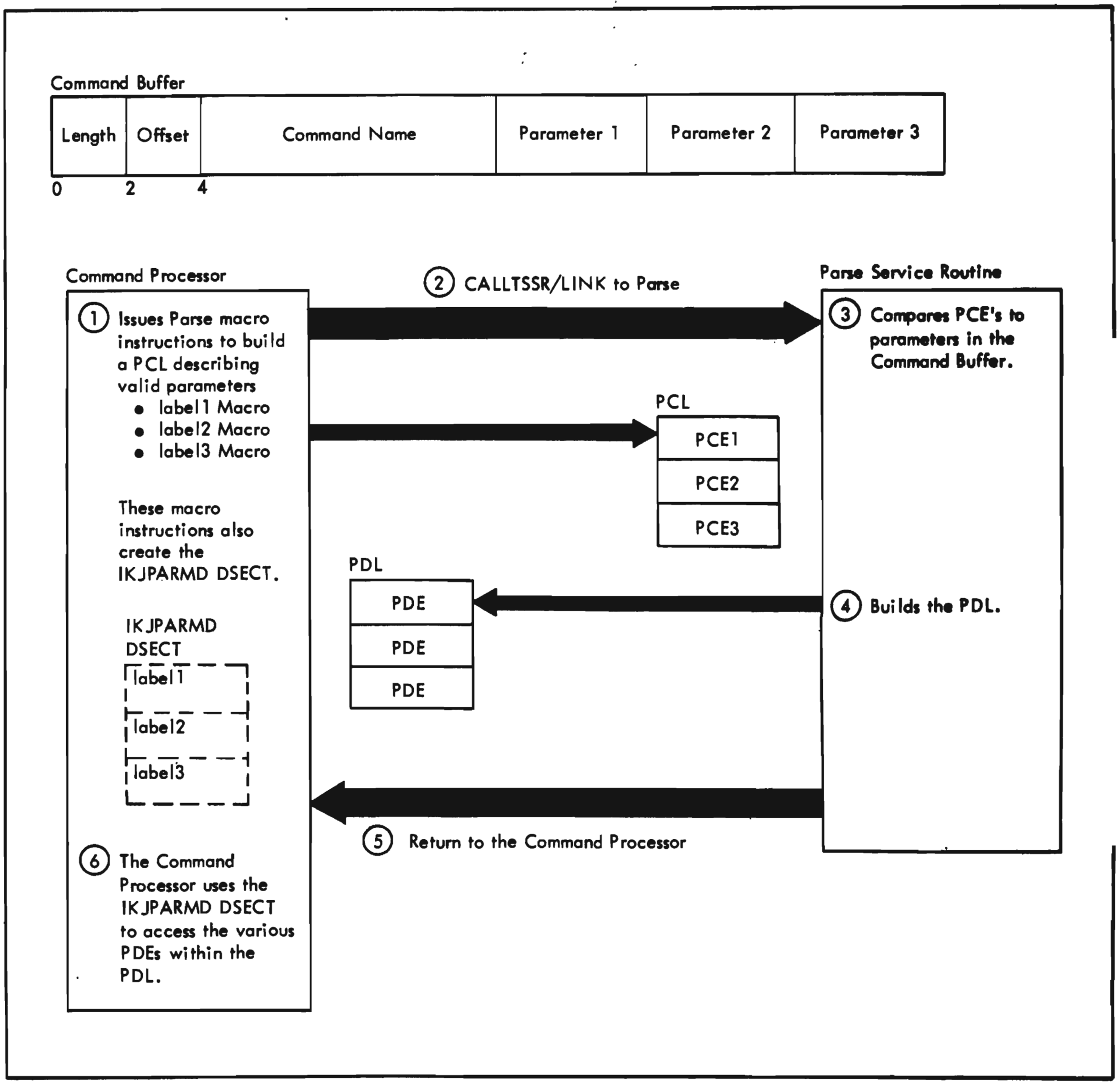

Command Buffer
Length
Offset
Command Name
Parameter 1
Parameter 2
Parameter 3
0
2
4
Command Processor
1 Issues Parse macro instructions to build a PCL describing valid parameters
• label1 Macro
• label2 Macro
• label3 Macro
These macro instructions also create the IKJPARMD DSECT.
IKJPARMD DSECT
label1
label2
label3
6 The Command Processor uses the IKJPARMD DSECT to access the various PDEs within the PDL.
2 CALLTSSR/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 113. A Command Processor Using the Parse Service Routine

The parse service routine support consists of the following:

1. The following set of macro instructions:

   IKJPPL builds an IKJPPL DSECT which maps the parse parameter list.

   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, IKJIDENT, 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 a reserved 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 virtual storage (allocated by the parse service routine) that remains after the parse service routine 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.

The parse service routine also provides the following services which may be selected by the calling routine:

- It translates the command operands to uppercase.
- 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 the parse service routine
- Additional facilities provided by the parse service routine
- 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, tab, comma, or a comment (see Figure 111). The command parameters end either at the end of a logical line (carrier 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, the parse service routine ignores that portion of the line that follows the semicolon. The parse service routine returns no message to indicate this condition.

There are two types of command parameters recognized by the parse service routine:

1. Positional parameters
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 the parse service routine finds an alphabetic character in that position, the numeric parameter is considered missing. The parse service routine 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 following delimiter-dependent parameter syntaxes shown in Figure 114.

Parameter	Macro Instruction Used to Describe Parameter
DELIMITER STRING VALUE ADDRESS PSTRING USERID UID2PSWD DSNAME DSTHING QSTRING SPACE JOBNAME	IKJPOSIT
CONSTANT VARIABLE STATEMENT NUMBER	IKJTERM
EXPRESSION	IKJOPER
RESERVED WORD	IKJRSVWD
HEX CHAR INTEG	IKJIDENT

Figure 114. Delimiter-Dependent Parameters

DELIMITER - It may be any character other than an asterisk, left parenthesis, right parenthesis, semicolon, blank, comma, tab, carrier return, or digit. A self-defining delimiter character is represented in this discussion by the symbol #. The delimiter parameter is used only in conjunction with the string parameter.

STRING - A string is the group of characters between two alike self-defining delimiter characters, such as

#string#

or, the group of characters between a self-defining delimiter character and the end of a logical line, such as

#string

The same self-defining delimiter character can be used to delimit two contiguous strings, such as

#string#string#

or

#string#string

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.

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, the parse service routine assumes that the string ends at the end of the logical line. If the parse service routine encounters 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 uppercase. 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.

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, or, in extended mode, from one to eight hexadecimal digits followed by a period. An extended absolute address must not exceed the address represented by the hexadecimal value 7FFFFFFF.

(For more information on extended addressing, see the description of the EXTENDED operand in "IKJPOSIT - Describing a Delimiter-Dependent Positional Parameter" below.)

Relative address - A relative address consists of from one to six hexadecimal digits preceded by a plus sign, or, in extended mode, from one to eight 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 uppercase or lowercase.

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 uppercase or lowercase.

Symbolic address - A symbolic address consists of any combination, up to 32 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 one of the following formats:

```
1.  modulename.entryname.relative-address
2.  modulename.entryname
3.  modulename.entryname.symbolic-address
4.  .entryname.symbolic-address
5.  .entryname.relative-address
6.  .entryname
```

- modulename - any combination of one to eight alphameric characters, of which the first is an alphabetic or national character
- entryname - same syntax as a modulename, and always preceded by a period
- symbolic address - syntax as defined above, and always preceded by a period
- relative address - syntax as defined above, and always preceded by a period

You may qualify symbolic or relative addresses to indicate that they apply to a particular module and CSECT as in formats 1-3. However, if the address applies to the currently active module, you do not have to specify *modulename* as in formats 4-6.

Indirect address - An indirect address is an absolute, relative, symbolic, or general register address followed by from one to 255 indirection symbols (percent signs), such as

+A%

The number of percent signs following the address indicates the number of levels of indirect addressing. In this example (+A%), the data is at the location pointed to by +A. See Figure 115.

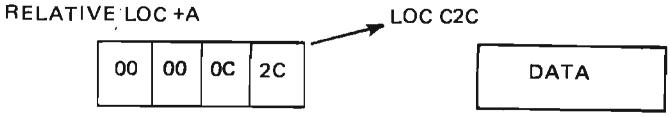


Figure 115. Example of an Indirect Address

Address expression - An address expression has the following format:

```
address{±}expression value [%...][{±}expression value [%...]]...
```

- address - can be an absolute, symbolic, indirect, relative, or general register address. If a general register is specified, it must be followed by at least one indirection symbol.
- expression value - a plus or minus displacement from an address in storage, consisting of from one to six decimal or hexadecimal digits.
  - When you specify the EXTENDED keyword of IKJPOSIT to indicate extended mode, the terminal user may specify a one to ten digit decimal number, or a one to eight digit hexadecimal number.
  - Decimal displacement is indicated by an "N" or "n" following the offset. The absence of an "N" or "n" indicates hexadecimal displacement.
  - There is no limit to the number of expression values in an address expression.
- Each expression value may be followed by from one to 255 percent signs, one for each level of indirect addressing.

For example, *addr1*+124n, an address expression in decimal format, indicates a location 124 decimal bytes beyond *addr1*. Another example, *addr2*-AC, is an address expression in hexadecimal format and indicates a location 172 decimal bytes before *addr2*.

The processing of an address expression, 12R%%+4N%, involving indirect addressing, is shown in Figure 116. The address in the expression is a general register address with two levels of indirect addressing. The result of the processing of this part of the address expression is location 1D0. The expression value indicates a displacement of four bytes beyond location 1D0 with one level of indirect addressing. The data, then, is at location 474.

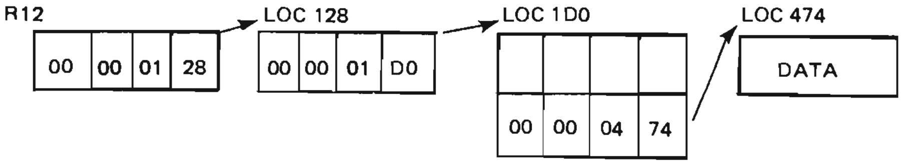


Figure 116. Example of an Address Expression with Indirect Addressing

Note: Blanks are not allowed within any form of the address parameter.

PSTRING - A parenthesized string is a string of characters enclosed within a set of parentheses, such as:

(string)

The string may consist 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.

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. If delimiters are used, the password must be enclosed in quotes. If quotes are to be used in the password, two quotes must be entered consecutively. One of them will be eliminated by the parse service routine.

Separators may be inserted between the identification and the slash, and between the slash and the password.

If just the identification is entered, the parse service routine does not prompt for a password. If the identification is entered followed by a slash and no password, the parse service routine prompts for a password by executing a PUTGET macro instruction specifying bypass mode. 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, the parse service routine builds the PDE and leaves the respective password field zero.

UID2PSWD – A userid consists of an identification optionally followed by two passwords. The delimiter between the three values is a slash. The format is:

identification[/password1[/password2]]

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.

password1 – can be any combination of alphameric characters up to eight characters in length. If delimiters are used, the password must be enclosed in quotes. If quotes are to be used in the password, two quotes must be entered consecutively. One of them will be eliminated by the parse service routine.

password2 – Same as password1.

IKJPOSIT generates a variable length parameter control entry (PCE). Within the PCE, a field contains a hexadecimal number indicating the type of positional operand described by the PCE. For UID2PSWD, the hexadecimal number is C.

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: The parse service routine 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 VOLSER option is specified, the first character may be numeric.

If it is numeric, only six characters are accepted for VOLSER. VOLSER is valid only for DSNAME or DSTHING. If USID is specified, the parse service routine will prefix all data set names not entered in quotes with the user identification (from the UPT).

If the slash and the password are not entered, the parse service routine does not prompt for the password. If the slash is entered and not the password, the parse service routine 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.

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. The parse service routine 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.

Note: If the first character of a dsname is an asterisk, the parse service routine will not prefix the USERID.

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 apostrophes within the string, two successive apostrophes must be entered for each single apostrophe desired; one of the apostrophes is removed by the parse service routine.

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.

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.

JOBNAME - The jobname may have an optional job identifier. Each job identifier is a maximum of eight alphameric characters of which the first is alphabetic or national. There is no separator character between the jobname and job identifier. The syntax is jobname (jobid).

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, entered 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 service 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 carrier return, however, the first parameter must be alphabetic.

VARIABLE - The following is the form of the variable parameter.

```
[program-id.]data-name [{OF} qualification]
                       [{IN}              ]
                       [ (subscript)      ]
```

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), entered as:

Data-name - consists of a maximum of 30 characters of the set:

A through Z (alphabetic)

0 through 9 (numeric)

- (hyphen)

typically entered as:

mydataset-123

The data-name cannot begin or end with a hyphen and must contain at least one alphabetic character.

here55.mydataset-123

Qualification - Is applied by placing after a data-name one or more data-names preceded by the qualifiers IN or OF, entered as:

mydataset-123 of yourdataset-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 entered as:

yourdataset-456 (mydataset-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, entered as:

here55 (abc def h15)

A separator character between subscripts is required.

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 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.

