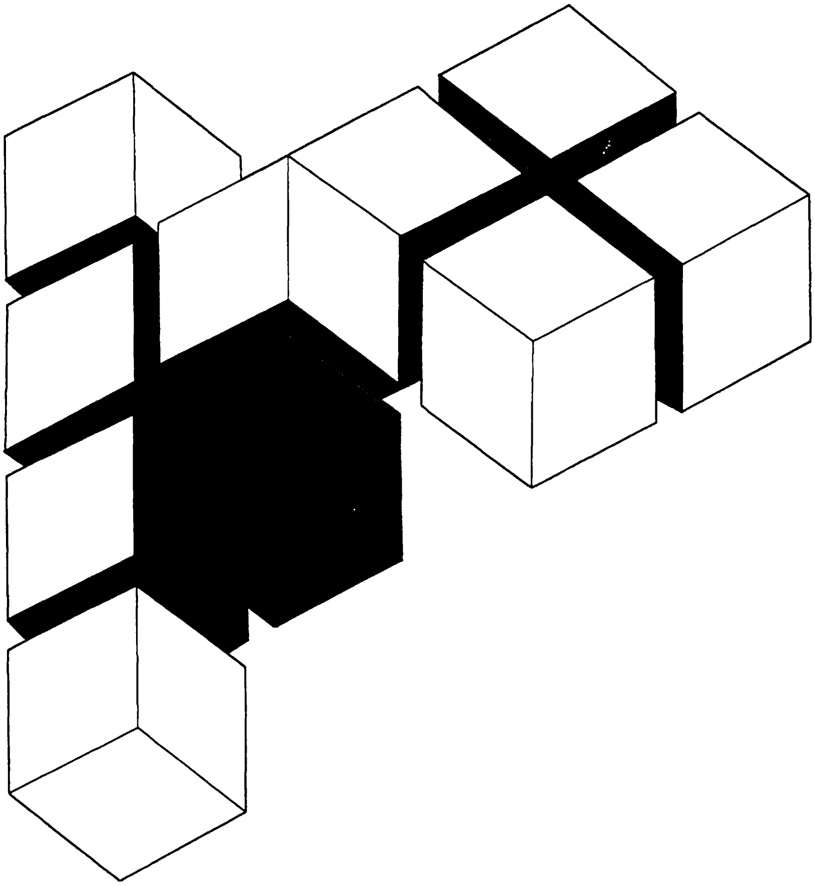
TSO Extensions System Diagnosis: Command Processors, E-S



LY28-1415-0

TSO Extensions System Diagnosis: Command Processors, E-S

First Edition (September, 1987)

This edition applies to TSO Extensions Release 4, Program Number 5665-285 and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Changes are made periodically to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370 Bibliography,* GC20-0001, for the editions that are applicable and current.

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

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

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Information Development, Department D58, Building 921-2, PO Box 390, Poughkeepsie, N.Y. 12602. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Preface

This publication describes the programs that handle the following TSO commands:

EDIT
FREE
HELP
LINK/LOADGO
LISTALC
LISTBC
LISTDS
LOGON
MVSSERV
OPERATOR
OUTPUT
PRINTDS
PROFILE
PROTECT
RACONVRT
RENAME
RUN
SEND
SUBMIT
SYNC

LISTCAT information is in *Access Method Services Logic.*

EXEC command information is documented in the "CLIST Processing and Diagnosis" chapter in *TSO Extensions System Diagnosis: Terminal Monitor Program and Service Routines.*

RECEIVE command information is documented in the "TRANSMIT Command Processing" chapter in *TSO Extensions System Diagnosis: Command Processors, T-Z.*

Who Should Read This Book?

The publication is for people who diagnose TSO/E problems or maintain TSO/E. Usually, this is a system programmer.

The level of detail at which this book is written assumes that the reader:

Can use TSO/E and code Job Control Language (JCL) statements to execute TSO/E programs or cataloged procedures.

Can code in assembler language and can read assembler output.

Can use system messages, system dumps, and IBM publications, such as *Diagnostic Techniques*, to locate errors in problem programs.

How is This Book Organized?

This publication contains information that pertains to TSO/E command processors, their subcommands and subroutines. The commands are presented in alphabetical order, starting with EDIT and ending with SYNC. Data areas and a module directory are included in a a separate chapter at the end of the book.

How Do You Use This Book?

This book is the second of three volumes that describe the TSO/E command processors in alphabetical order. It contains method of operation diagrams and written descriptions designed to help you follow the internal operation of a program and determine the location of a program malfunction. The book provides pointers for specific functions; you can use these pointers to access program listing information without having to scan the listings for the data needed.

Method of operation diagrams are not included for the following commands:

- LISTBC
- MVSSERV
- PRINTDS
- RACONVRT
- SEND
- SYNC

Extended diagnostic information is instead provided to help you isolate and fix a problem with these commands.

If you have never used this book, look over the Table of Contents and each chapter to become familiar with the book's content and method of presentation.

To diagnose a problem that occurs when you use TSO/E, start with *TSO Extensions System Diagnosis: Guide and Index.* Use this, and the other command processor books, to further diagnose a suspected problem in an IBM-supplied command processor. Should you trace a problem to the terminal monitor program or an IBM-supplied service routine, refer to *TSO Extensions System Diagnosis: Terminal Monitor Program and Service Routines.*

If you are unable to solve the problem, see *TSO Extensions System Diagnosis: Guide and Index* for information on how to report a problem to IBM.

Where Can You Find Additional Information?

Additional information is available in the following publications:

TSO Extensions User's Guide, SC28-1333.

TSO Extensions Command Reference, SC28-1307.

TSO Extensions Customization, SC28-1136.

TSO Extensions Programming Guide, SC28-1363.

TSO Extensions Programming Services, SC28-1364.

TSO Extensions System Diagnosis: Command Processors, A-D, SC28-1414.

TSO Extensions System Diagnosis: Command Processors, E-S, SC28-1415.

TSO Extensions System Diagnosis: Command Processors, T-Z, SC28-1416.

TSO Extensions System Diagnosis: Terminal Monitor Program and Service Routines, SC28-1308.

TSO Messages, SC28-1308.

MVS/Extended Architecture Catalog Diagnostic Guide, SY26-3899.

MVS/Extended Architecture Message Library: System Codes, GC28-1157.

MVS/Extended Architecture Message Library: System Messages Volume 1 and Volume 2, GC28-1376 and GC28-1377.

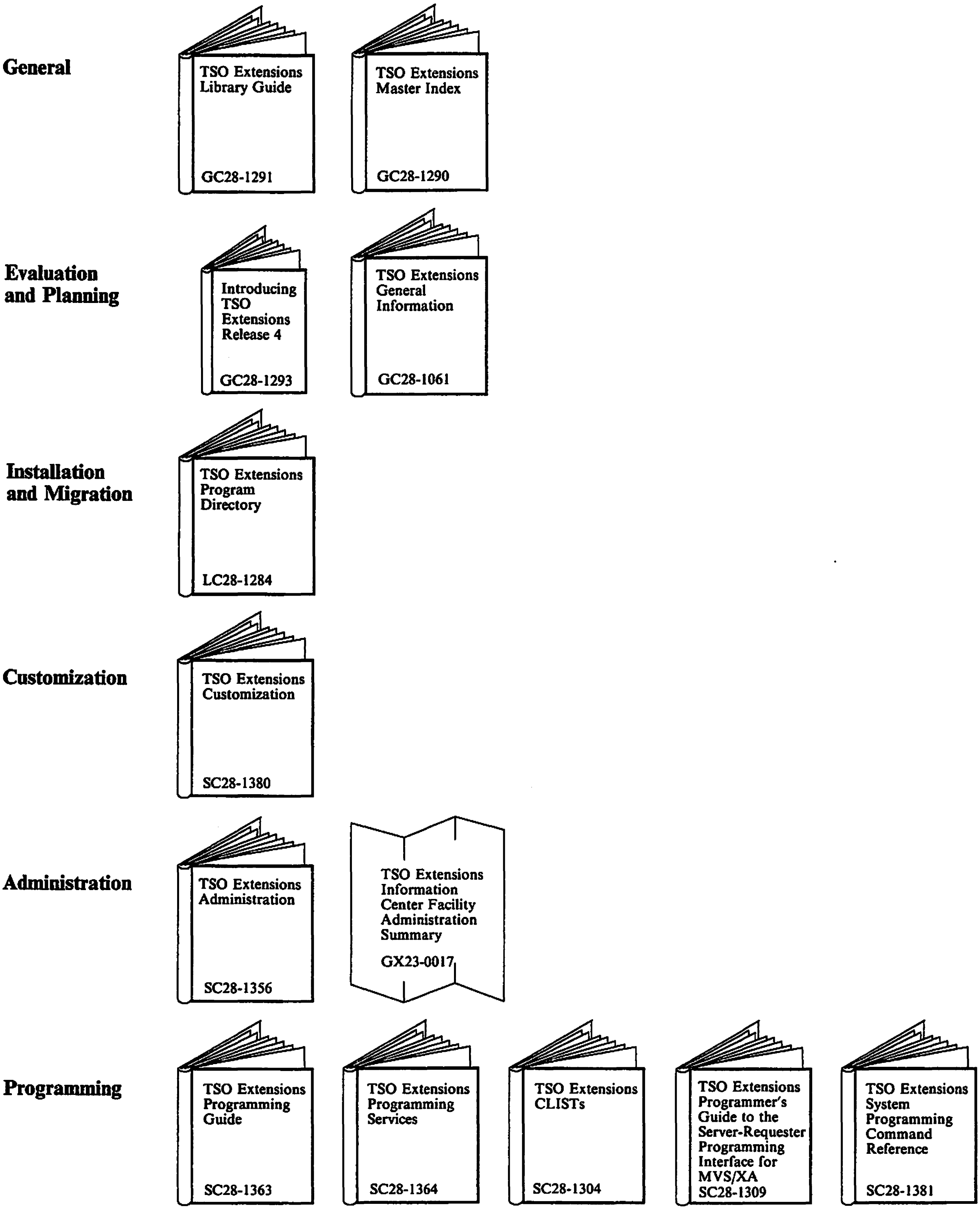
MVS/Extended Architecture System Programming Library: 31-Bit Addressing, GC28-1158

MVS/Extended Architecture Data Areas (microfiche), LYB8-1000.

Do You Have Problems, Comments, or Suggestions?

Your suggestions and ideas can contribute to the quality and the usability of this book. If you have problems using this book, or if you have suggestions for improving it, complete and mail the Reader's Comment Form at the back of the book.