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.

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, any one 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, any one 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.

The HEX, CHAR, and INTEG operands on the IKJIDENT macro describe delimiter positional parameters.

- HEX - indicates that any quantity of the form X'nn', 'ABC' (quoted string), or any nonquoted character string in which case a separator or delimiter indicates the end, will be accepted as valid data.
- CHAR - indicates that any data in the form of a quoted or nonquoted string will be accepted as valid data.
- INTEG - indicates that any numeric character in the following form will be converted by the parse service routine to its appropriate binary value.

  (X'nn')---where n is a valid hexadecimal digit(A-F,0-9), maximum of 8.

  (B'mm')---where m is a valid binary bit (0-1), maximum of 32.

  dddddd---decimal digits (0-9), maximum of 10.

Note: The maximum decimal value for INTEG is 2147843647.

Positional Parameters Not Dependent on Delimiters: A positional parameter that is not dependent on delimiters is passed 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 carrier return. A right parenthesis must, however, be balanced by a left parenthesis.

NONATABC - Indicates an alphabetic character only is accepted. (No national characters.)

NONATNUM - Indicates numbers and alphabetic characters are accepted.(No national characters.)

An asterisk can be entered in place of any positional parameter that is not dependent on delimiters.

Entering Positional Parameters as Lists of 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
- UID2PSWD
- DSNAME
- DSTHING
- JOBNAME
- CONSTANT
- STATEMENT NUMBER
- VARIABLE
- HEX
- CHAR
- INTEG
- Any positional parameters 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:

- HEX (form X' ' only)
- ADDRESS
- VALUE
- CONSTANT

- STATEMENT NUMBER
- VARIABLE
- INTEG
- 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, 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 consist 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 IKJKEYD, 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 keyword1:

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, the parse service routine 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.

The parse macros that generate input to parse must be issued by a program that is loaded below 16 megabytes so that IKJPARS can access the PCL. The IKJRLSA macro must be issued in 24-bit addressing mode. There are eleven parse macro instructions:

- IKJPARM
- IKJPOSIT
- IKJTERM
- IKJOPER
- IKJRSVWD
- IKJIDENT
- IKJKEYWD
- IKJNAME
- IKJSUBF
- IKJENDP
- IKJRLSA

These macro instructions perform the following functions:

1. The IKJPARM macro instruction 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 virtual 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 the 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 117 shows the format of the IKJPARM macro instruction. Each of the operands is explained following the figure. Appendix A describes the notation used to define macro instructions.

```
                                  {dsect name}
   label  |  IKJPARM  |    DSECT= {IKJPARMD  }
```

Figure 117. 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 118. This PCE begins the parameter control list.

Number of Bytes	Field	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 118. The Parameter Control Entry Built by IKJPARM

IKJPOSIT - Describing a Delimiter-Dependent Positional Parameter

Code the IKJPOSIT macro instruction to describe most of the delimiter-dependent positional parameters. IKJIDENT is used to describe the others.

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 119 shows the format of the IKJPOSIT macro instruction. Each of the operands is explained following the figure. Appendix A describes the notation used to define macro instructions.

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 is 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.

The Parameter Control Entry Built by IKJPOSIT: The IKJPOSIT macro instruction generates the variable length parameter control entry (PCE) shown in Figure 120.

Number of Bytes	Field	Contents or Meaning
2		Flags. These flags are set to indicate which options were specified in the IKJPOSIT macro instruction.
	Byte 1	
	001.	This is an IKJPOSIT PCE.
	...1	PROMPT
	.... 1...	DEFAULT
	.... .1..	This is an extended format PCE. If the VALIDCK parameter was specified, the length of the field containing the address of the validity checking routine is four bytes.
	.... ..1.	HELP
	1	VALIDCK
	Byte 2	
	1...	LIST
	.1..	ASIS
	..1.	RANGE
	.... 1...	SQSTRING
	.... .1..	USID
	.... ..1.	VOLSER
	1	DDNAME
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:
	HEX	
	1	DELIMITER
	2	STRING
	3	VALUE
	4	ADDRESS
	5	PSTRING
	6	USERID
	7	DSNAME
	8	DSTHING
	9	QSTRING
	A	SPACE
	B	JOBNAME
	C	UID2PSWD
	D	EXTENDED ADDRESS
	E to FF	Not used.
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 was specified on the IKJPOSIT macro instruction.

Figure 120. The Parameter Control Entry Built by IKJPOSIT (Part 1 of 2)

Number of Bytes	Field	Contents or Meaning
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 or 4		This field contains the address of a validity checking routine if VALIDCK was specified on the IKJPOSIT macro. If the "extended format PCE" bit is on in the IKJPOSIT PCE, the address is four bytes long; if the bit is off, the address is three bytes long. This field is not present if VALIDCK was not specified.

Figure 120. 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 121 shows the format of the IKJTERM macro instruction. Each of the operands is explained following the figure. Appendix A 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 121. The IKJTERM Macro Instruction

label
This name is used to address the PCE built by the IKJTERM macro. The hexadecimal offset to the parameter descriptor entry described by this IKJTERM macro instruction is contained in the PCE.

Note: The hexadecimal offset to the PDE will contain binary zero when the IKJTERM macro 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:

```
commandname (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:

```
commandname 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 "Delimiter-Dependent Parameters."

SBSCRIPT[=label-PCE]

Specifies one of two conditions:

1. If SBSCRIPT 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 SBSCRIPT 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, the parse service routine 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, the parse service routine 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 service 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 *IKJ*NAME 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 122.

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
	.... .1..	This is an extended format PCE. If the VALIDCK parameter was specified, the length of the field containing the address of the validity checking routine is four bytes.
	.... ..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.
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.

Figure 122. The Parameter Control Entry Built by IKJTERM (Part 1 of 2)

Number of Bytes	Field	Contents or Meaning
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 or 4		This field contains the address of a validity checking routine if VALIDCK was specified on the IKJTERM macro. If the "extended format PCE" bit is on in the IKJTERM PCE, the address is four bytes long; if the bit is off, the address is three bytes long. This field is not present if VALIDCK was not specified.

Figure 122. 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)
```

typically entered 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 123 shows the format of the IKJOPER macro instruction. Each of the operands is explained following the figure. Appendix A 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 123. The IKJOPER Macro Instruction

label

This name is used to address the PCE built by the IKJOPER macro. The hexadecimal offset to the parameter descriptor entry described by this macro is contained in the PCE.

'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, the parse service routine 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, the parse service routine 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 service 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. The parse service routine 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 describe 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 service 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 124.

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
	.... .1..	This is an extended format PCE. If the VALIDCK parameter is specified, the length of the field containing the address of the validity checking routine is four bytes.
	.... ..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 service 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.

Figure 124. The Parameter Control Entry Built by IKJOPER (Part 1 of 2)

Number of Bytes	Field	Contents or Meaning
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.
3 or 4		This field contains the address of a validity checking routine if VALIDCK was specified on the IKJOPER macro. If the "extended format PCE" bit is on in the IKJOPER PCE, the address is four bytes long; if the bit is off, the address is three bytes long. This field is not present if VALIDCK was not specified.

Figure 124. 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 125 shows the format of the IKJRSVWD macro instruction. Each of the operands is explained following the figure. Appendix A describes the notation used to define macro instructions.

```
label | IKJRSVWD |     'parameter-type'    [,PROMPT='prompt data'   ]
      |          |                         [,DEFAULT='default value']
      |          |
      |          |  [,HELP=('help data','help data',...)]
```

Figure 125. The IKJRSVWD Macro Instruction

label

This name is used to address the PCE built by the IKJRSVWD macro. The hexadecimal offset to the parameter descriptor entry described by this macro is contained in the PCE.

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.

The parse service routine 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 126.

Number of Bytes	Field	Contents or Meaning
2	Byte 1	Flags. These flags are set to indicate options on the IKJRSVWD macro instruction.
	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 service routine. Note: The following fields are omitted if this PCE is used with the IKJTERM macro to describe a figurative constant.
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.
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.

Figure 126. The Parameter Control Entry Built by IKJRSVWD (Part 1 of 2)

Number of Bytes	Field	Contents or Meaning
2		Contains the length (including this field) of ali 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 126. 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 "Positional Parameters Not Dependent on Delimiters."

These positional parameters must be in the form of a character string, with restrictions on the beginning character, additional characters, and length, decimal integers, or hexadecimal characters.

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 127 shows the format of the IKJIDENT macro instruction. Each of the operands is explained following the figure. Appendix A 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' ┘
                    ┌,CHAR  ┐
                    │,INTEG │
                    └,HEX   ┘
                    [,VALIDCK=symbolic-address]
                    [,HELP=('help data', 'help data',...)]
```

Figure 127. 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:

```
commandname (parameter,parameter,...)
```

RANGE

This positional parameter may be entered by the terminal user as a range, that is, in the form:

```
commandname 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.

Note: ASTERISK and INTEGER are mutually exclusive.

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. This is true unless HEX, INTEG, or CHAR is specified. FIRST= and OTHER= serve no purpose in these cases.

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, the parse service routine 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, the parse service routine 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.

CHAR

Specifies that the parse service routine is to accept a string of characters as input. This input string may be either quoted or unquoted.

INTEG

Specifies that the parse service routine is to accept a numeric quantity as input. This quantity may be decimal, hexadecimal, or binary. The number is stored internally as a fullword binary value, regardless of how INTEG was specified.

Note: A maximum length is automatically implied if the INTEG option is specified. For binary input, the maximum number of characters is 32. For hexadecimal input, the maximum length is 8. For decimal input, the maximum length is 10.

HEX

Specifies that the parse service routine is to accept a hexadecimal value as input. This string quantity may be hexadecimal or a quoted or non-quoted string.

Note: All input entered in the form X'n...' must be valid hexadecimal digits (0-9, A-F). All input entered in the form B'n...' must be valid binary digits (0,1). All input entered as unquoted decimals must be valid decimal digits (0-9).

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.
The parse service routine 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 128.

Number of Bytes	Field	Contents or Meaning
2		Flags. These flags are set to indicate which options were specified in the IKJIDENT macro instruction.
	Byte 1	
	100.	This is an IKJIDENT PCE.
	...1	PROMPT
	.... 1...	DEFAULT
	.... .1..	This is an extended format PCE. If the VALIDCK parameter is specified, the length of the field containing the address of the validity checking routine is four bytes.
	.... ..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
	...1	Integer
	.... 1...	Character
	.... .1..	Hexadecimal
	00	Reserved
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

Figure 128. 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 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 maximum 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 defaults 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.
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.

Figure 128. The Parameter Control Entry Built by IKJIDENT (Part 2 of 3)

Number of Bytes	Field	Contents or Meaning
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 or 4		This field contains the address of a validity checking routine if VALIDCK was specified on the IKJIDENT macro. If the "extended format PCE" bit is on in the IKJIDENT PCE, the address is four bytes long; if the bit is off, the address is three bytes long. This field is not present if VALIDCK was not specified.

Figure 128. 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 129 shows the format of the IKJKEYWD macro instruction. Each of the operands is explained following the figure. Appendix A describes the notation used to define macro instructions.

label	IKJKEYWD	[DEFAULT='default-value']

Figure 129. 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 130.

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 IKJKEYWD PCE.
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 130. The Parameter Control Entry Built by IKJKEYWD

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 131 shows the format of the IKJNAME macro instruction. Each of the operands is explained following the figure. Appendix A describes the notation used to define macro instructions.

```
     IKJNAME 'keyword-name'[,SUBFLD=subfield-name]
             [,INSERT='keyword-string']
             [ALIAS=('name','name',...)]
```

Figure 131. 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.

ALIAS=('name','name',...)
Specifies up to 32 alias names for a keyword. Each name represents a valid abbreviation or alternate name and must be enclosed in quotes. All abbreviations or names must be enclosed in a single set of parentheses.

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 132 shows the format of the IKJNAME macro instruction. Each of the operands is explained following the figure. Appendix A describes the notation used to define macro instructions.

```
     IKJNAME 'reserved-word name'
```

Figure 132. The IKJNAME Macro Instruction (when used with the IKJRSVWD Macro Instruction)

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 133.

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
	.... ..1.	ALIAS
	.... 00.0	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 names 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.
1		The total number of aliases.
1		The length minus one of first alias.
Variable		The first alias.
1		The length minus one of second alias.
Variable		The second alias.

Figure 133. 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 134 shows the format of the IKJSUBF macro instruction. Appendix A describes the notation used to define macro instructions.

label	IKJSUBF

Figure 134. 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 135.

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 135. 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 136 shows the format of the IKJENDP macro instruction. Appendix A describes the notation used to define macro instructions.