The TSO Extensions Library



General

TSO Extensions Library Guide GC28-1291

TSO Extensions Master Index GC28-1290

Evaluation and Planning

Introducing TSO Extensions Release 4 GC28-1293

TSO Extensions General Information GC28-1061

Installation and Migration

TSO Extensions Program Directory LC28-1284

Customization

TSO Extensions Customization SC28-1380

Administration

TSO Extensions Administration SC28-1356

TSO Extensions Information Center Facility Administration Summary GX23-0017

Programming

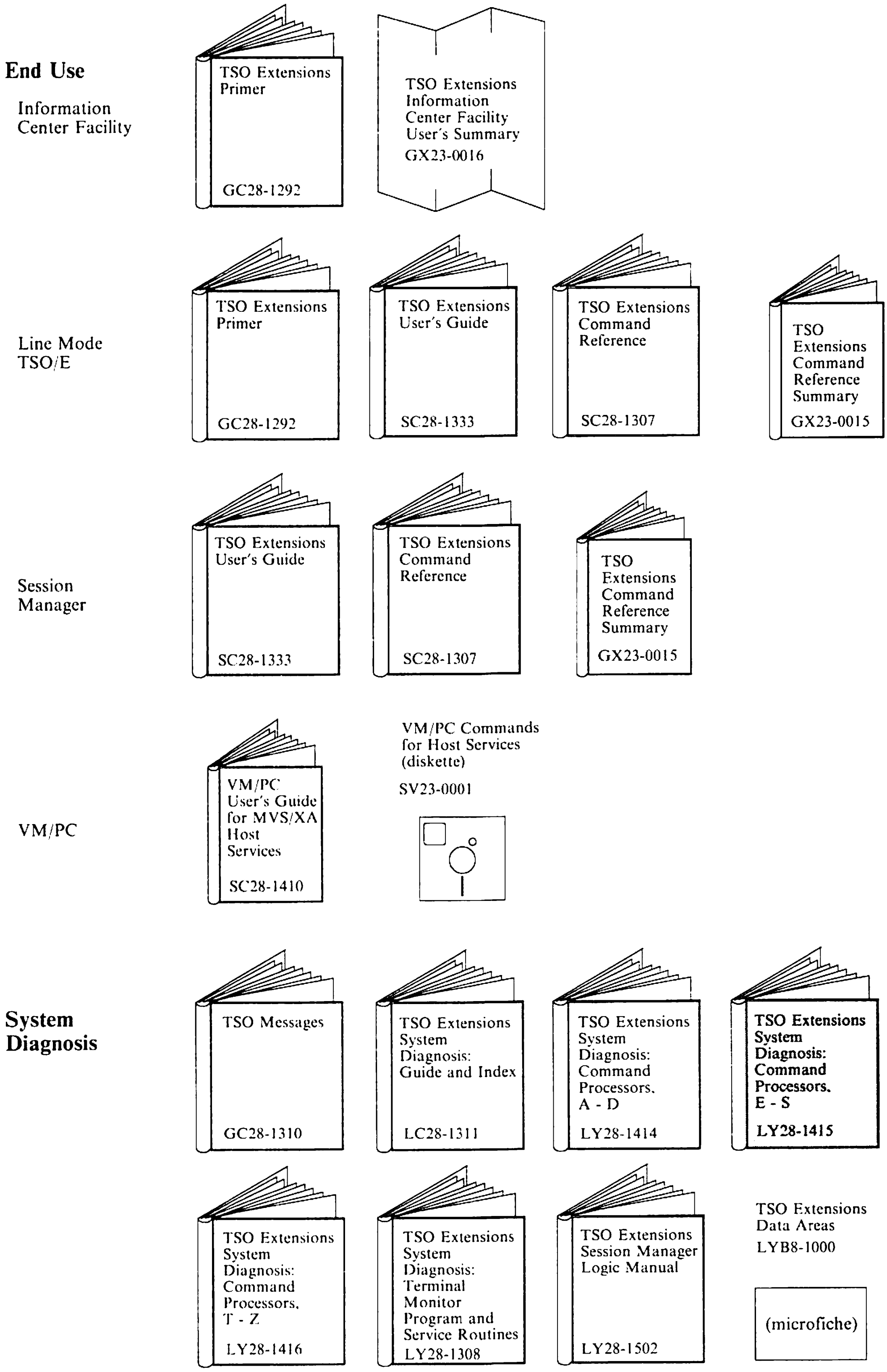
TSO Extensions Programming Guide SC28-1363

TSO Extensions Programming Services SC28-1364

TSO Extensions CLISTs SC28-1304

TSO Extensions Programmer's Guide to the Server-Requester Programming Interface for MVS/XA SC28-1309

TSO Extensions System Programming Command Reference SC28-1381



End Use

Information Center Facility

- TSO Extensions Primer GC28-1292
- TSO Extensions Information Center Facility User's Summary GX23-0016

Line Mode TSO/E

- TSO Extensions Primer GC28-1292
- TSO Extensions User's Guide SC28-1333
- TSO Extensions Command Reference SC28-1307
- TSO Extensions Command Reference Summary GX23-0015

Session Manager

- TSO Extensions User's Guide SC28-1333
- TSO Extensions Command Reference SC28-1307
- TSO Extensions Command Reference Summary GX23-0015

VM/PC

- VM/PC User's Guide for MVS/XA Host Services SC28-1410
- VM/PC Commands for Host Services (diskette) SV23-0001

System Diagnosis

- TSO Messages GC28-1310
- TSO Extensions System Diagnosis: Guide and Index LC28-1311
- TSO Extensions System Diagnosis: Command Processors, A - D LY28-1414
- **TSO Extensions System Diagnosis: Command Processors, E - S LY28-1415**
- TSO Extensions System Diagnosis: Command Processors, T - Z LY28-1416
- TSO Extensions System Diagnosis: Terminal Monitor Program and Service Routines LY28-1308
- TSO Extensions Session Manager Logic Manual LY28-1502
- TSO Extensions Data Areas LYB8-1000 (microfiche)

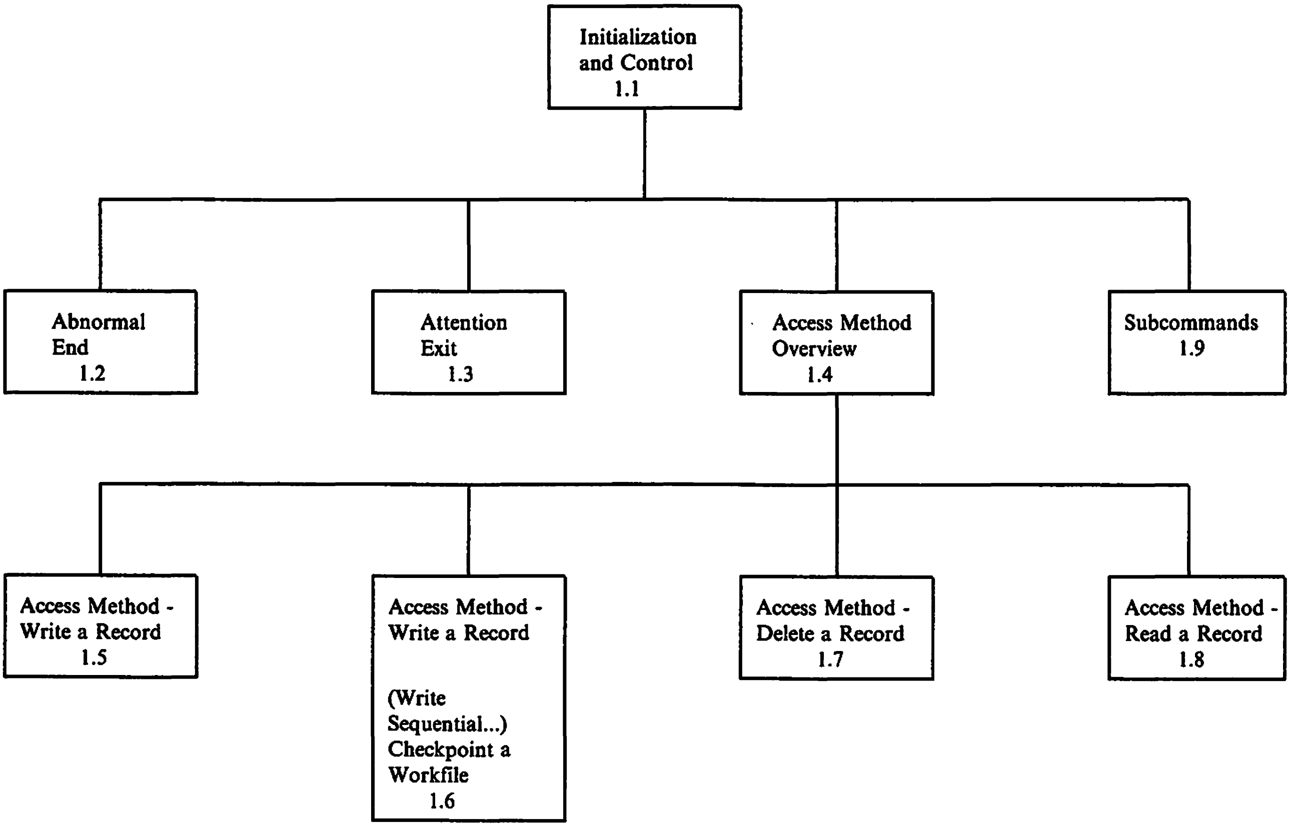
Contents

Figures

The following is a list of the EDIT subcommands and their functions:

ALLOCATE	allocates data sets and filenames.
ATTRIB	builds a list of attributes for a non-VSAM data set.
BOTTOM	moves the line pointer to the last line of the data set.
CHANGE	modifies record text.
CKPOINT	protects input and changes to data set while in EDIT.
COPY	copies lines within a data set.
DELETE	removes records from the data set.
DOWN	moves the line pointer toward the end of the data set.
END	terminates the EDIT command.
EXEC	executes a command procedure.
FIND	locates a character string.
FORMAT	prints out a data set or a portion of a data set in a particular format. Requires IBM COPY, FORMAT, LIST, MERGE program product, or equivalent.
FREE ALL	releases unneeded data sets.
HELP	explains use of EDIT subcommands.
INPUT	accepts new lines of data from the terminal.
INSERT	inserts records into the data set.
Line Insert/Replace/Delete Function	inserts, replaces, or deletes a line of data.
LIST	prints out specific lines of data or the entire data set.
MERGE	merges data sets or parts of data sets into the EDIT utility data set. Requires IBM COPY, FORMAT, LIST, MERGE program product, or equivalent.
MOVE	moves lines within a data set.
PROFILE	redefines the set of options which specify control characters, prompting options, and message options.
RENUM	numbers or renumbers lines of data.
RUN	compiles, loads, and executes the data set.
SAVE	stores data sets on a direct access device.
SCAN	controls syntax checking.
SEND	sends messages to other terminal users and to the system operator.
SUBMIT	causes jobs to be scheduled for batch processing.
TABSET	sets the tab positions for editing.
TOP	moves the line pointer to line zero, if line zero exists; otherwise, moves the pointer in front of the first line of the data set.
UNNUM	removes line numbers from a line-numbered data set.
UP	moves the line pointer toward the beginning of the data set.
VERIFY	causes a display of the line the current line pointer indicates after modification by a subcommand or movement of the current line pointer.

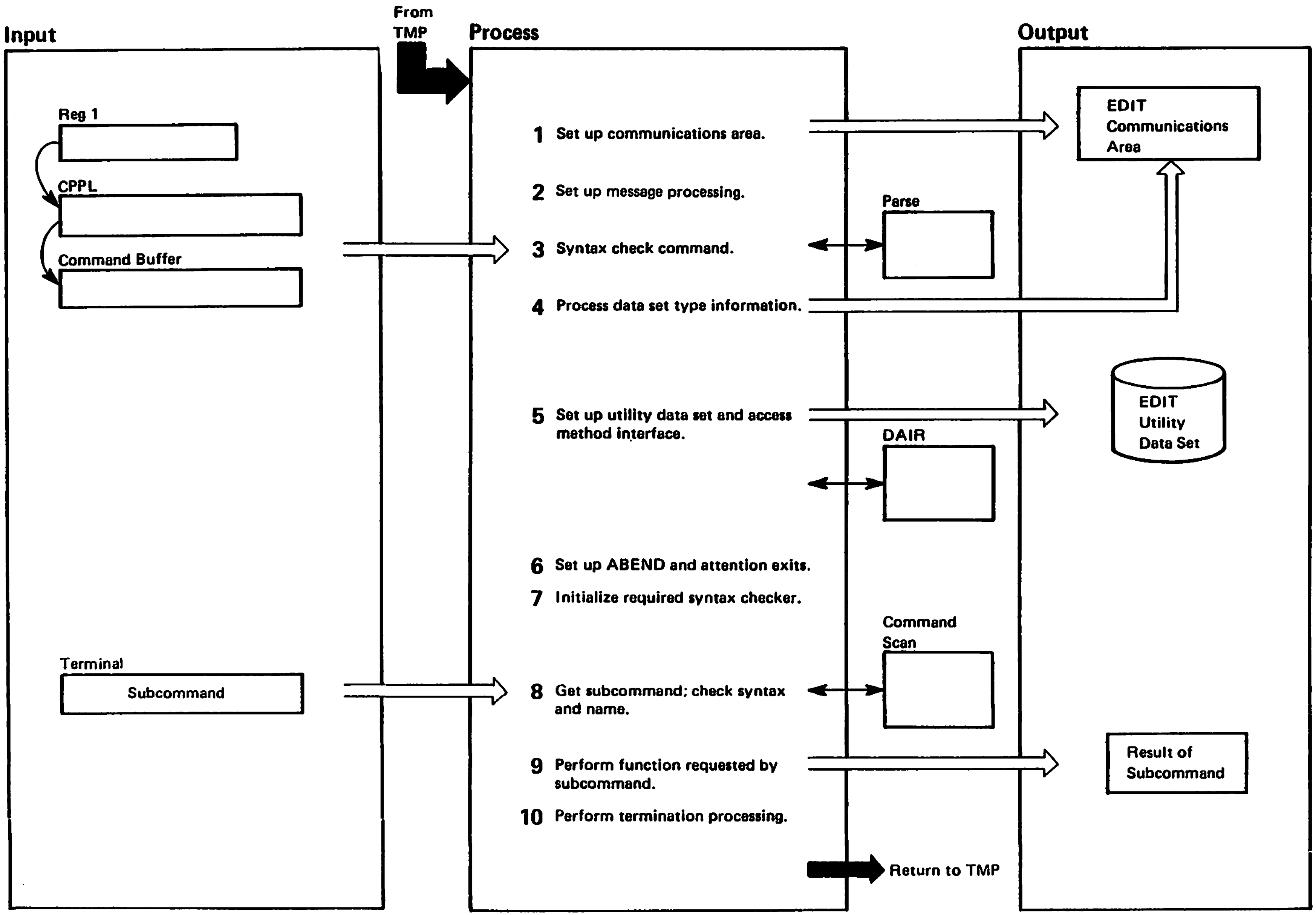
Method of Operation Diagrams


Initialization and Control 1.1
Abnormal End 1.2
Attention Exit 1.3
Access Method Overview 1.4
Subcommands 1.9
Access Method - Write a Record 1.5
Access Method - Write a Record (Write Sequential...) Checkpoint a Workfile 1.6
Access Method - Delete a Record 1.7
Access Method - Read a Record 1.8

Subcommands

1.9	BOTTOM
1.10	CHANGE
1.10.1	CKPOINT
1.11	MOVE/COPY
1.12	DELETE
1.13	DOWN
1.14	END
1.15	FIND
1.16	FORMAT
1.17.	FREE ALL
1.18	INPUT
1.19	INSERT
1.20	LINE I/R/D
1.21	LIST
1.22	MERGE
1.23	PROFILE/SEND/HELP/ALLOCATE
1.24	EXEC
1.25	RENUM
1.26	RUN
1.27	SAVE
1.28	SCAN
1.29	SUBMIT
1.30	TABSET
1.31	TOP
1.32	UNNUM
1.33	UP
1.34	VERIFY

Diagram 1.1. Initialization and Control (Part 1 of 2)



1 Set up communications area.

2 Set up message processing.

3 Syntax check command.

4 Process data set type information.

5 Set up utility data set and access method interface.

6 Set up ABEND and attention exits.

7 Initialize required syntax checker.

8 Get subcommand; check syntax and name.

9 Perform function requested by subcommand.

10 Perform termination processing.

Diagram 1.1. Initialization and Control (Part 2 of 2)

Extended Description **Module** **Label**

1 Set up EDIT communications area (IKJEBECA) to be used as a work area for all EDIT modules. Information about the data set being edited will be saved in this area.

Object Module: IKJEBEIN

2 Set up message processing routine (IKJEBEMS) to be used by all EDIT routines to build and issue messages.

Object Module: IKJEBEIN

3 Use Parse to check the syntax of the EDIT command. If Parse fails, return control to the TMP.

Object Module: IKJEBEIN

4 Process "data set type" information. If there is missing information, use Parse to prompt the user for it.

Initialize the EDIT communication area with information dependent on the data set type.

Object Module: IKJEBEPS

If the user entered the NEW parameter, continue processing. If the user enters the OLD parameter, use DAIR to allocate the data set. Also check the data set attributes to see if they are valid for the data set type and command operands.

Object Module: IKJEBEIN

5 Use IKJEBEUI to set up the utility data set if the data set is new, old and empty, or a new member of a PDS. If the data set is old and not empty, use IKJEBECO to copy the data set into the utility data set. If there is an error in processing, return control to the TMP.

Object Module: IKJEBEIN

6 Set up asynchronous exits for ESTAE (IKJEBEAE) and attention processing (IKJEBEAT).

Object Module: IKJEBEMA

7 Initialize a preprocessor or syntax checker, if required for the data set type being processed. The SCAN subcommand (IKJEBESC) sets up the interface.

Object Module: IKJEBEMA

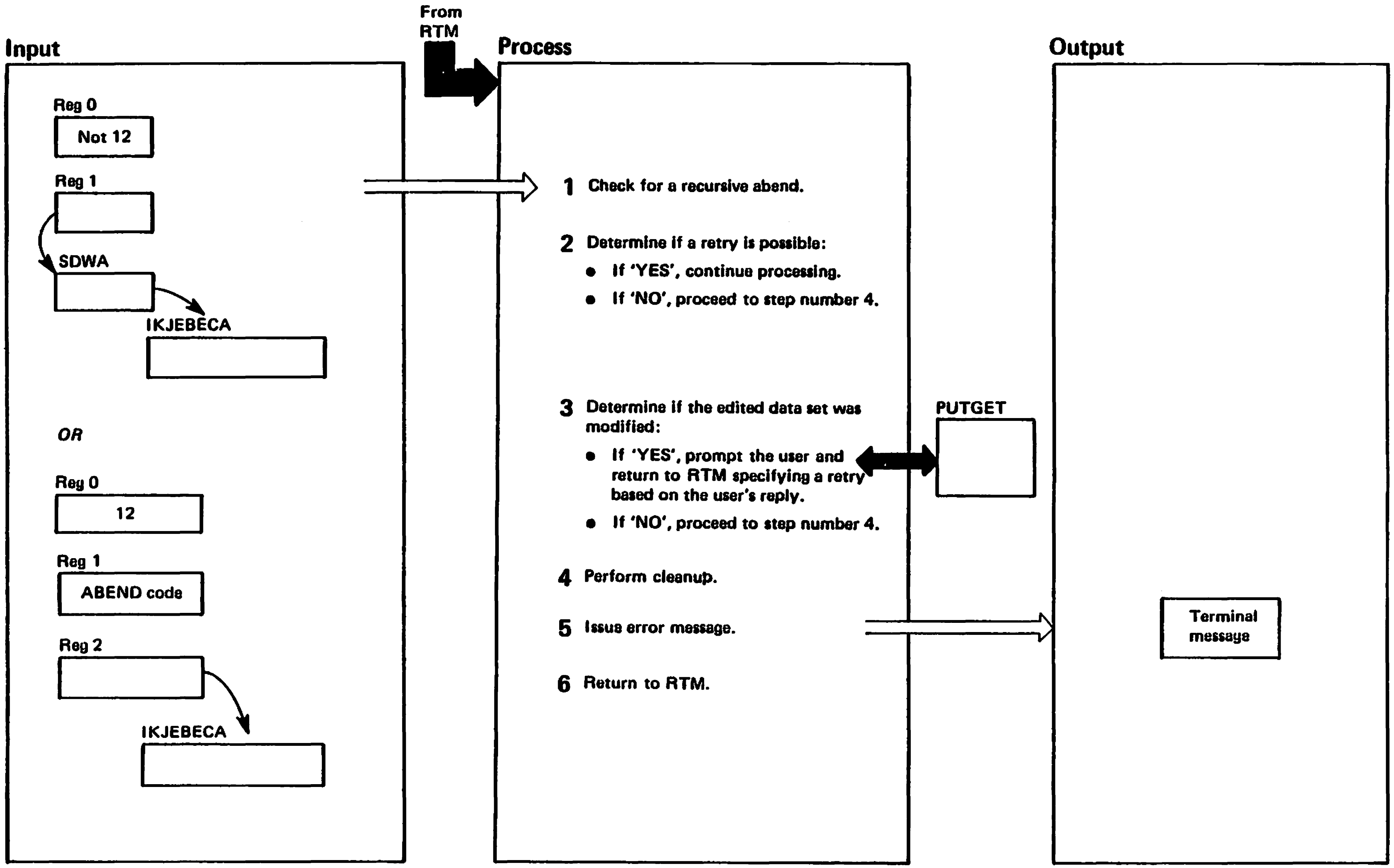
8 Get a subcommand from the terminal and use the command scan routine to check the command syntax. Also check the subcommand name against a list of valid subcommand names (in CSECTs IKJEBMA8 and IKJEBMA9).

9 Perform the function requested by the subcommand being processed.

10 Perform END subcommand processing (IKJEBEEN) if the user enters the END subcommand at a terminal, or if there is an error termination in a component. Delete the input stack and clear all terminal input queues. Return control to the TMP.

Object Modules: IKJEBEAE, IKJEBEAT, IKJEBEEN, IKJEBEMA, IKJEBEMS, IKJEBEIN, IKJEBESC, IKJEBEUI, IKJEBEPS

Diagram 1.2. Abnormal End (ESTAE Routine) Processing (Part 1 of 2)



1 Check for a recursive abend.

2 Determine if a retry is possible:

- If 'YES', continue processing.
- If 'NO', proceed to step number 4.