```
     IKJENDP
```

Figure 136. The IKJENDP Macro Instruction

The Parameter Control Entry Built by IKJENDP: The IKJENDP macro instruction generates the parameter control entry (PCE) shown in Figure 137. 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 137. The Parameter Control Entry Built by IKJENDP

IKJRLSA - Releasing Virtual Storage Allocated by Parse

Execute the IKJRLSA macro instruction to release virtual storage *allocated* by the parse service routine and not previously released by the parse service routine. This virtual 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 the parse service routine. In that case, this macro instruction need not be issued, but will not cause an error if it is issued.

Figure 138 shows the format of the IKJRLSA macro instruction. Each of the operands is explained following the figure. Appendix A describes the notation used to define macro instructions.

label	IKJRLSA	Address of the answer place (1-12)

Figure 138. The IKJRLSA Macro Instruction

address of the answer place
The address of the word in which the parse service routine 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 "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 CALLTSSR macro instruction specifying IKJPARS as the entry point. IKJPARS must receive control in 24-bit addressing mode. See "Passing Control to the TSO Service Routines" for a description of a restriction on using the CALLTSSR macro to invoke parse. Before you invoke 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 the parse service routine returns control to the calling routine. Figure 139 shows this flow of control between a command processor and the parse service routine.

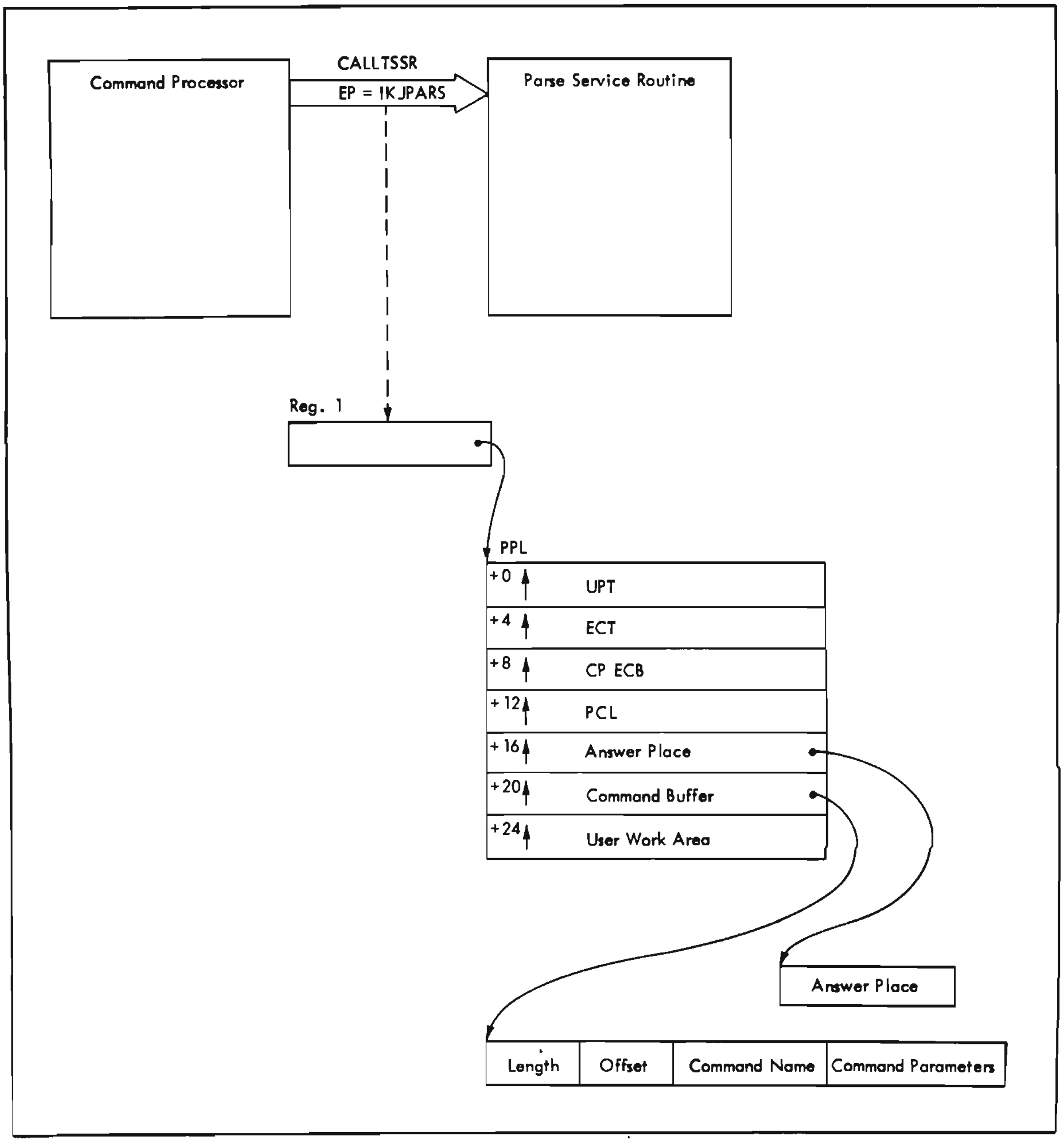


Figure 139. Control Flow Between Command Processor and the Parse Service Routine

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 140 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 virtual 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 X'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 140. The Parse Parameter List

Formats of the PDEs Returned by the Parse Service Routine

The parse service routine 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 the 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 virtual storage	
+4 Subpool number	+5 Length

Pointer to the next block of virtual storage:
The parse service routine gets virtual 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 X'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: The parse service routine 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: The parse service routine 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: The parse service routine uses the IKJPOSIT macro to build a six-word PDE to describe a DSNAME or a DSTHING parameter.

The PDE has the following format:

<table>
<tr><td colspan="3">+0
A pointer to the dsname</td></tr>
<tr><td>+4
Length1</td><td>+6
Flags1</td><td>+7
Reserved</td></tr>
<tr><td colspan="3">+8
A pointer to the member name</td></tr>
<tr><td>+12
Length2</td><td>+14
Flags2</td><td>+15
Reserved</td></tr>
<tr><td colspan="3">+16
A pointer to the password</td></tr>
<tr><td>+20
Length3</td><td>+22
Flags3</td><td>+23
Reserved</td></tr>
</table>

Pointer to the dsname:

Contains a pointer to the first character of the data set name. Contains zero if the data set name was omitted. Contains a pointer to the USID if it is prefixed.

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. When the USID is prefixed, this field will contain the total length of the data set name and the USID.

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.

JOBNAME: The parse service routine uses the IKJPOSIT macro to build a four word PDE to describe a JOBNAME parameter. The PDE has the following format:

+0 A pointer to the jobname		
+4 Length1	+6 Flags1	+7 Reserved
+8 The pointer to the jobid name		
+12 Length2	+14 Flags2	+15 Reserved

Pointer to the jobname:
Contains a pointer to the beginning of the job*name. Cont*ains zero if the jobname was omitted.

Length1:
Contains the length of the jobname. The jobname may not be entered in quotes.

Flags1:

0...	The jobname is not present.
1...	The jobname is present.

Pointer to the jobid:
Contains a pointer to the beginning of the jobid. Contains zero if the jobid was omitted.

Length2:
Contains the length of the jobid. This length figure does not include the parentheses around the jobid.

Flags2:

0...	The jobid is not present.
1...	The jobid is present.
.xxx xxxx	Reserved bits.

ADDRESS: The parse service routine 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 routine</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 entry name, 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.
0000 0100	04	Non-qualified entry name (optionally preceded by a load name).

Sign:

Contains the arithmetic sign character used before the following 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 X'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 the parse service routine to describe the address parameter. The last expression value PDE is indicated by X'FF000000' in its fourth word, the forward chaining field.

The parse service routine 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 X'FF000000' if this is the last expression value PDE.

USERID: The parse service routine uses the IKJPOSIT macro to build a four-word PDE to describe a USERID parameter; it has the following format:

<table>
<tr><td colspan="3">+0
A pointer to the userid</td></tr>
<tr><td>+4
Length1</td><td>+6
Flags1</td><td>+7
Reserved</td></tr>
<tr><td colspan="3">+8
A pointer to the password</td></tr>
<tr><td>+12
Length2</td><td>+14
Flags2</td><td>+15
Reserved</td></tr>
</table>

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.
.xxx xxxx	Reserved bits.

UID2PSWD: The parse service routine uses the IKJPOSIT macro to build a six-word PDE to describe a UID2PSWD parameter. It has the following format:

+0 A pointer to the userid		
+4 Length1	+6 Flags1	+7 Reserved
+8 A pointer to password1		
+12 Length2	+14 Flags2	+15 Reserved
+16 A pointer to password2		
+20 Length3	+22 Flags3	+23 Reserved

Pointer to the userid:

Contains a pointer to the beginning of the userid. It contains zero if the userid was omitted.

Length1:

Contains the length of the userid.

Flags1:

0...	Userid is not present.
1...	Userid is present.
.xxx xxxx	Reserved.

Pointer to password1:

Contains a pointer to the beginning of password1. It contains zero if the password1 is omitted.

Length2:

Contains the length of password1, excluding the slash.

Flags2:

0...	Password1 is not present.
1...	Password1 is present.
.xxx xxxx	Reserved.

Pointer to password2:
Contains a pointer to the beginning of password2. It contains zero if the password2 is omitted.

Length3:
Contains the length of password2, excluding the slash.

Flags3:

0...	Password2 is not present.
1...	Password2 is present.
.xxx xxxx	Reserved.

CONSTANT: The parse service routine uses the IKJTERM 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:

Byte1:	
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 minus.
..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: The parse service routine 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 does not include the following period. Contains zero if the program-id is not present.

Length2:

Contains the length of the line number entered but does not include the delimiting periods. Contains zero if the line number is not present.

Length3:

Contains the length of the verb number entered but does not include 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: The parse service routine 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: The parse service routine 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:

<table>
<tr><td>+0
Reserved</td><td colspan="2">+2
Reserved-word number</td></tr>
<tr><td>+4
Reserved</td><td>+6
Flags</td><td>+7
Reserved</td></tr>
</table>

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.

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, this field is set to zero.

Flags:

Byte1:

0...	The parameter is missing.
1...	The parameter is present.
.xxx xxxx	Reserved.

EXPRESSION: The parse service routine 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: The parse service routine 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 positional parameter</td></tr>
<tr><td>+4
Length</td><td>+6
Flags</td><td>+7
Reserved</td></tr>
</table>

Pointer to the positional parameter:

Contains a pointer to the beginning of the positional parameter. If INTEG was specified on the IKJIDENT macro instruction, this will contain a pointer to a fullword binary value.
Contains zero if the positional parameter is omitted.

Length:

Contains the length of the positional parameter.

Flags:

0...	The parameter is not present.
1...	The parameter is present.
.xxx xxxx	Reserved bits.

Effect 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
- HEX
- INTEG
- CHAR
- 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 X'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 X'FF000000'.

Figure 141 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.

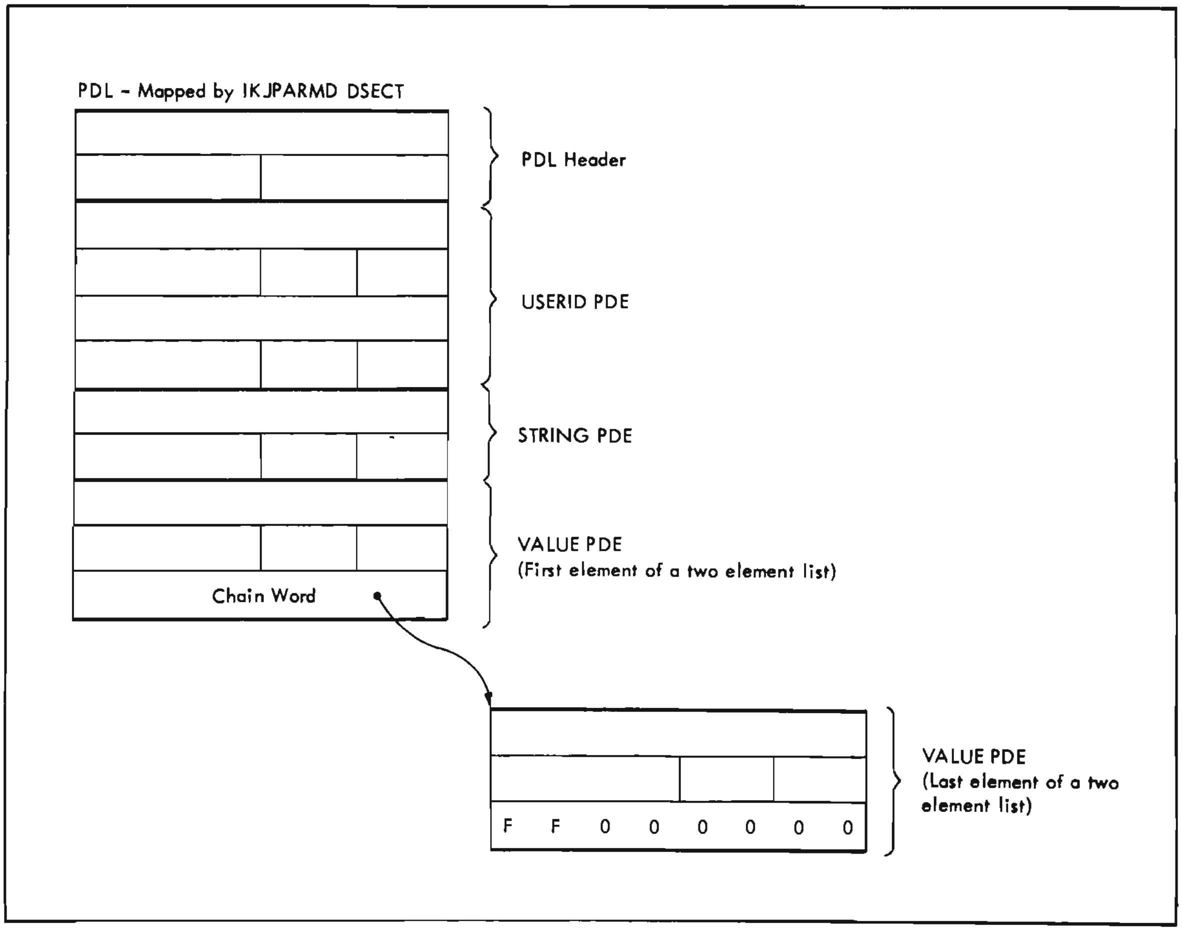

PDL - Mapped by IKJPARMD DSECT
PDL Header
USERID PDE
STRING PDE
VALUE PDE
(First element of a two element list)
Chain Word
VALUE PDE
(Last element of a two element list)
F F 0 0 0 0 0 0

Figure 141. A PDL Showing PDEs Describing a List

RANGE: The RANGE option may be used with the following positional parameter types:

- HEX (X' ' only)
- ADDRESS
- VALUE
- CONSTANT
- VARIABLE
- STATEMENT NUMBER
- INTEG
- 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 142 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.

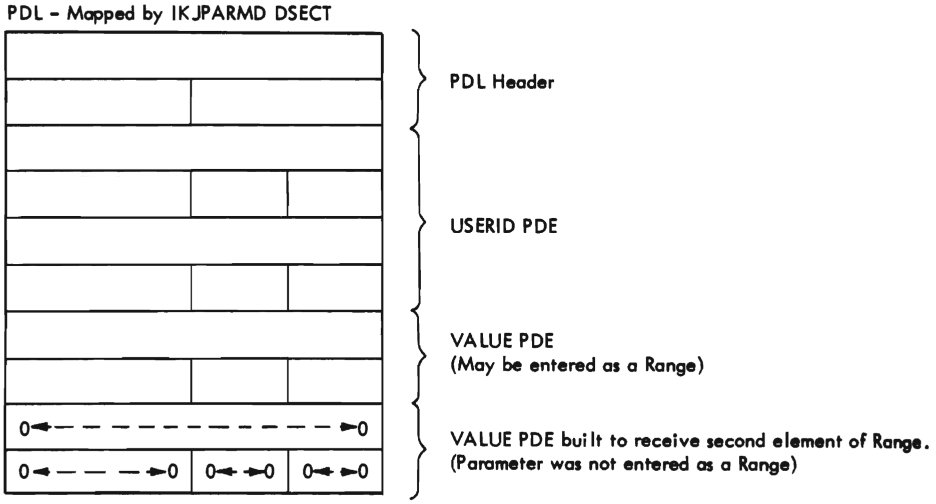

PDL - Mapped by IKJPARMD DSECT
PDL Header
USERID PDE
VALUE PDE
(May be entered as a Range)
0 0
0 0 0 0 0 0
VALUE PDE built to receive second element of Range.
(Parameter was not entered as a Range)

Figure 142. 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 143 shows this general format.

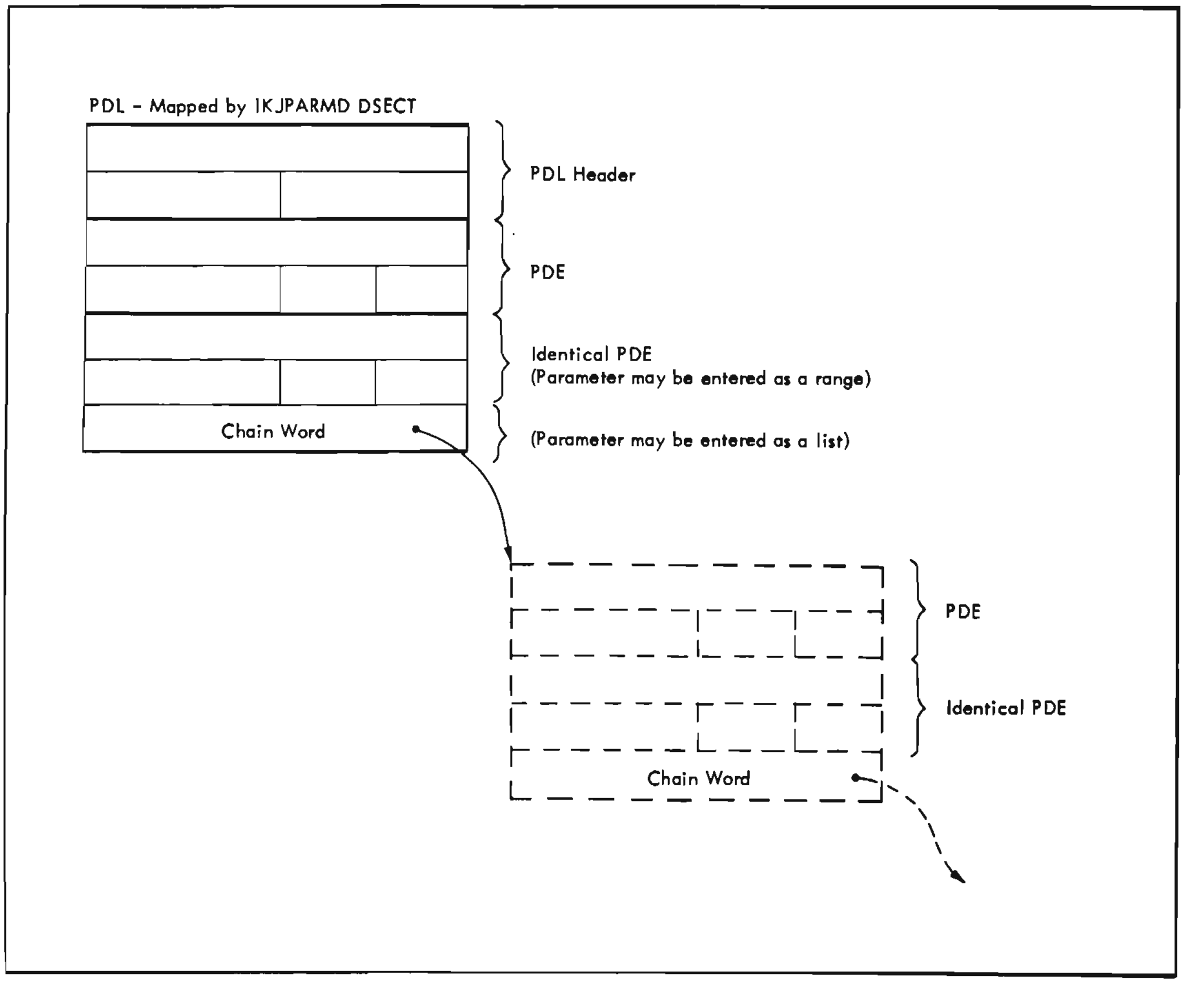

PDL - Mapped by IKJPARMD DSECT
PDL Header
PDE
Identical PDE
(Parameter may be entered as a range)
Chain Word
(Parameter may be entered as a list)
PDE
Identical PDE
Chain Word

Figure 143. A 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 144 shows the PDL returned by the parse service routine if the user enters a single parameter.

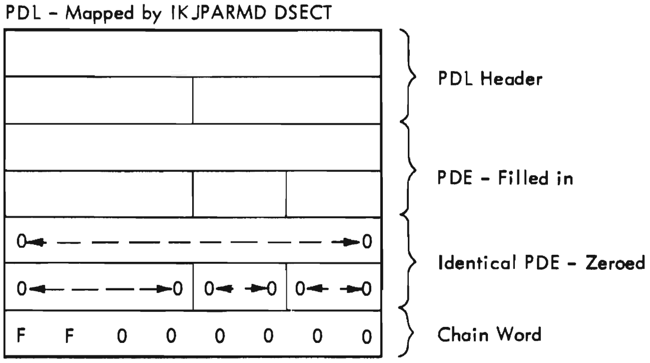


Figure 144. PDL - LIST and RANGE Acceptable, Single Parameter Entered

As Figure 144 further 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 145 shows the PDL returned by the parse service routine if the user enters a single range of the form:

```
parameter:parameter
```

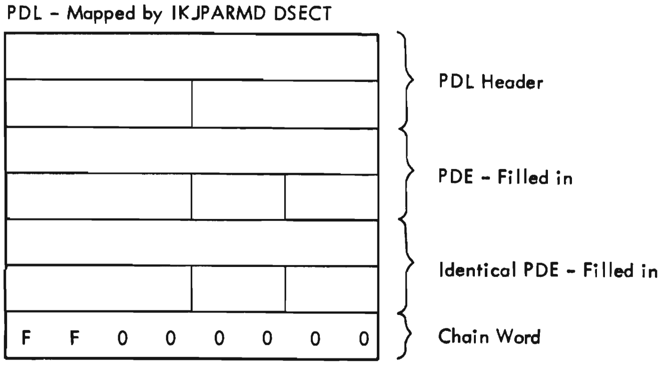


Figure 145. PDL - LIST and RANGE Acceptable, Single Range Entered

As Figure 145 further shows, both PDEs are filled in to describe the single RANGE parameter entered by the user. The chain word is set to X‘FF000000’ to indicate that there are no elements chained onto this one; that is, the parameter was not entered in the form of a LIST.

Figure 146 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,...)
```

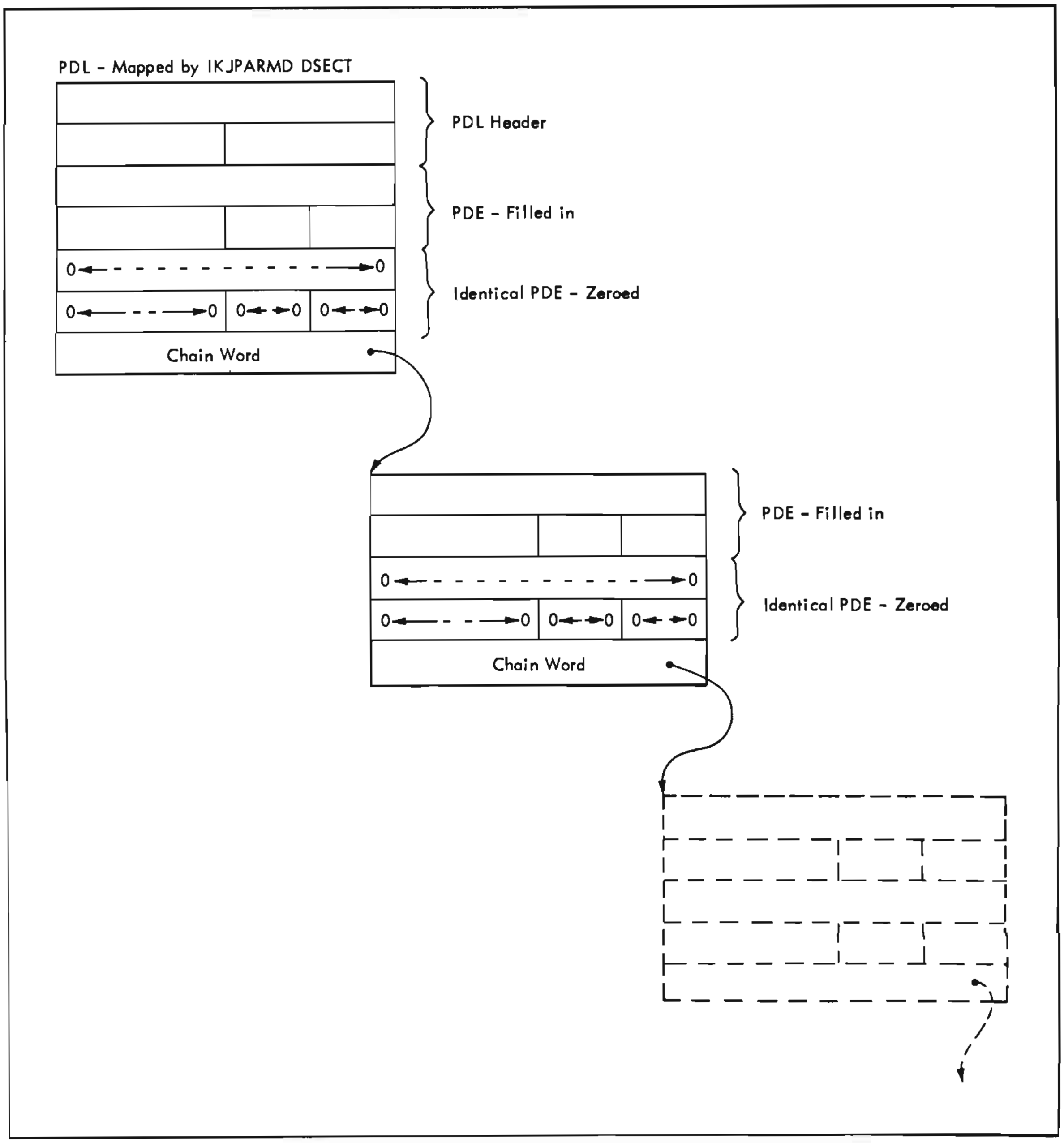


Figure 146. PDL - LIST and RANGE Acceptable, LIST Entered

As Figure 146 further 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 X'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 the parse service routine to describe the ranges entered.