3 Determine if the edited data set was modified:

- If 'YES', prompt the user and return to RTM specifying a retry based on the user's reply.
- If 'NO', proceed to step number 4.

4 Perform cleanup.

5 Issue error message.

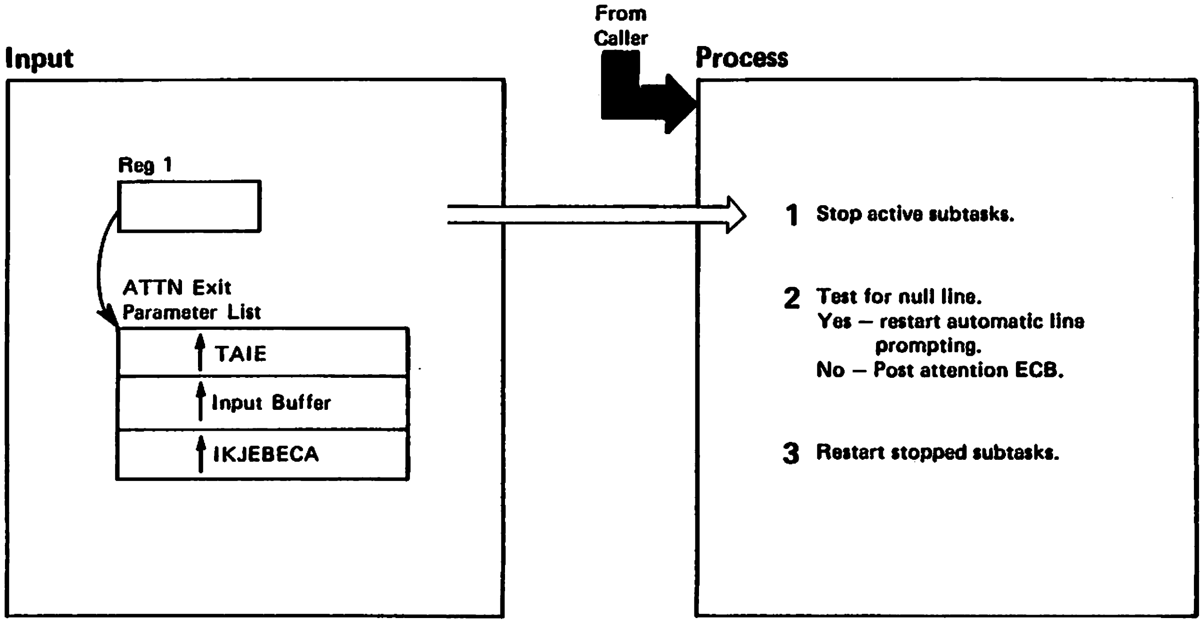
6 Return to RTM.

Diagram 1.2. Abnormal End (ESTAE Routine) Processing (Part 2 of 2)

	Extended Description	Module	Label
1	Check for the recursive abend indicator in the EDIT communications area (IKJEBECA). If 'NO', return control to RTM, letting the ABEND continue. If 'OFF', continue processing.		
2	Determine if a retry is possible. If 'YES', continue processing. If 'NO', perform cleanup (Step number 4).		
3	Determine if the EDIT data set was modified. If 'YES', prompt the user and return to RTM, specifying a retry based on the user's reply. If 'NO', perform cleanup (Step number 4).		
4	Perform cleanup (IKJEBEMA, entry point IKJEBMA2) by stopping automatic line prompting and deleting the input stack.		
5	Issue an error message indicating the type of abend.		
6	Return to RTM, indicating a retry if it is possible, or else letting the ABEND continue.		

Object Modules: IKJEBEAE, IKJEBEEN, IKJEBEMA

Diagram 1.3. Attention Exit Processing (Part 1 of 2)



1 Stop active subtasks.

2 Test for null line.
Yes – restart automatic line prompting.
No – Post attention ECB.

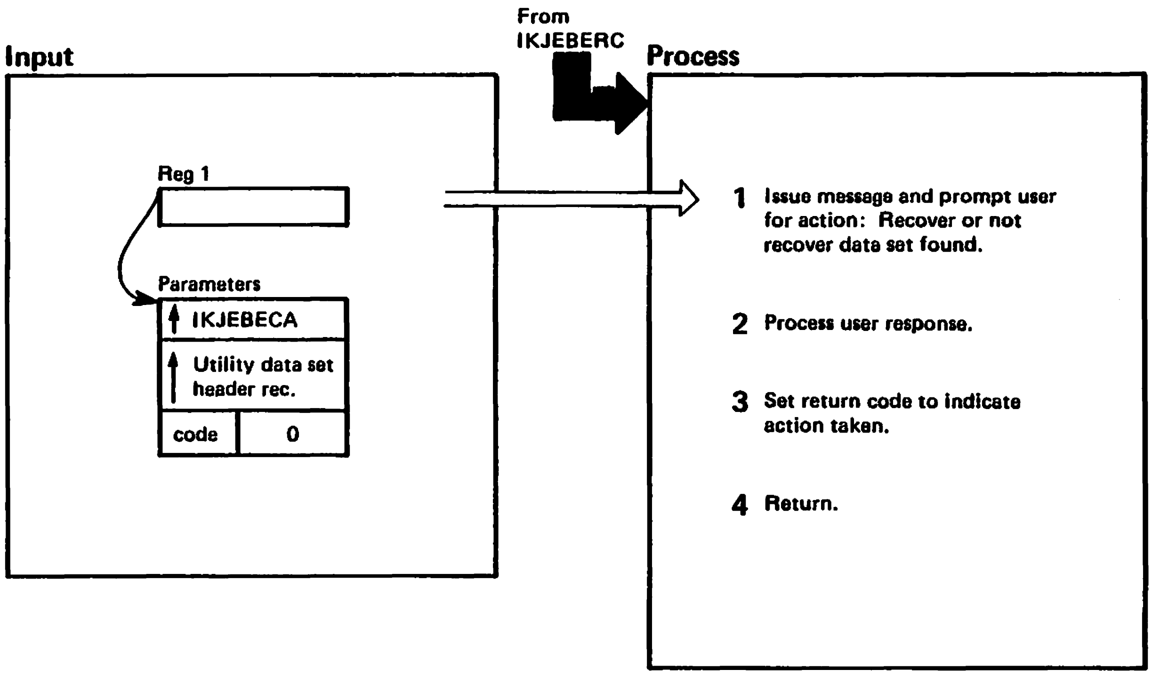
3 Restart stopped subtasks.

Diagram 1.3. Attention Exit Processing (Part 2 of 2)

	Extended Description	Module	Label
1	Stop all active subtasks.		
2	Check the terminal attention interrupt exit (TAIE) to see if the user entered a null line.		
	Yes - restart the automatic line prompting.		
	No - retrieve (PUTGET) the input line. Post the attention ECB.		
3	Restart all subtasks that were stopped. Return control to the caller.		

Object Modules: IKJEBEAT, IKJEBEMA

Diagram 1.3.1. Automatic Recovery Routine Processing (Part 1 of 2)



1 Issue message and prompt user for action: Recover or not recover data set found.

2 Process user response.

3 Set return code to indicate action taken.

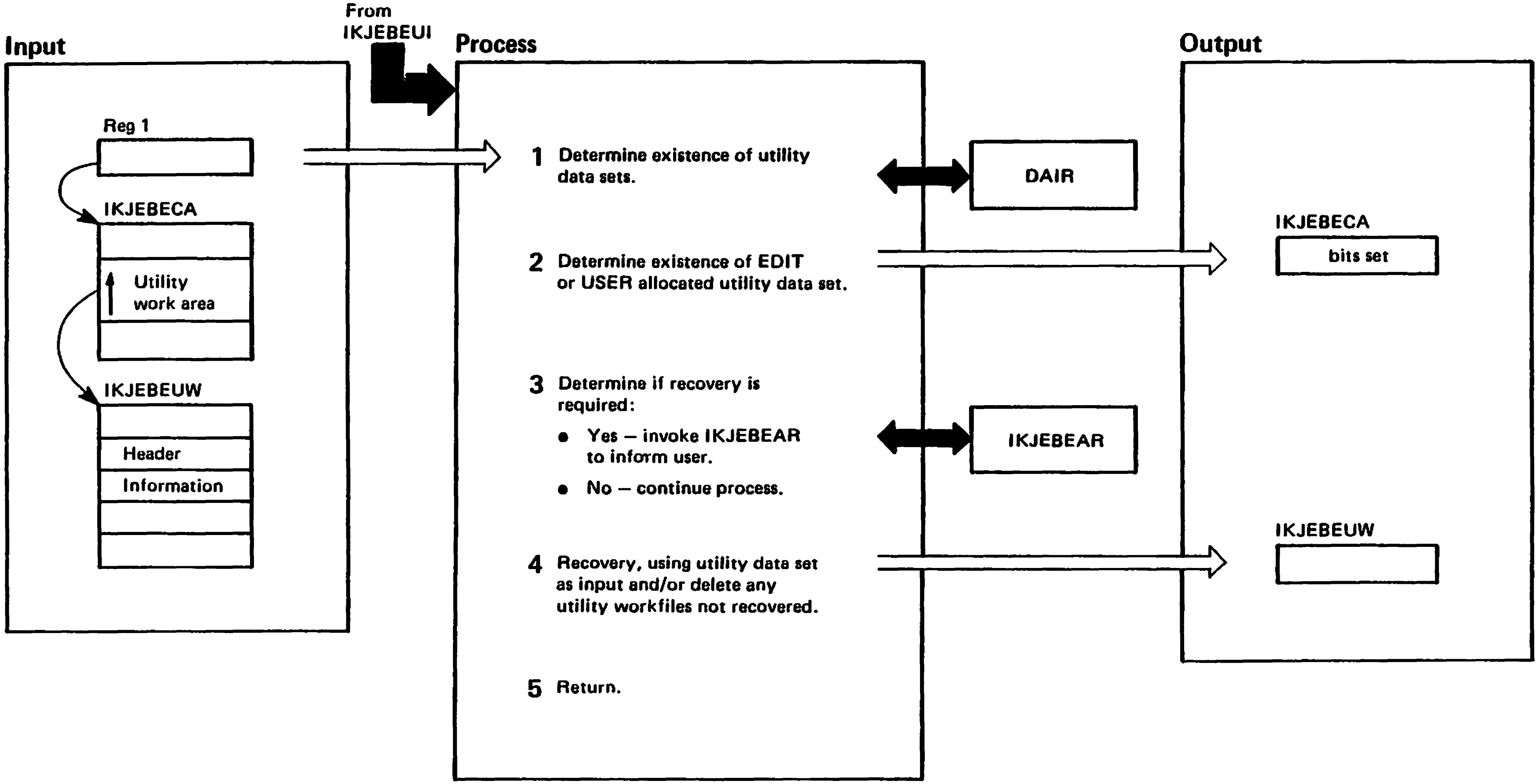
4 Return.

Diagram 1.3.1. Automatic Recovery Routine Processing (Part 2 of 2)

	Extended Description	Module	Label
1	Issue message to tell the user there is a recoverable EDIT workfile. Prompt the user for a decision about recovering the data set found.		
2	Process the user's response. If the user replies 'YES': (a) Build a buffer to reflect information in the workfile header. (b) Issue a message to the user, indicating the options in effect for this edit session. If the user replies 'NO', no function is performed.		
3	If the user replies 'YES', set return code to 4. If the user replies 'NO', set return code to 0.		
4	Return to caller.		

Object Module: IKJEBEAR

Diagram 1.3.2 Recovery/Cleanup Routine Processing (Part 1 of 2)



1 Determine existence of utility data sets.

2 Determine existence of EDIT or USER allocated utility data set.

3 Determine if recovery is required:

- Yes – invoke IKJEBEAR to inform user.
- No – continue process.

4 Recovery, using utility data set as input and/or delete any utility workfiles not recovered.

5 Return.

Diagram 1.3.2 Recovery/Cleanup Routine Processing (Part 2 of 2)

	Extended Description	Module	Label
1	Determine if either userid.EDITUTL1 or userid.EDITUTL2 exists by trying to allocate them as 'OLD'.		
2	Determine if EDIT or the user allocated the utility data set(s) which exists. Set indicator in IKJEBECA.		
3	If a recoverable utility data set exists, and if the RECOVER operand was not specified or the data set name and options do not match, invoke IKJEBEAR.		
4	Recover the utility data set. Update the utility data set control blocks in the utility work area (IKJEBEUW), and/or delete any EDIT utility data sets not being recovered.		
5	Return to caller.		

Object Module: IKJEBERC

Diagram 1.4. Access Method Overview (Part 1 of 2)

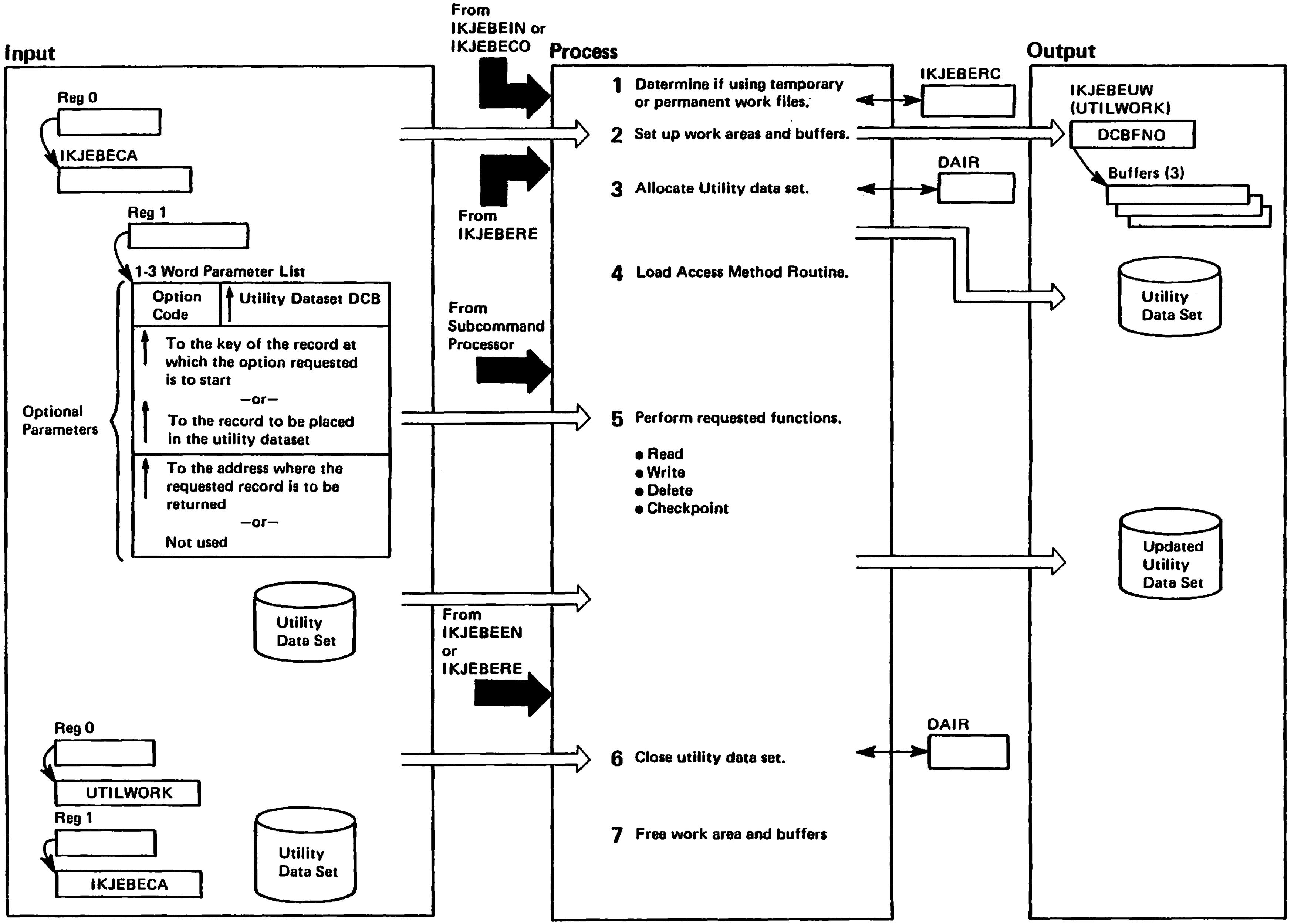
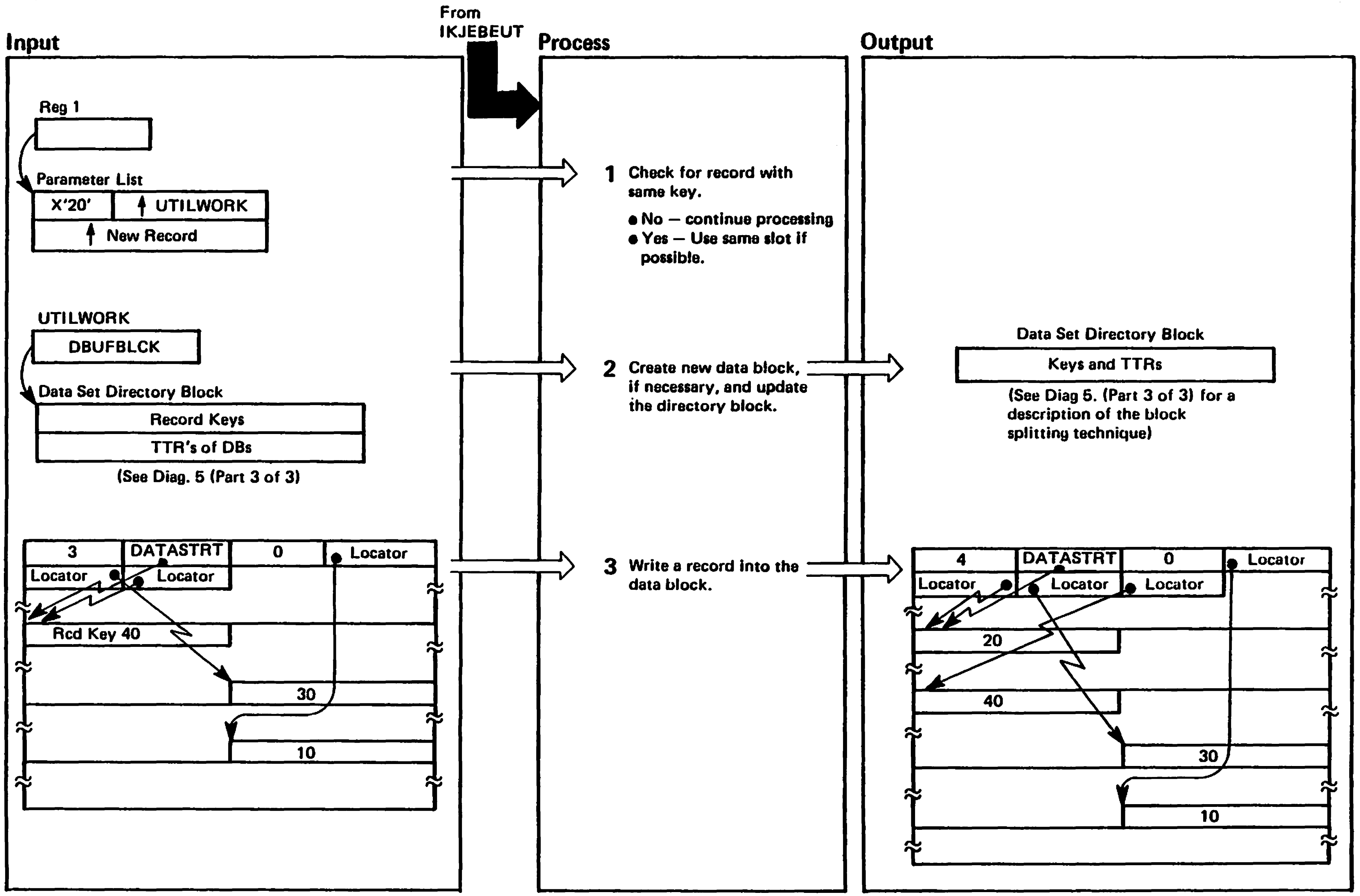

Input
Reg 0
IKJEBECA
Reg 1
1-3 Word Parameter List
Option Code
Utility Dataset DCB
Optional Parameters
To the key of the record at which the option requested is to start
–or–
To the record to be placed in the utility dataset
To the address where the requested record is to be returned
–or–
Not used
Utility Data Set
Reg 0
UTILWORK
Reg 1
IKJEBECA
Utility Data Set
From IKJEBEIN or IKJEBECO
From IKJEBERE
From Subcommand Processor
From IKJEBEEN or IKJEBERE
Process
1 Determine if using temporary or permanent work files.
2 Set up work areas and buffers.
3 Allocate Utility data set.
4 Load Access Method Routine.
5 Perform requested functions.
● Read
● Write
● Delete
● Checkpoint
6 Close utility data set.
7 Free work area and buffers
IKJEBERC
DAIR
DAIR
Output
IKJEBEUW (UTILWORK)
DCBFNO
Buffers (3)
Utility Data Set
Updated Utility Data Set

Diagram 1.4. Access Method Overview (Part 2 of 2)

	Extended Description	Module	Label
1	Determine if the utility data sets are to be permanent or temporary. If permanent, link to IKJEBERC.		
2	Set up (IKJEBEUI) the access method work area, UTILWORK (IKJEBEUW), which is used by all of the access method routines. Also set up three buffers which are used when accessing the utility data set.		
3	Using DAIR, allocate (IKJEBEUI) a new utility data set. The data set is either a system-generated temporary data set (&EDIT or &EDIT2) or a permanent data set (userid.EDITUTL1 or userid.EDITUTL2) The size of the data set is determined by whether the data set being edited is new or old. If the user LOGON procedure predefined an EDIT utility data set, that data set will be used.		
4	Load (IKJEBEUI) the access method routines (IKJEBEAA). The access method routines will remain for the duration of the terminal session.		
5	Perform the requested function (read, write, or delete) (IKJEBEAA). The function is requested through the access method interface (IKJEBEUT) by each subcommand requiring an update of or information about the EDIT utility data set.		
6	Close the EDIT utility data set and, if END processing is not in progress, free the data set (IKJEBEEX).		
7	Delete the access method (IKJEBEAA), and free the work area (UTILWORK) and buffers (IKJEBEEX).		