Figure 147 further 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,...)
```

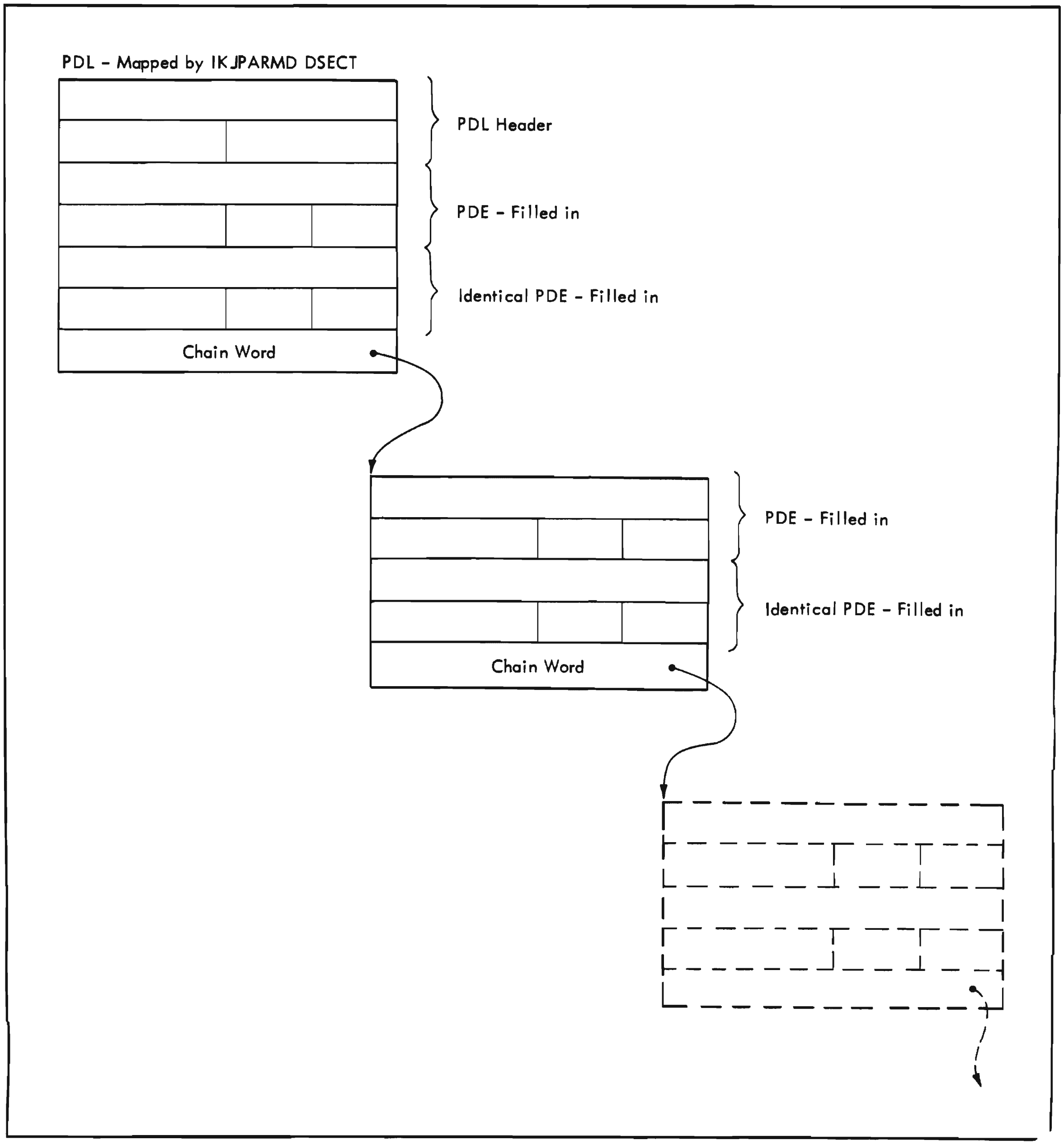

PDL - Mapped by IKJPARMD DSECT
PDL Header
PDE - Filled in
Identical PDE - Filled in
Chain Word
PDE - Filled in
Identical PDE - Filled in
Chain Word

Figure 147. PDL - LIST and RANGE Acceptable, List of Ranges Entered

As Figure 147 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 X'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 into the PDE the number of the IKJNAME macro instruction that corresponds to the keyword name entered.
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 Uppercase

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.

The parse service routine always translates keyword parameters to uppercase.

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, the parse service routine inserts the default value.

The parse service routine 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.

The parse service routine can call validity checking routines for the following types of positional parameters:

- HEX
- VALUE
- ADDRESS
- QSTRING
- USERID
- DSNAME
- DSTHING
- CONSTANT
- VARIABLE
- STATEMENT NUMBER
- EXPRESSION
- JOBNAME
- INTEG
- 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 the 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, the parse service routine calls the validity check routine only after both items of the range are parsed.

When control is passed from the parse service routine to a validity checking routine, the parse service routine uses standard linkage conventions. The validity check routine must save parse's registers and restore them before returning control to the parse service routine. 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 148.

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 X'FF000000' by parse. A user-provided validity checking routine can place the address of a second level message in this field.

Figure 148. 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 the parse service routine of the results of the validity check and determine the action that parse should take. Figure 149 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.	The parse service routine 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.	The parse service routine stops all further syntax checking and returns to the calling routine.

Figure 149. Return Codes from a Validity Checking Routine

If the parse service routine 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.

The format of these second level messages is the same as the HELP second level message portion of the PCE for the macro from which the validity checking routine received control.

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. The parse service routine 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. The parse service routine 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 "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. The parse service routine 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 the parse service routine returned to the original command buffer and found a positional parameter.

   Note 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. The parse service routine 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 ensure that the correct password was entered if one was required, by checking the password pointed to by the PDE returned by the parse service routine.

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, the parse service routine accepts one or more parameters after the prompt. The parse service routine considers these newly entered parameters to be part of the original list. Parameters that are 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 the parse service routine 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.

The parse service routine 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

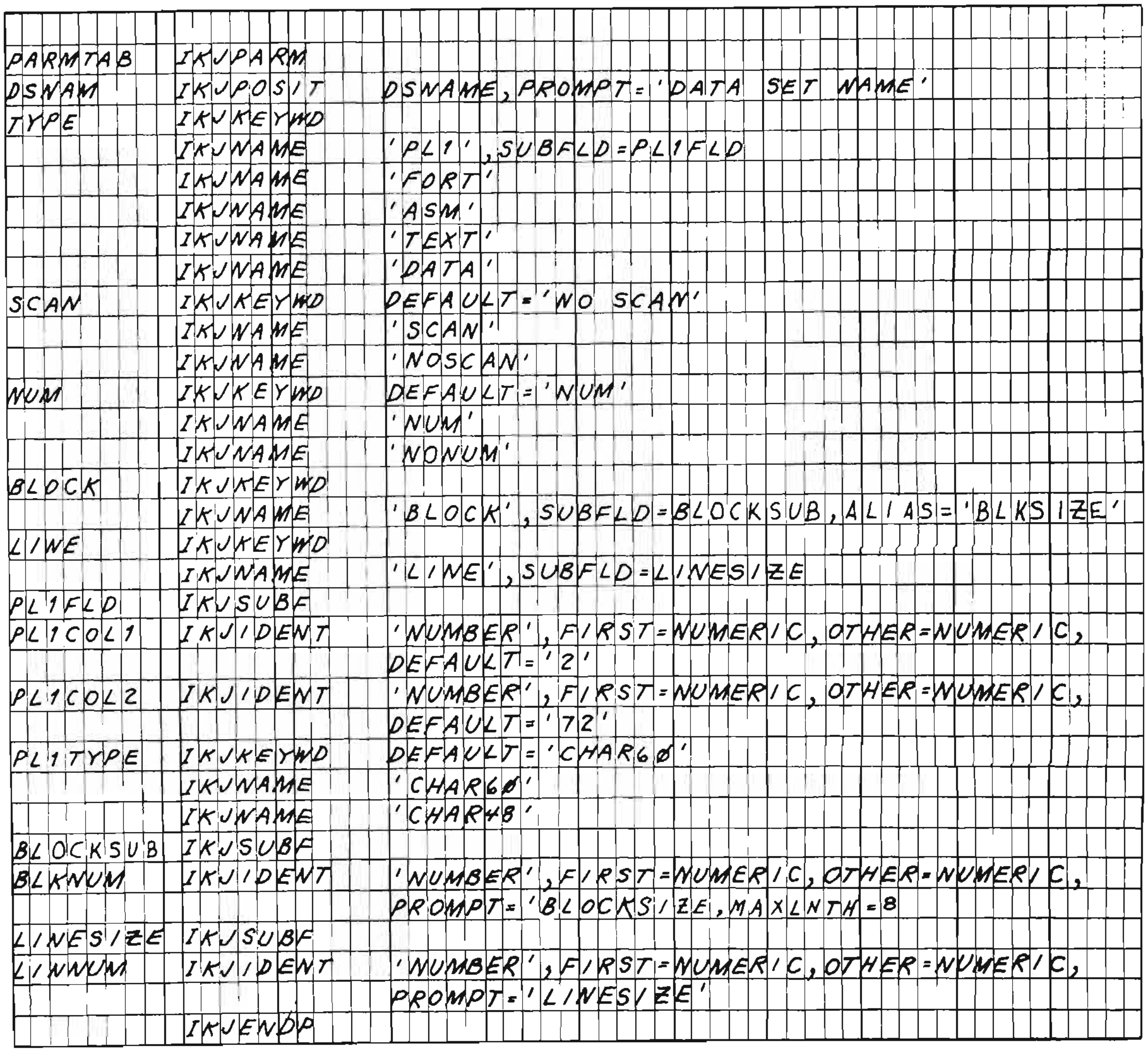
Example 1

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 sample EDIT command we are describing to the parse service routine has the following format:

```
EDIT     dsname
         ┌                                                   ┐
         │ PLI   ┌ (┌ number   ┌ number ┐┐┌ CHAR60 ┐) ┐       │
         │       └  └   2      └   72   ┘┘└ CHAR48 ┘  ┘       │
         │                                                   │
         │ FORT                                              │
         │ ASM                                               │
         │ TEXT                                              │
         └ DATA                                              ┘

         ┌ SCAN   ┐
         └ NOSCAN ┘

         ┌ NUM   ┐
         └ NONUM ┘

         ┌ BLOCK (number)   ┐
         └ BLKSIZE (number) ┘

           LINE(number)
```

Figure 150 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=BLOCKSUB,ALIAS='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'
BLOCKSUB IKJSUBF
BLKNUM   IKJIDENT  'NUMBER',FIRST=NUMERIC,OTHER=NUMERIC,
                   PROMPT='BLOCKSIZE,MAXLNTH=8
LINESIZE IKJSUBF
LINNUM   IKJIDENT  'NUMBER',FIRST=NUMERIC,OTHER=NUMERIC,
                   PROMPT='LINESIZE'
         IKJENDP
```

Figure 150. Coding Example 1 - Using Parse Macros to Describe Command Parameter Syntax

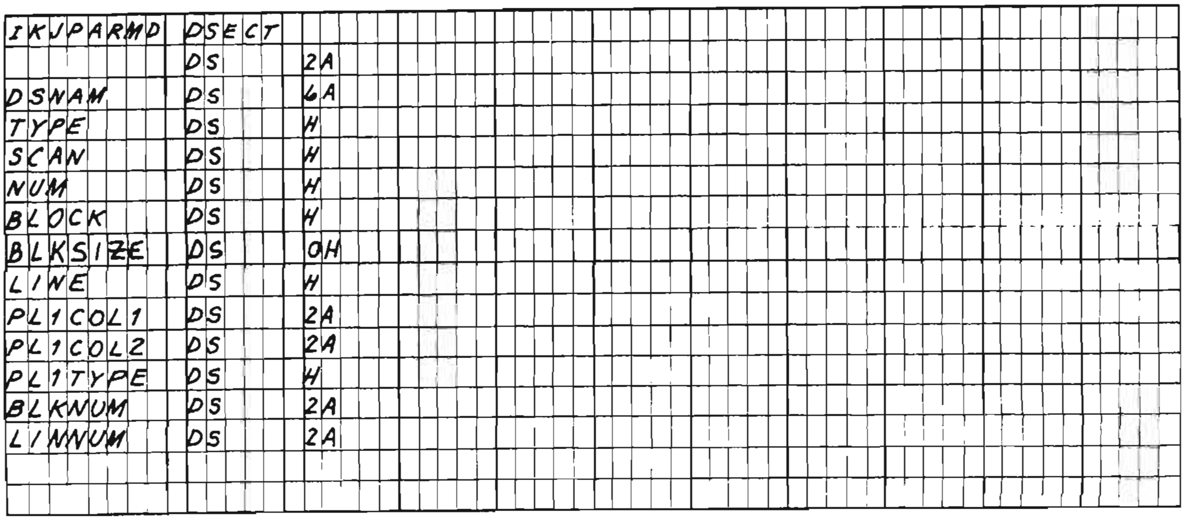
The parse macro instructions shown in Figure 150 perform two distinct functions when executed:

1. 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. 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 151 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
BLKSIZE  DS    0H
LINE     DS    H
PL1COL1  DS    2A
PL1COL2  DS    2A
PL1TYPE  DS    H
BLKNUM   DS    2A
LINNUM   DS    2A
```

Figure 151. 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
```

the parse service routine would prompt for the blocksize as follows:

```
ENTER BLOCKSIZE
```

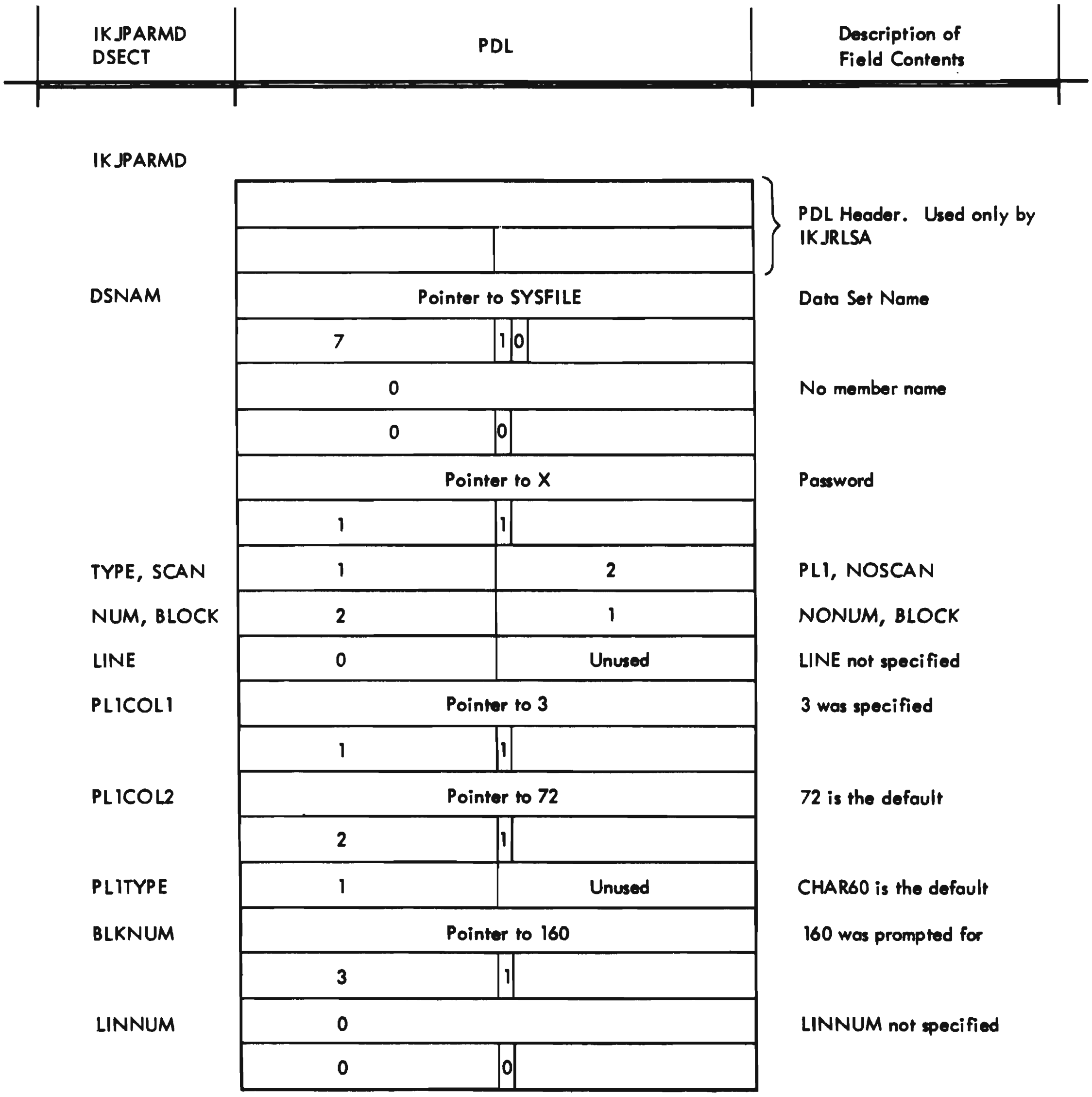
The user at the terminal might respond with:

```
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 152 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	

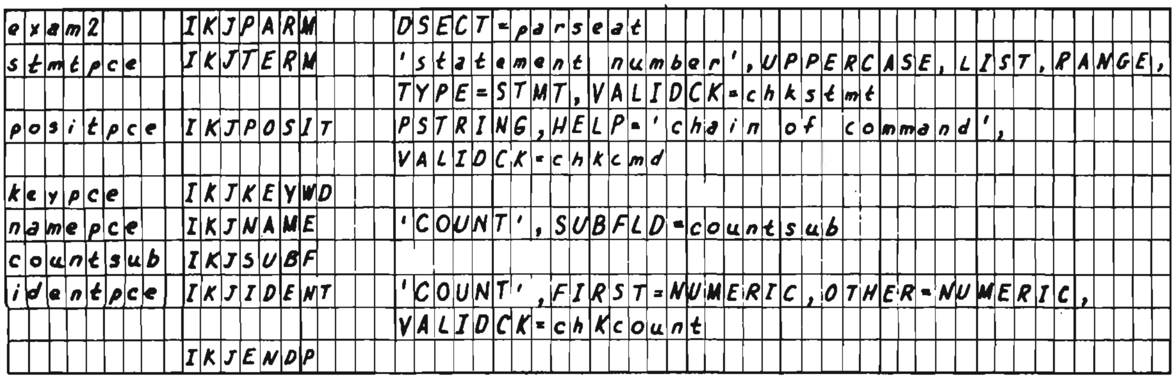
Figure 152. The IKJPARMD DSECT and the PDL (Example 1)

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)

Figure 153 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
```

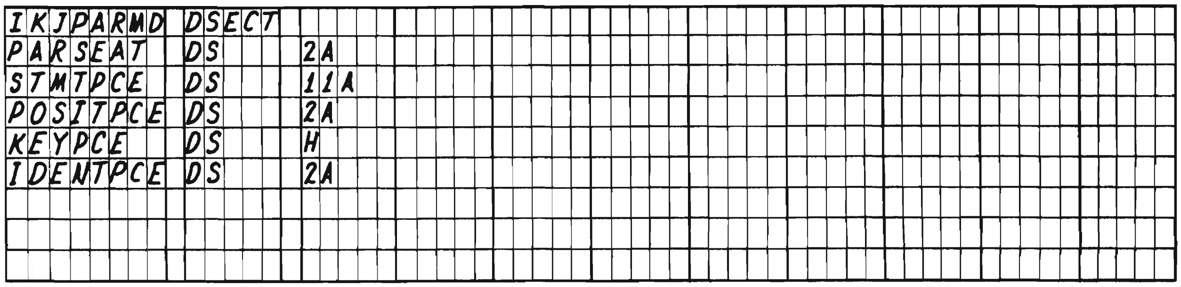
Figure 153. Coding Example 2 - Using Parse Macros to Describe Parameter Syntax

The parse macro instructions shown in Figure 153 perform two distinct functions when executed:

1. 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. Create the IKJPARMD DSECT that you use to map the parameter descriptor list. The PDL is returned by the parse service routine 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 154 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 154. 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 service routine would prompt the terminal user with the message:

```
INVALID STATEMENT NUMBER, 200/3
REENTER
```

The user might respond with:

```
200.3
```

the parse service routine would then prompt the user with:

```
INVALID COUNT, a
REENTER
```

The user might respond with:

```
3
```

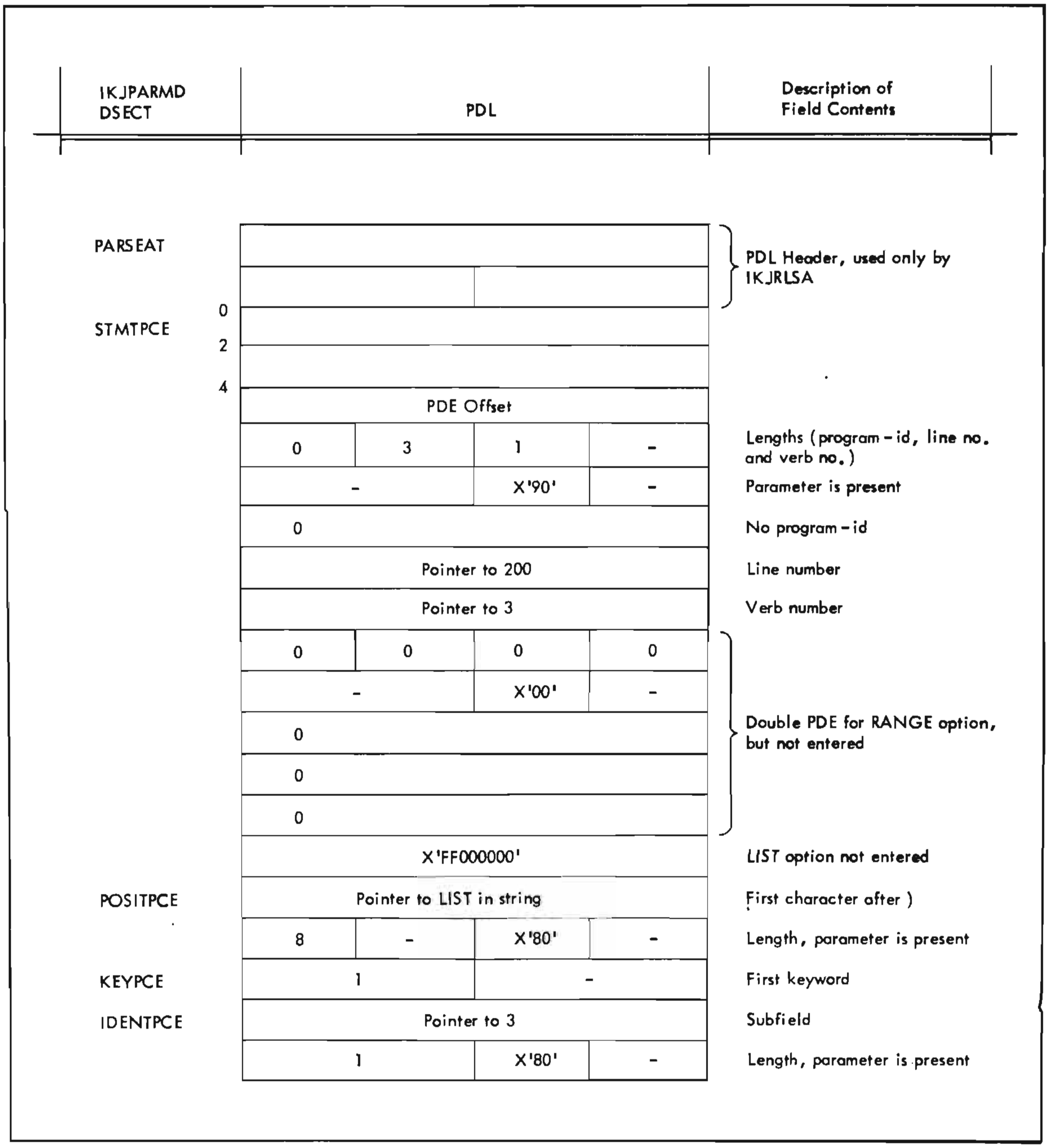
This sequence resulted in the syntactically correct command of:

```
at 200.3 (list all) count(3)
```

The parse service 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.

The parse service routine then returns to the caller and the caller uses the address of the PDL as a base address for the IKJPARMD DSECT.