Object Modules: IKJEBEUI, IKJEBEAA, IKJEBERC, IKJEBEUT, IKJEBEEX

Diagram 1.5. Access Method – Write a Record (Part 1 of 4)



1 Check for record with same key.

- ● No – continue processing
- ● Yes – Use same slot if possible.

2 Create new data block, if necessary, and update the directory block.

3 Write a record into the data block.

Diagram 1.5. Access Method – Write a Record (Part 2 of 4)

Extended Description **Module** **Label**

1 Check to see if there is a record with the same key value (IKJEBELO).

No - continue processing.

Yes - If the length of the old record is greater than or equal to the length of the new record, write the new record in place of the old one. Return control to IKJEBEUT.

If the length of the old record is smaller than the length of the new record, delete (IKJEBEDL) the old record. Update the data block fields to indicate a record was deleted by removing the record locator, and updating the number of records and the recoverable space in the data block. Continue processing with Step 2.

2 Check the data block to see if there is space for the new record. If the record logically belongs there, but there is not enough space, split the data block into two data blocks. Each data block contains half the records in the old block. Update (IKJEBEDU) the directory block pointing to the data blocks. For additional information see Diagram 1.5 (Part 3 of 4).

3 Write a record into the data block and update the following data block field descriptions:

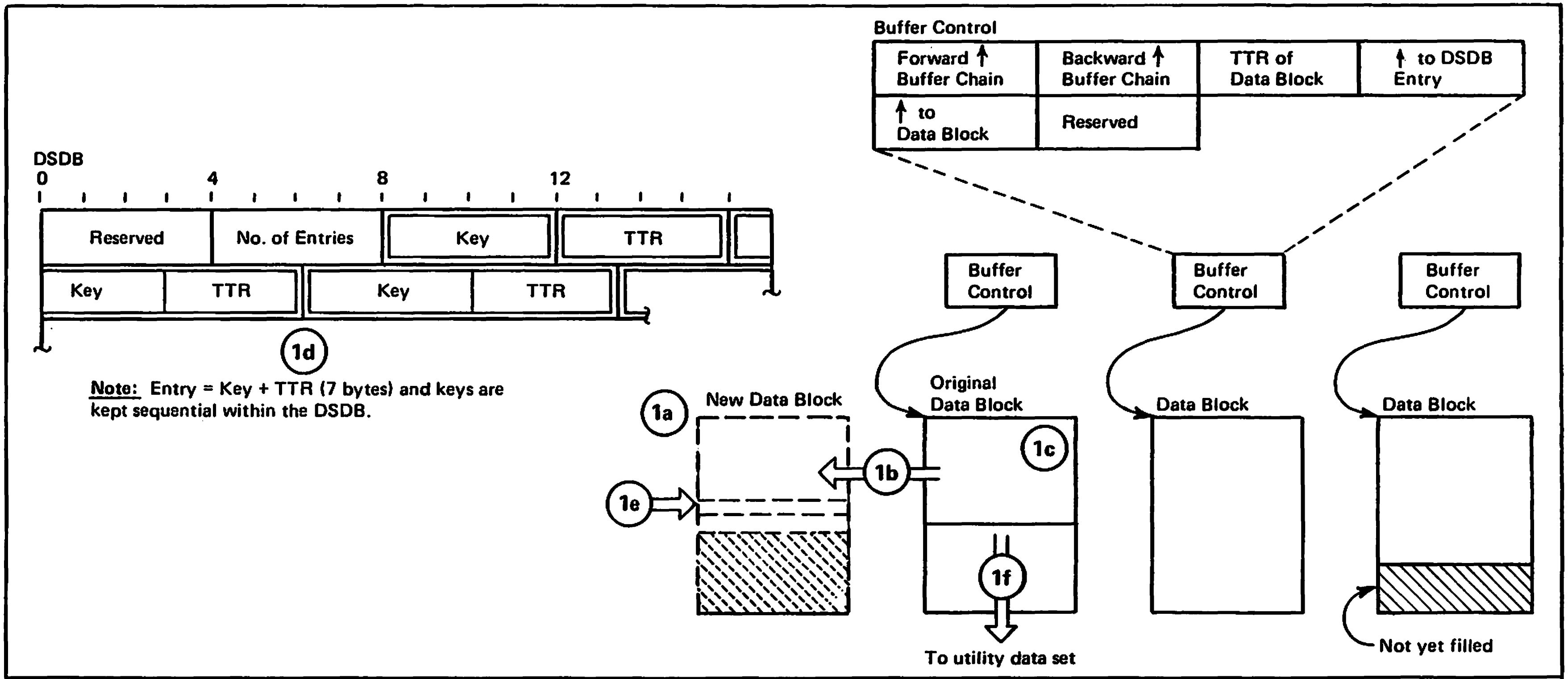
- the number of records in the data block
- the locator is set to point at the new record.
- the amount of recoverable space in the data block.

Return control to IKJEBEUT with a return code in register 15.

Object Module: IKJEBEAA

CSECTS: IKJEBEDL, IKJEBEDU, IKJEBELO

Diagram 1.5. Access Method – Write a Record (Part 3 of 4)



Note: Entry = Key + TTR (7 bytes) and keys are kept sequential within the DSDB.

1 If a new record is to be written and the appropriate data block is full:

a. Space is obtained for a new data block.

b. Half of the records from the original block are written into the new block.

c. The moved records are deleted from the original block.

d. The data set directory block (DSDB) pointing to the data blocks is updated.

e. The new record is written into the new data block.

f. The original block is written out to the utility data set.

g. The space of the original data block is deleted.

2 If a new record is to be added and a new data block must be added but the data set directory block (DSDB) is full:

a. Space equal to the current size of the DSDB plus the original size of the DSDB is obtained.

b. The contents of the current DSDB are copied to the new DSDB.

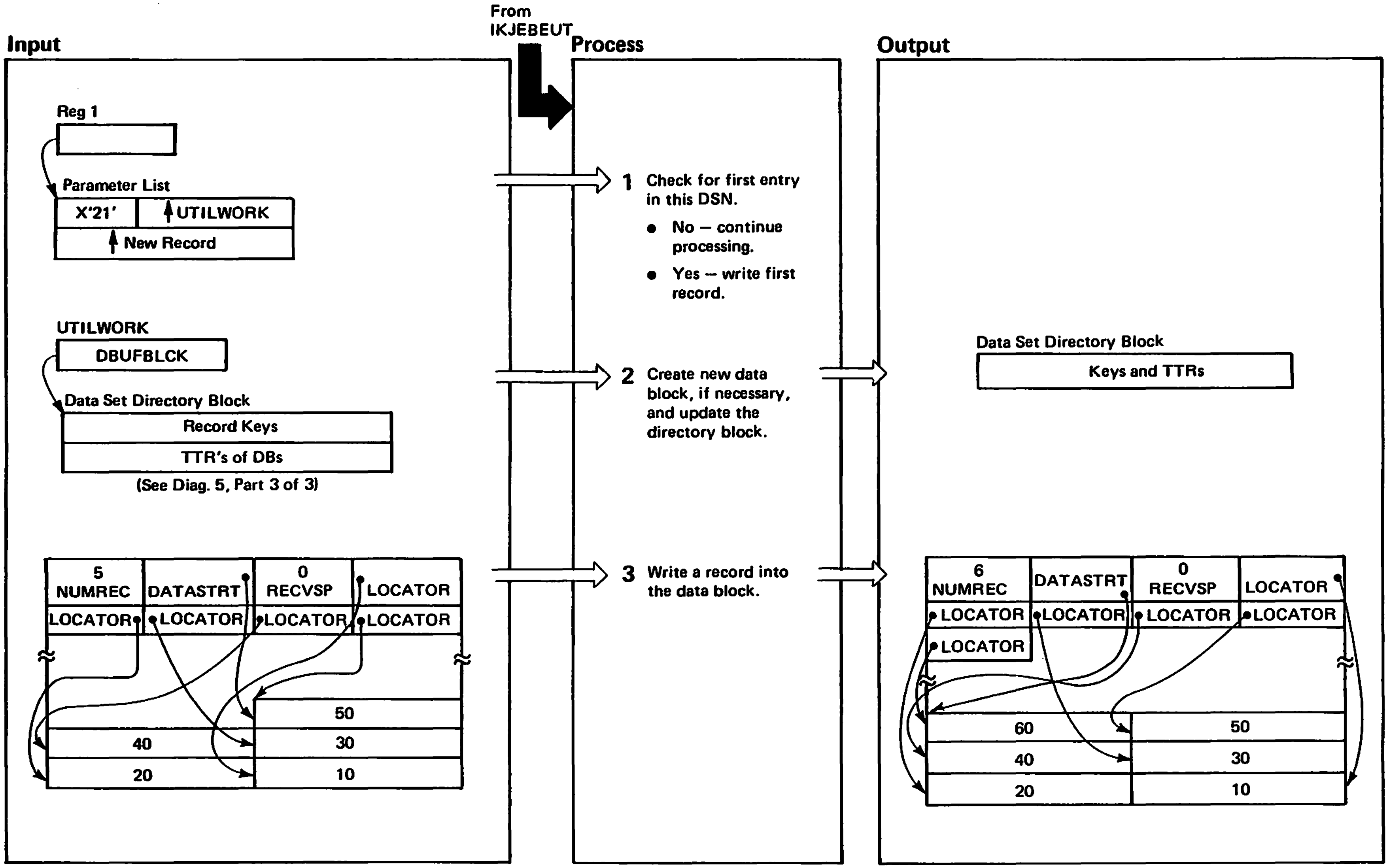
c. The space occupied by the current DSDB is freed; the new DSDB is now used.

d. Normal processing continues.

Diagram 1.5. Access Method — Write a Record (Part 4 of 4)

	Extended Description	Module	Label
1	If a new record is to be written and the appropriate data block is full: a. Obtain space for a new data block. b. Write half of the records from the original block into the new block. c. Delete the moved records from the original block. d. Update the data set directory block (DSDB) pointing to the data blocks. e. Write the new record into the new data block. f. Write out the original block to the utility data set. g. Delete the space of the original data block.		
2	If a new data block must be added for a new record, but the data set directory block (DSDB) is full: a. Obtain space equal to the current size of the DSDB plus the original size of the DSDB. b. Copy the contents of the current DSDB to the new DSDB. c. Free the space occupied by the current DSDB; use the new DSDB. d. Continue normal processing.		

Diagram 1.6. Access Method - Write a Record (Write Sequential Operation into a New Utility Data Set) (Part 1 of 2)



1 Check for first entry in this DSN.

- No – continue processing.
- Yes – write first record.

2 Create new data block, if necessary, and update the directory block.

3 Write a record into the data block.

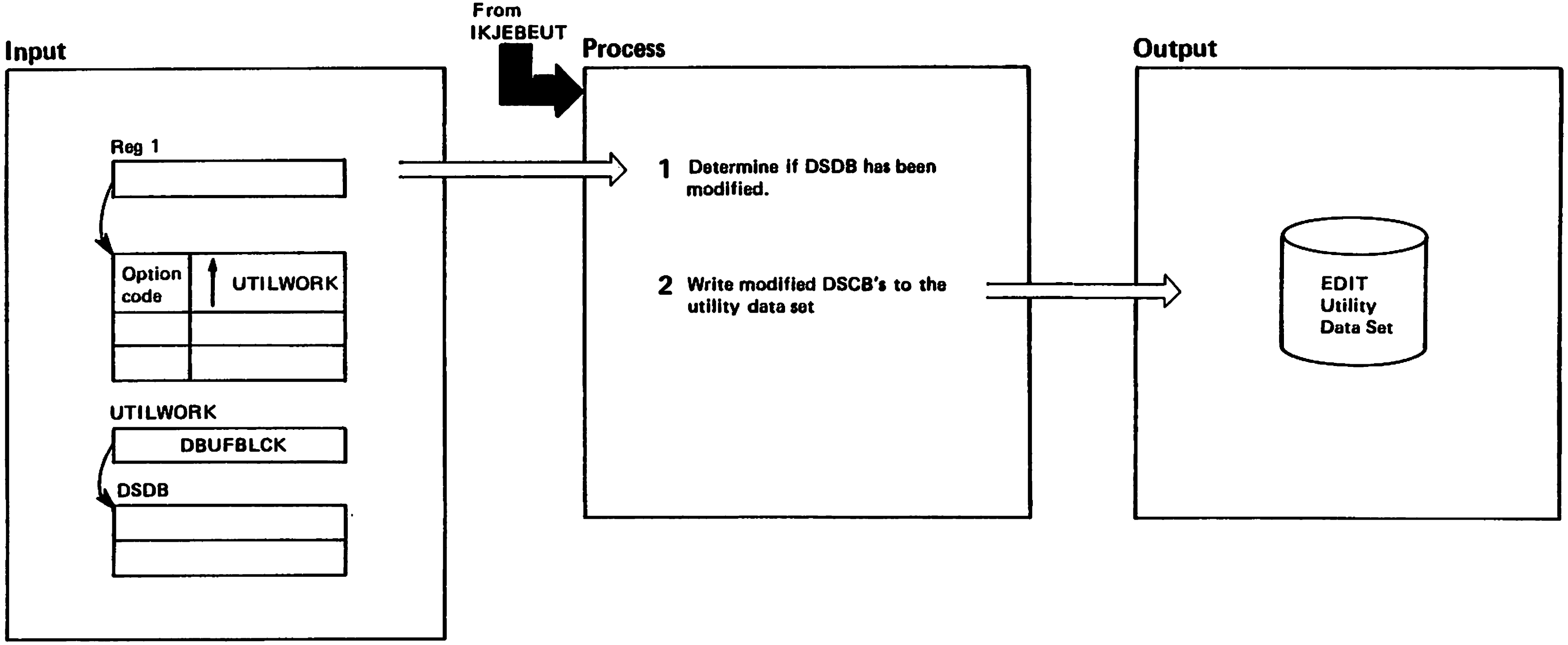
Diagram 1.6. Access Method - Write a Record (Write Sequential Operation into a New Utility Data Set) (Part 2 of 2)

	Extended Description	Module	Label
1	Check to see if this is the first record to be written into this dataset. NO - Process using current block. YES - Indicate this DSN has been used. Write first record (IKJEBEMV).		
2	Check block to see if there is sufficient space for the current record. NO - Check for outstanding write operation. YES - Issue WAIT (IKJEBEWA). NO - Issue Write on filled block. YES - Insert record.		
3	Update control blocks. • Number of records in block. • Update the directory (IKJEBEDU).		

Object Module: IKJEBEAA

CSECTS: IKJEBEDU, IKJEBEMV, IKJEBEWA

Diagram 1.6.1. Access Method - Checkpoint a Workfile (Part 1 of 2)



1 Determine if DSDB has been modified.

2 Write modified DSCB's to the utility data set

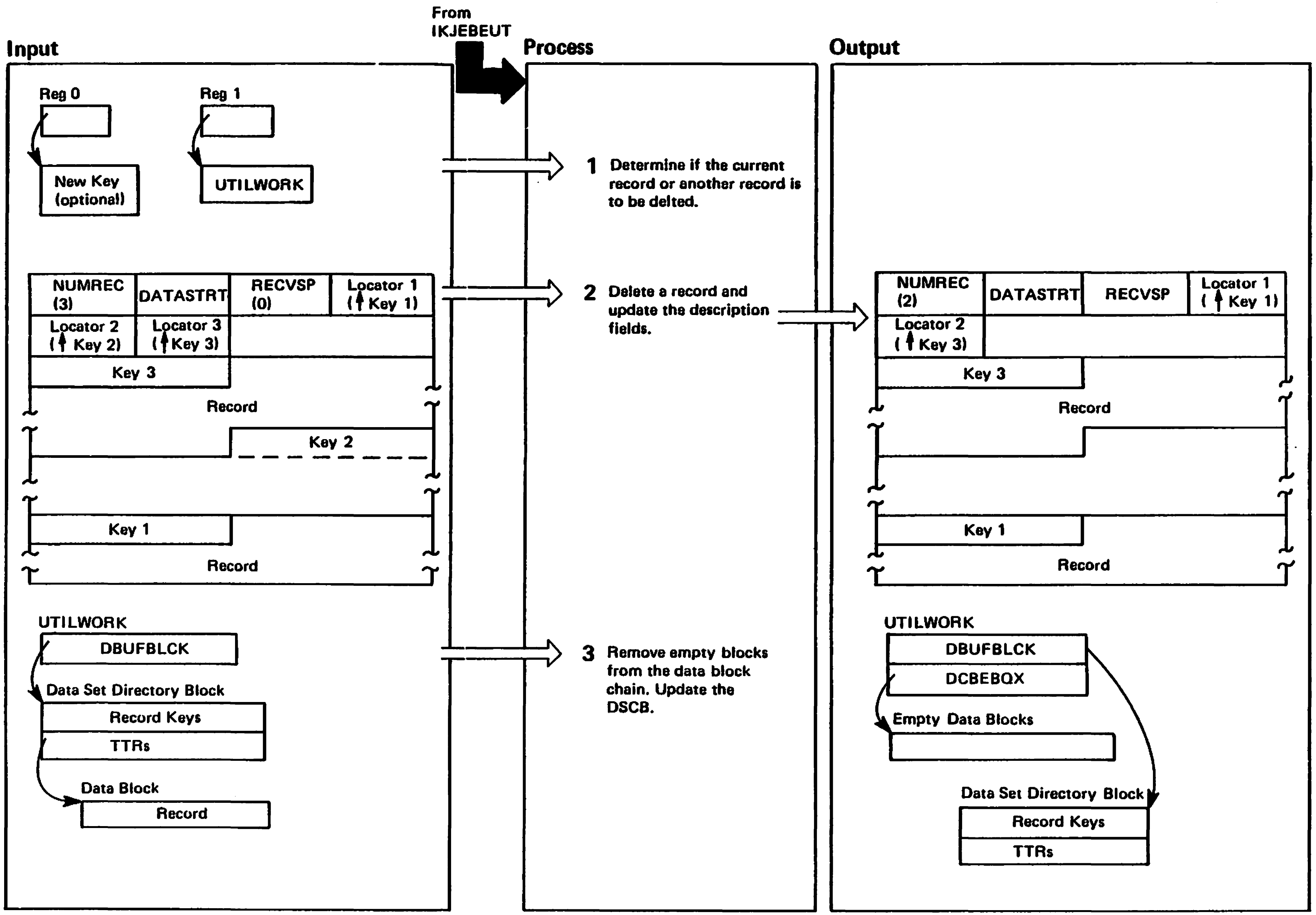
Diagram 1.6.1. Access Method - Checkpoint a Workfile (Part 2 of 2)

	Extended Description	Module	Label
1	Determine which data blocks have been modified.		
2	Write modified data blocks to the direct access device containing the utility data set.		

Object Module: IKJEBEAA

CSECT: IKJEBEWB, IKJEBEWM

Diagram 1.7. Access Method - Delete a Record (Part 1 of 2)



1 Determine if the current record or another record is to be delted.

2 Delete a record and update the description fields.

3 Remove empty blocks from the data block chain. Update the DSCB.

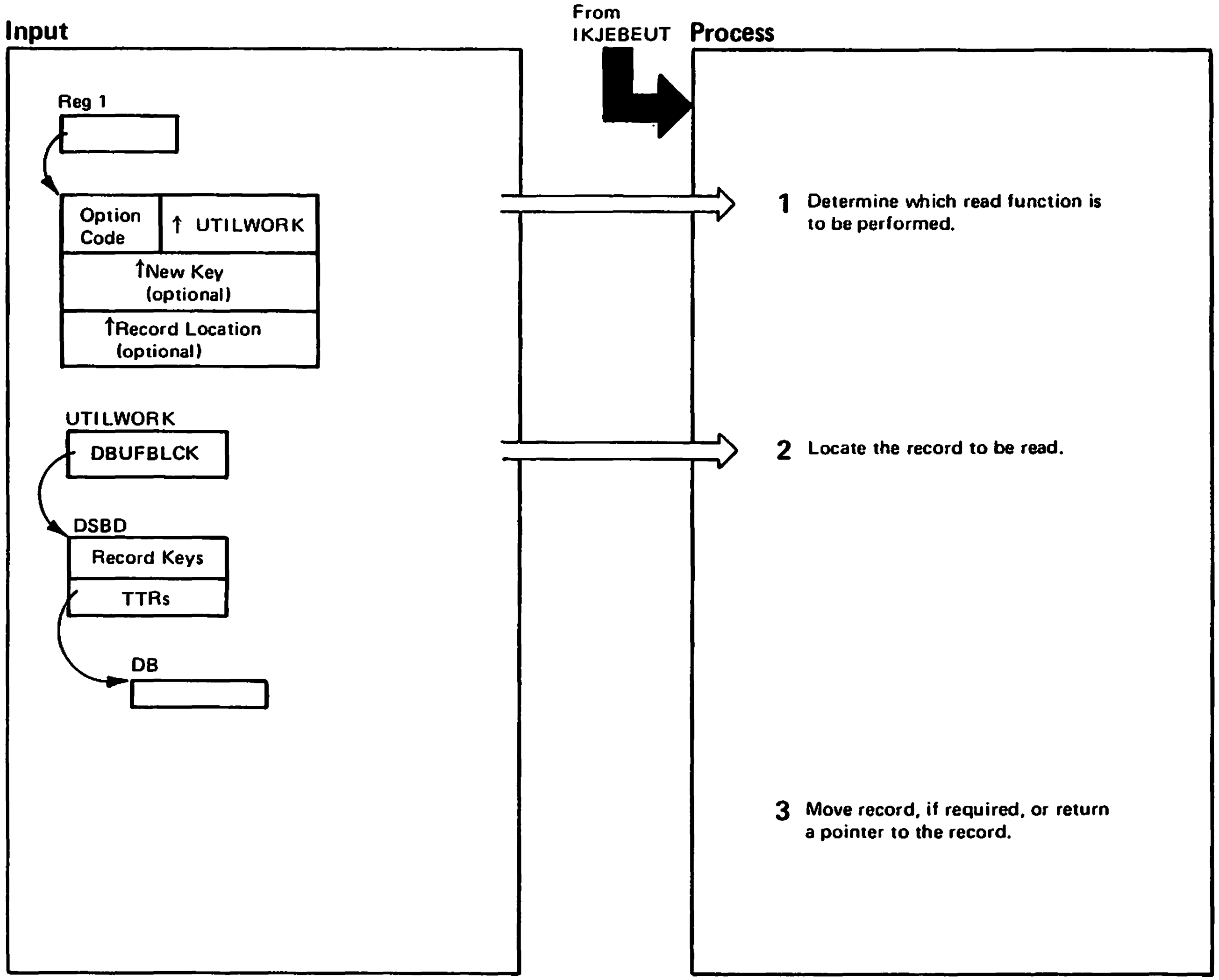
Diagram 1.7. Access Method - Delete a Record (Part 2 of 2)