Figure 155 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					
	2					
	4	PDE Offset				
		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

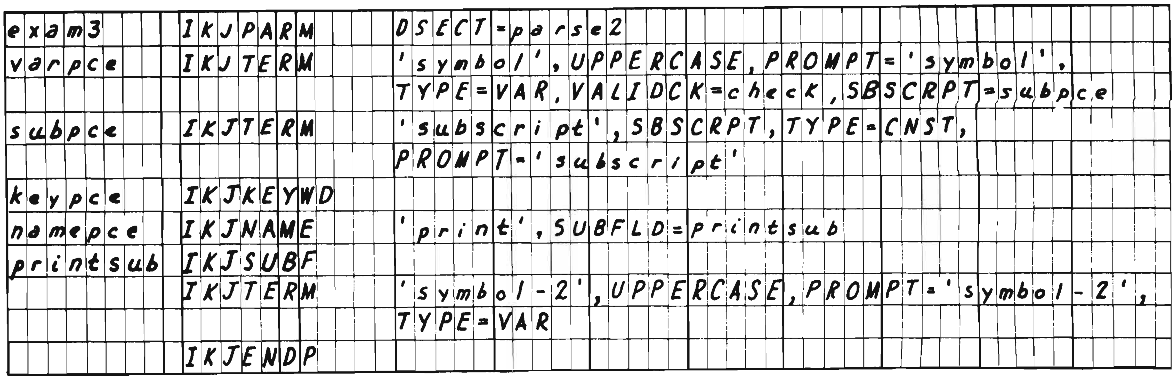
Figure 155. The IKJPARMD DSECT and the PDL (Example2)

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)

Figure 156 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
```

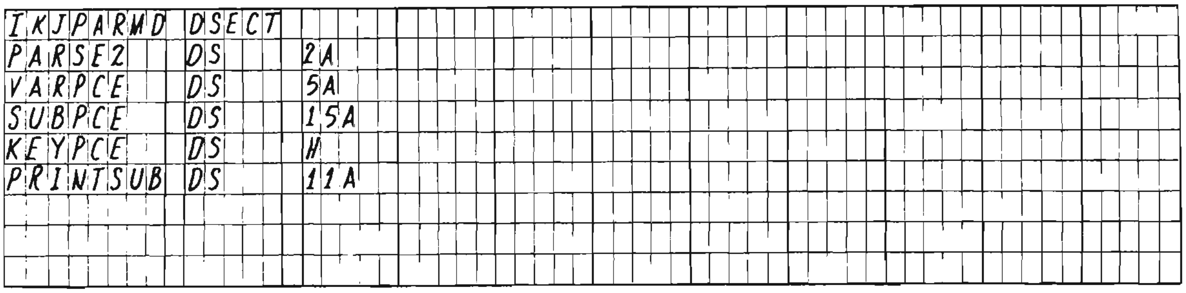
Figure 156. Coding Example 3 - Using Parse Macros to Describe Parameter Syntax

The parse macro instructions shown in Figure 156 perform two distinct functions when executed:

1. 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. Create the IKJPARMD DSECT that you use to map the parameter descriptor list. The PDL is returned by the parse service routine 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 157 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 157. 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 service routine would prompt the terminal user with:

```
INVALID SYMBOL, a...1 in 3(1)
REENTER
```

The user might respond with:

```
a of b in 3(1)
```

the parse service 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 service 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.

The parse service routine then returns to the caller and the caller uses the address of the PDL as a base address for the IKJPARMD DSECT.

Figure 158 shows the PDL returned by the parse service 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.

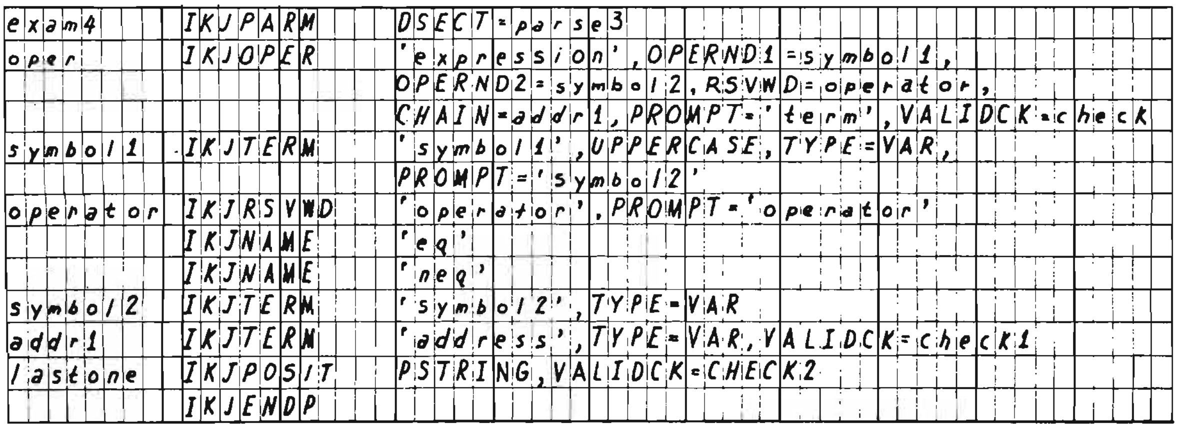
Figure 158. The IKJPARMD DSECT and the PDL (Example 3)

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)

Figure 159 shows the sequence of parse macro instructions that describe this sample WHEN command to the parse service routine.



```
exam4    IKJPARM  DSECT=parse3
oper     IKJOPER  'expression',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
```

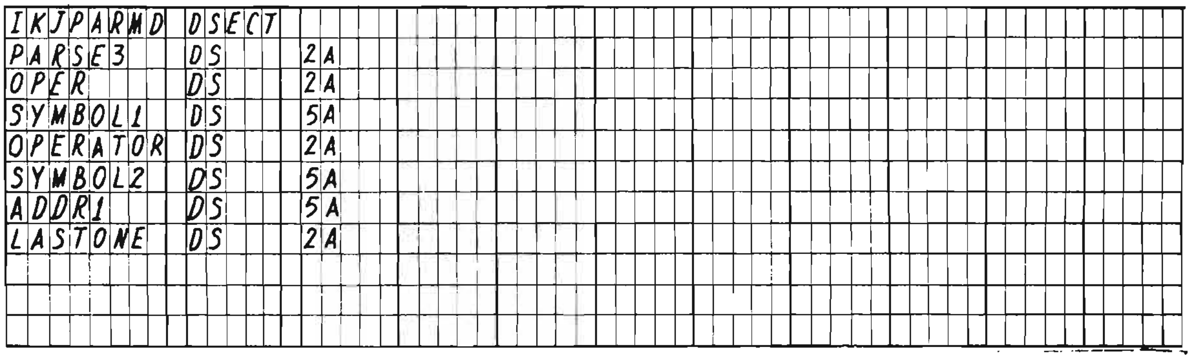
Figure 159. Coding Example 4 - Using Parse Macros to Describe Parameter Syntax

The parse macro instructions shown in Figure 159 perform two distinct functions when executed:

1. 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. Create the IKJPARMD DSECT that you use to map the parameter descriptor list. The PDL is returned by the parse service routine 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 160 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 160. 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 service routine would prompt the terminal user with:

```
ENTER OPERATOR
```

The user might then respond:

```
eq
```

the parse service 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 service 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.

The parse service routine then returns to the caller and the caller uses the address of the PDL as a base address for the IKJPARMD DSECT.

Figure 161 shows the PDL returned by the parse service 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 161. The IKJPARMD DSECT and 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.
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.
28	Terminal has been disconnected.

No error message is needed for return codes 4 and 20. The parse service routine issues a message before returning a code of 4 and the validity checking routine issues an error message before it requests termination. The GNRLFAIL routine can be invoked to issue meaningful error messages for the other parse return codes. See "GNRLFAIL/VSAMFAIL Routine (IKJEFF19)" in this book.

Catalog Information Routine (IKJEHCIR)

The catalog information routine retrieves information from the system catalog. This information may include data set name, index name, control volume address, or volume ID. The information may be requested from a specific user catalog, or, if no catalog was specified, the system default catalog search is used. An entry code indicates the requested kind of information as follows:

- The next level qualifiers for a name
- All names having the same name as the high-level qualifier and the data set type associated with each name
- The volume serial numbers and device types associated with a name

The requester can also ask for combinations of the information above.

The catalog information routine resides in SYS1.LPALIB and executes with the protection key of the caller.

IKJEHCIR may be invoked in either 24- or 31-bit addressing mode. However, all input passed to IKJEHCIR must reside below 16 megabytes. IKJEHCIR executes in 24-bit addressing mode and returns control in the same addressing mode in which it is invoked.

The catalog information routine parameter list (CIRPARM) is shown in Figure 162.

Number of Bytes	Field	Contents or Meaning
1	CIROPT	Entry code/options used. See Figure 163.
2		Reserved.
1	CIRLOCRC	Locate return code.
4	CIPSRCH	Address of the search argument. This search argument is a userid and a data set name which are names of catalog index levels.
4	CIRCVOL	Address of the volume ID of CVOL. If not given, SYSRES is assumed.
4	CIRWA	Address of the user work area. See Figure 164 for the user work area.
4	CIRSAVE	Address of a 72-byte save area.
4	CIRPSWD	Address of an 8-byte password or zero.

Figure 162. Catalog Information Routine Parameter List (CIRPARM)

The CIROPT values and data returned are shown in Figure 163.

Code	Meaning	Data Required
X'01'	Move the data set names having one more level of qualifier above what the user specified.	8-byte qualifiers are moved into the user's work area.
X'02'	Move all data set names to the user work area.	44-byte data set names are moved to the user work area.
X'04'	Get a volume associated with a given data set name.	Volume information is moved to the user work area. See Figure 165 for volume information format.
X'05'	Get the next level data set name and volume information.	44-byte data set name and volume information is moved to the user work area.
X'06'	Get all level data set names and volume information.	44-byte data set name followed by volume information is moved to the user work area for all levels.

Figure 163. Data Returned from Valid CIROPT Values

Note: For codes X'05', and X'06' the first byte of the field will contain one of the following data set types:

- V for volume
- C for cluster
- G for alternate index
- R for path
- F or FREE
- Y for upgrade
- B for GDG base
- X for alias name
- P for page space
- M for master catalog
- U for user catalog
- A for non-VSAM data set

The user work area that is based on CIRWA is shown in Figure 164.

Number of Bytes	Field	Contents or Meaning
2	AREALN	Length of work area.
2	DATALIN	Length of data returned +4.
Variable	DATA	The area data is stored.

Figure 164. User Work Area for CIRPARM

Figure 165 describes the format of the volume information.

Number of Bytes	Field	Contents or Meaning
2		Reserved.
4	DEVTYP	Device type.
6	VOLSER	Volume serial number.
2	FILESEQ	File sequence number. (This field is provided for compatibility with the OS/VS catalog, and is used for non-VSAM data sets that reside on tape volumes.)
1		Reserved.

Figure 165. Volume Information Format

Return Codes from IKJEHCIR

The IKJEHCIR return codes have the following meaning:

Return Code Hexadecimal	**Meaning**
00	Successful completion of the request.
04	The LOCATE macro instruction has failed. The LOCATE return code will be stored in CIRLOCRC.
0C	Volumes were returned by LOCATE, indicating a dsname (fully qualified) was passed in the parameter list but options other than volumes were requested. The list of the volumes returned by LOCATE is in the work area.

Return Codes from LOCATE

The LOCATE return codes have the following meaning:

Return Code Hexadecimal	**Meaning**
00	Successful completion of the request.
04	Required VSAM volume was not mounted or the specified volume was not open.
08	The data set name qualifier was not found.
18	A permanent I/O error was found when processing the catalog.
20	User work area outside user region or invalid user-supplied parameter list.
24	User catalog must be allocated *and opened.*
2C	Work area too small.
38	Password verification failure.
3C	STEPCAT or JOBCAT not open.

Default Service Routine (IKJEHDEF)

The default service routine (IKJEHDEF) constructs a fully-qualified data set name when the calling routine provides a partially-qualified data set name. A fully-qualified data set name has three fields: a userid, a data set name, and a descriptive qualifier.

Use the CALL, CALLTSSR or LINK macro instruction to invoke the default service routine.

IKJEHDEF may be invoked in either 24- or 31-bit addressing mode. However, all input passed to IKJEHDEF must reside below 16 megabytes. IKJEHDEF executes in 24-bit addressing mode and returns control in the same addressing mode in which it is invoked.

At entry, general register 1 must point to the default parameter list (DFPL). IKJEHDEF then invokes the catalog information routine (IKJEHCIR) to search the system catalog for the required qualifiers. When the search argument is satisfied, the default service routine returns to the calling control program. All of the general registers are restored except for register 15 which contains the return code.

Note: For additional information concerning the default service routine, see *Terminal Monitor Program and Service Routines Logic.*

Appendix A: 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 **4-8**.

2. Uppercase letters and words, numbers, and the set of symbols listed below should be written in macro instructions exactly as shown in the definition:

   apostrophe '
   asterisk *
   comma ,
   equal sign =
   parentheses ()
   period .

3. Lowercase 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

   $$\text{ALPHA=(}\begin{Bmatrix}\text{A}\\\text{B}\\\text{C}\end{Bmatrix}\text{,D)}$$

   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. Hyphens join lowercase letters, words, and symbols to form a single 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.

7. An underscore indicates a default option. If an underscored alternative is selected, it need not be written in the actual macro instruction.

   Example: The representation

   $$\begin{matrix} \texttt{A} \\ \texttt{B} \\ \texttt{C} \end{matrix} \quad \texttt{or} \quad \begin{Bmatrix} \texttt{A} \\ \texttt{B} \\ \texttt{C} \end{Bmatrix}$$

   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.

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

   Example: The representation

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

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

Appendix B: Using VTAM Full-Screen Mode (STFSMODE and STLINENO)

You should use the STFSMODE and STLINENO macros if your command processor issues full-screen messages (TPUT macros with the FULLSCR or NOEDIT operands, or TPG macros). The following paragraphs discuss uses of STFSMODE and STLINENO and aspects of command processors operating in full-screen mode.

Notes:

1. In the following text, full-screen message refers to a TPUT FULLSCR, TPUT NOEDIT, or TPG depending on your environment. Also, non-full-screen message refers to a non-full-screen TPUT.
2. This section describes considerations for coding programs that provide the functions of a full-screen editor like SPF. For example, the full-screen messages issued in Figures 166-169 might be primary menus.

Protection of Screen Contents

A full-screen command processor should use the STFSMODE macro to prevent unexpected non-full-screen messages from overlaying the screen. For instance, unexpected messages from the operator or from other TSO users could cause invalid input to be sent to the command processor. Also, the STFSMODE macro prevents full-screen messages from overlaying unexpected non-full-screen messages before the user has a chance to read them.

Figure 166 illustrates what happens when a command processor, operating in full-screen mode, issues a full-screen message while non-full-screen messages are being displayed at the terminal.

COMMAND PROCESSOR	TSO/VTAM	TERMINAL
input ←(1)	input ←(1) non-full-scrn msg1 (1)→ non-full-scrn msg2 (2)→	←(1) input ←(1) ENTER (1)→ non-full-scrn msg1 (2)→ non-full-scrn msg2
TPUT (3)→ (full-scrn msg1) (3)→ TGET (3)→	*** (4)→	(4)→ ***
RESHOW ←(5a) TPUT (6)→ (full-scrn msg1) (6)→ TGET (6)→	ENTER ←(5) full-scrn msg1 (6)→	←(5) ENTER (7)→ full-scrn msg1

Figure 166. Function of RESHOW Code in Full-Screen Message Processing

The following events occur in Figure 166.

1. As soon as the user presses the ENTER key to send input to the command processor, TSO/VTAM clears the screen, the alarm sounds (if the terminal is so equipped), and TSO/VTAM then displays the non-full-screen message.

2. As long as TSO/VTAM receives non-full-screen messages, it displays them, one after another on the screen.

3. The command processor's normal processing of the input received from the terminal (see step 1) may cause it to construct a new full-screen message. The command processor issues a TPUT and a TGET.

4. When TSO/VTAM receives the full-screen message, it displays three asterisks (***) at the terminal and unlocks the keyboard to ensure that the user has time to view the non-full-screen messages.

5. When the user presses the ENTER key to acknowledge having seen the messages, TSO/VTAM:

   a. Puts a RESHOW code on the input queue to indicate to the command processor that the screen contents should be completely restored. This RESHOW code is picked up by the command processor's current TGET request.

   b. Discards the full-screen message constructed by the command processor.

3. The command processor responds to the RESHOW code by issuing a full-screen message(s) to restore the screen contents and to request new input from the user.

4. Finally, TSO/VTAM displays the full-screen message(s) at the terminal.

Use of TGET

To protect the screen, TSO/VTAM discards full-screen messages that immediately follow non-full-screen messages. Therefore the full-screen command processor must issue a TGET macro after every TPUT FULLSCR macro so that it can receive the RESHOW code. When a TGET is issued following a TPUT FULLSCR, VTAM unlocks the display keyboard. When a TGET is issued following a TPUT NOEDIT or a TPG, VTAM does not unlock the keyboard. Users of TPUT NOEDIT and TPG are responsible for all device command and write-control-character bit settings.

Screen Content Restoration

Upon receiving a RESHOW code, the full-screen command processor must be able to restore the complete contents of the screen-- for example, reissue the full-screen message. The VTAM default RESHOW code is X'6E', the key-code for PA2. If the command processor uses any other PF key for the RESHOW code, it must specify the RSHWKEY keyword on the STFSMODE macro when it first turns on full-screen mode. To set the RESHOW code, issue STFSMODE ON, RSHWKEY=n, where n is the PF key number. VTAM uses the hexadecimal representation of the specified PF key as the RESHOW code.

NOEDIT Mode

To obtain input (via a TGET macro) that is not edited in any way, specify the NOEDIT keyword on STFSMODE. Regardless of the options specified on the TGET macro, in NOEDIT mode, the data is not edited, broken into separate input lines, or modified. VTAM receives the input from the terminal and puts it on the input queue intact. To establish NOEDIT mode, the command processor must issue STFSMODE ON,NOEDIT=YES. (Use of the NOEDIT keyword has no effect on the treatment of TPUTs and TPGs.)

Full-Screen Protection Responsibilities of Attention Exit Routines

To maintain screen protection when the user presses the PA1 or ATTENTION key, the command processor must have an attention exit routine. When the terminal user presses the PA1 or *ATTENTION* key, VTAM sets FULLSCR to OFF, the RESHOW code to the default, and NOEDIT mode to NO. If the command processor does not have an attention exit and the user presses ENTER (in response to the attention indication), the command processor resumes execution at the point of interruption with these default values. If the command processor has an attention exit routine, the exit routine should issue *the* STFSMODE macro to reestablish full-screen mode, the desired RESHOW code, and NOEDIT mode. In this way, the attention exit routine maintains screen protection.

Determining Screen Protection in Full-Screen Mode

The first time the command processor issues the STFSMODE macro to establish full-screen mode, it may specify INITIAL=YES to prevent unnecessary protection of the screen contents. This issuance of the macro sets full-screen mode on and prevents the acknowledgement message from being sent to the terminal when the last transaction at the terminal was input. The INITIAL indicator is set to NO after the first full-screen message.

Figure 167 illustrates a situation in which INITIAL=YES is specified on the STFSMODE macro and the first message issued is a full-screen message.

COMMAND PROCESSOR	TSO/VTAM	TERMINAL
	READY (1)→ cmdname←(1)	(1)→READY ←(1) cmdname
STFSMODE (2) INITIAL=YES TPUT (2)→ (full-scrn msg1) (2)→ TGET (2)→	full-scrn msg1 (3)→	(3)→full-scrn msg1

Figure 167. Function of INITIAL=YES When First Message is Full-Screen

The following events occur in Figure 167:

1. In response to the READY message, the user enters a command name, SPF for example.
2. The command processor issues the STFSMODE macro with INITIAL=YES. Then, the CP sends a full-screen message the terminal.
3. TSO/VTAM sends the message without warning because INITIAL=YES has been specified and because its previous interaction with the terminal involved input, not output. There is nothing to protect.

If the command processor specifies INITIAL=YES on the STFMODE macro, and the first message is a non-full-screen message, VTAM ignores the keyword and protects the screen contents. Figure 168 illustrates this situation.

COMMAND PROCESSOR	TSO/VTAM	TERMINAL
	READY (1)→ cmdname ←(1)	(1)→READY ←(1) cmdname
STFSMODE (2) INITIAL=YES	non-full-scrn msg1 (3)→	(3)→non-full-scrn msg1
TPUT (4)→ (full-scrn msg1) (4)→ TGET (4)→	*** (5)→	(5)→***
RESHOW ←(6a) TPUT (7)→ (full-scrn msg1) (7)→ TGET (7)→	ENTER ←(6) full-scrn msg1 (7)→	←(6) ENTER (8)→full-scrn msg1

Figure 168. Function of INITIAL=YES When First Message is *Non-Full*-Screen

The following events occur in Figure 168:

1. In response to the READY message, the user enters a command name.
2. The command processor issues the STFSMODE macro with INITIAL=YES.
3. TSO/VTAM displays a non-full-screen message, an operator warning for example, which effectively overrides the INITIAL=YES keyword.
4. The command processor sends a full-screen message to the terminal.
5. TSO/VTAM protects the screen contents.
6. When the user presses the ENTER key to acknowledge having seen the messages, TSO/VTAM:
   1. Puts a RESHOW code on the input queue to indicate to the command processor that the screen contents should be completely restored. This RESHOW code is picked up by the command processor's current TGET request.
   2. Discards the full-screen message constructed by the command processor because the screen contents are completely restored.
7. The command processor responds to the RESHOW code by issuing a full-screen message(s) to restore the screen contents and to request new input from the user.
8. Finally, TSO/VTAM displays the full-screen message(s) at the terminal.

When INITIAL=NO is specified, or allowed to default, no full-screen messages are displayed without warning. Figure 169 illustrates an example of this situation.

COMMAND PROCESSOR	TSO/VTAM	TERMINAL
	READY (1)—► cmdname ◄—(1)	(1)—►READY ◄—(1) cmdname
STFSMODE (2) INITIAL=NO TPUT (2)—► (full-scrn msg1) (2)—► TGET (2)—►	*** (3)—►	(3)—►***
RESHOW ◄—(4a) TPUT (5)—► (full-scrn msg1) (5)—► TGET (5)—►	ENTER ◄—(4) full-scrn msg1 (5)—►	◄—(4) ENTER (6)—►full-scrn msg1

Figure 169. Function of INITIAL=NO

The following events occur in Figure 169:

1. In response to the READY message, the user enters a command name.

2. The command processor issues the STFSMODE macro with INITIAL=NO. Then, the CP sends a full-screen message to the terminal.

3. TSO/VTAM protects the screen contents.

4. When the user presses the ENTER key to acknowledge having seen the messages, TSO/VTAM:

   a. Puts a RESHOW code on the input queue to indicate to the command processor that the screen contents should be completely restored. This RESHOW code is picked up by the command processor's current TGET request.

   b. Discards the full-screen message constructed by the command processor because the screen contents are completely restored.

5. The command processor responds to the RESHOW code by issuing a full-screen message(s) to restore the screen contents and to request new input from the user.

6. Finally, TSO/VTAM displays the full-screen message(s) at the terminal.

Exiting and Reentering Full-Screen Mode

If the command processor issues non-full-screen messages (or invokes routines that issue non-full-screen messages), it may issue the STLINENO macro to set full-screen mode off, and to set the line number for the next non-full-screen message. In so doing, the command processor eliminates the screen protection function and determines where the next non-full-screen message will appear. If the line number is set to 1, VTAM clears the screen. When the command processor issues the last non-full-screen message (or when the invoked routine returns control to the command processor), the command processor should issue STFSMODE ON to reestablish full-screen mode. This STFSMODE macro should be issued before the next full-screen message macro is issued.

If the command processor exits full-screen mode, expecting to reenter full-screen mode at a later time before termination, STLINENO should be used to set full-screen mode off. (Use of STFSMODE to set the mode off results in the RESHOW code being set to the default.) After a TGET request, issue STLINENO LINE=n where **n** is the desired line number. When all non-full-screen messages are completed, issue STFSMODE ON before issuing the next full-screen message macro.

You may want either to clear part of the screen before issuing STLINENO, or to display information that is to remain on the screen after the STLINENO macro is issued. In either case, issue a full-screen message macro (including the HOLD option) before issuing the STLINENO. The HOLD option guarantees that the full-screen message reaches the terminal before the STLINENO macro takes effect.

Since VTAM clears the screen when the line number is set to 1, STLINENO LINE=1 is an efficient way for the command processor to clear the screen. Use of a full-screen message macro (including the HOLD option) to clear the screen reduces performance because it causes a swap-out of the address space to wait for the I/O to complete.

Full-Screen Command Processor Termination

When the full-screen command processor terminates, it must issue STFSMODE OFF to exit full-screen mode. This resets the RESHOW code and NOEDIT mode values to the defaults. The following termination procedure is recommended:

When a TGET request is satisfied with data that causes the command processor to begin exit processing, issue the following:

- STLINENO LINE=1 (causes VTAM to clear the screen)
- STFSMODE=OFF (resets the RESHOW code and NOEDIT mode to the defaults)
- non-full-screen TPUTS (optional-- perhaps to provide session summary information)

If the command processor issues a TPUT or TPG macro before (or instead of) issuing the STLINENO macro, it should be issued with the HOLD option to guarantee that the message reaches the terminal before full-screen mode is set off. If the macro is also a full-screen message, a TCLEARQ INPUT should be issued just before termination to clear the

RESHOW code, which may have been put on the input queue by the screen protection function.

Use of TERMINAL BREAK

When a command processor establishes full-screen mode, VTAM treats all devices as if the terminal user had entered the TERMINAL NOBREAK command. If the user specifies TERMINAL BREAK before a full-screen command processor is invoked, VTAM supports the BREAK mode whenever the command processor exits from full-screen mode.

Index

MVS/Extended Architecture TSO Guide to Writing a Terminal Monitor Program or a Command Processor (File No. S370-39) Printed in U.S.A. GC28-1295-0

GC28-1295-0

MVS/Extended Architecture
TSO Guide to Writing a Terminal
Monitor Program or a Command
Processor
GC28-1295-0

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: ***Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.***

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name, company, mailing address, and date:

____

____

____

____

What is your occupation? ______________________________

How do you use this publication? ______________________________

Number of latest Newsletter associated with this publication: ____________________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

Cut or Fold Along Line

GC28-1295-0

Reader's Comment Form

Fold and tape Please Do Not Staple Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department D58, Building 920-2
PO Box 390
Poughkeepsie, New York 12602

Fold and tape Please Do Not Staple Fold and tape

IBM®

Cut or Fold Along Line

MVS/Extended Architecture TSO Guide to Writing a Terminal Monitor Program or a Command Processor (File No. S370-39) Printed in U.S.A. GC28-1295-0