	Extended Description	Module	Label
1	Determine if it is the current record or another record that is to be deleted (IKJEBEDR) by checking to see if a key was provided. If no key was provided, the current record is to be deleted. If a key was provided, save the current record key and locate the new record key (IKJEBELO). If the new key cannot be located, return control to IKJEBEUT with a return code of 4.		
2	Delete (IKJEBEDL) a record by updating the data block description fields as follows: • Reduce the record count in the data block. • Add the empty space to the recoverable space in the data block. • Remove record locater for the deleted record and adjust the locaters to fill any empty positions.		
3	Remove any empty data blocks from the chain and place them on the empty queue. These are held in reserve for future data block splits. Return control to IKJEBEUT with a return code in register 15.		

Object Module: IKJEBEAA

CSECTS: IKJEBEDL, IKJEBEDR, IKJEBELO

Diagram 1.8. Access Method - Read a Record (Part 1 of 2)



1 Determine which read function is to be performed.

2 Locate the record to be read.

3 Move record, if required, or return a pointer to the record.

Diagram 1.8. Access Method - Read a Record (Part 2 of 2)

	Extended Description	Module	Label
1	Check the parameter list for the requested function (IKJEBERR): read first, read last, read next, read previous, or read the record specified by the key passed in the parameter list.		
2	Locate (IKJEBELO) the requested record by searching the directory blocks (IKJEBEDS) and moving the data block containing the record into storage.		
3	Check the parameter list for a pointer to a subcommand buffer area. If there is a pointer, move (IKJEBEMV) the record to the buffer. Otherwise, return a pointer to the record in register 1. If an error occurred, register 15 will contain a 4 (record not found) or a 12 (I/O error). Return control to IKJEBEUT.		

Object Module: IKJEBEAA

CSECTS: IKJEBEDS, IKJEBELO, IKJEBEMV

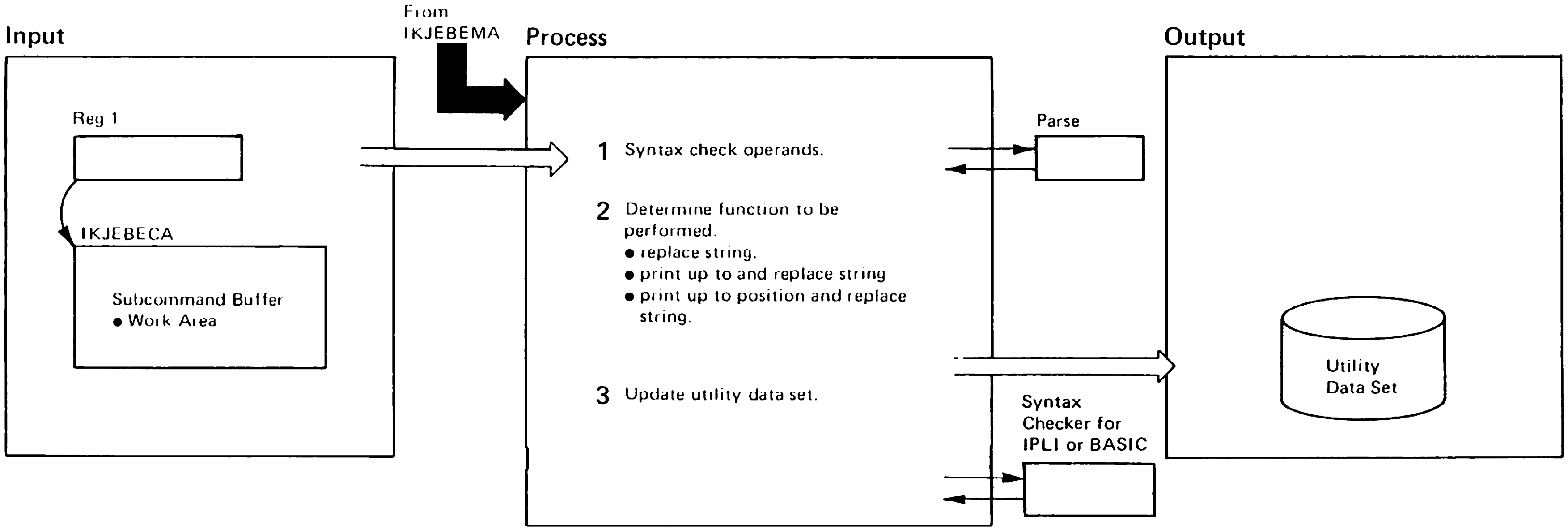
Diagram 1.9. BOTTOM Subcommand Processing (Part 1 of 1)

(No drawing with text.)

	Extended Description	Module	Label
1	Pass control to the BOTTOM subcommand processor (IKJEBEBO) from the controller routine (IKJEBEMA).		
2	Request the EDIT access method interface routine (IKJEBEUT) to locate and read the last record. Set the current line pointer to the key of the final record and turn on the "line to be verified" switch. If the data set is empty (return code 4), send message IKJ52501I to the user. Set the current line pointer to zero.		
3	Return control to IKJEBEMA.		

Object Modules: IKJEBEBO, IKJEBEUT

Diagram 1.10. CHANGE Subcommand Processing (Part 1 of 2)



1 Syntax check operands.

2 Determine function to be performed.
- replace string.
- print up to and replace string
- print up to position and replace string.

3 Update utility data set.

Diagram 1.10. CHANGE Subcommand Processing (Part 2 of 2)

Extended Description **Module** **Label**

1 Use Parse to check the syntax of the operands. Check the first two operands for an '*' or a line number reference. If a line number reference is specified, check the validity of the operands. Read in the first record to be processed; terminate and return control to IKJEBEMA if there are no lines in the range (IKJEBECH).

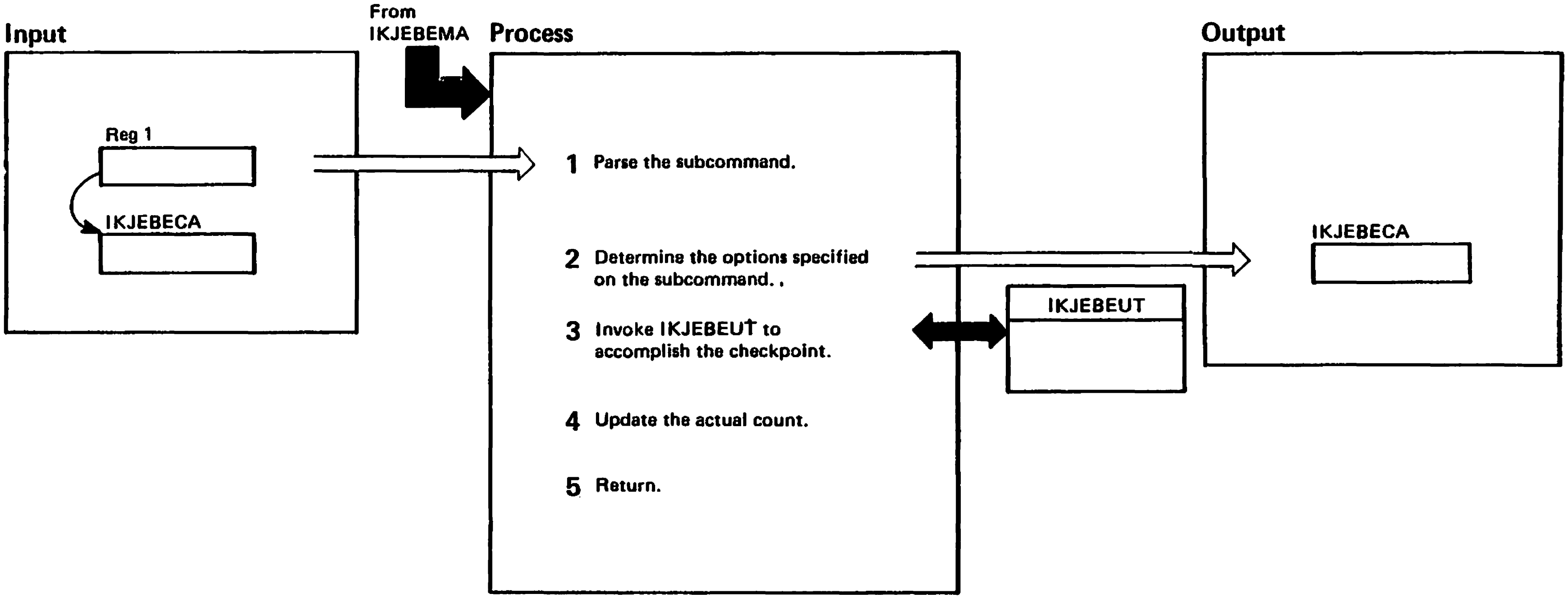
2 Determine what function has been requested (IKJEBECH).

- Replace string - format the character string (IKJEBELE), and scan the record for a character string match (IKJEBESE). If a match is not found, issue an error message. Replace the old string with the new string. If a line overflow occurs, issue an error message. If the overflow occurred for a text data set without line numbers (NONUM/TEXT), create a new line to handle the overflow. If verify is in effect, print the changed line at the terminal.
- Print up to and replace string - print the line at the terminal up to the point where the character string replacement will begin. Enter the new data, overlaying the old text in the record. Format the changes (IKJEBELE).

3 Write the changed record into the utility data set. If the data set type is IPLI or BASIC, update the reverse polish data set. Repeat steps 2 and 3 until the line range is exhausted. Return control to IKJEBEMA.

Object Modules: IKJEBECH, IKJEBELE, IKJEBESE

Diagram 1.10.1 CKPOINT Subcommand Processing (Part 1 of 2)



1 Parse the subcommand.

2 Determine the options specified on the subcommand. ,

3 Invoke IKJEBEUT to accomplish the checkpoint.

4 Update the actual count.

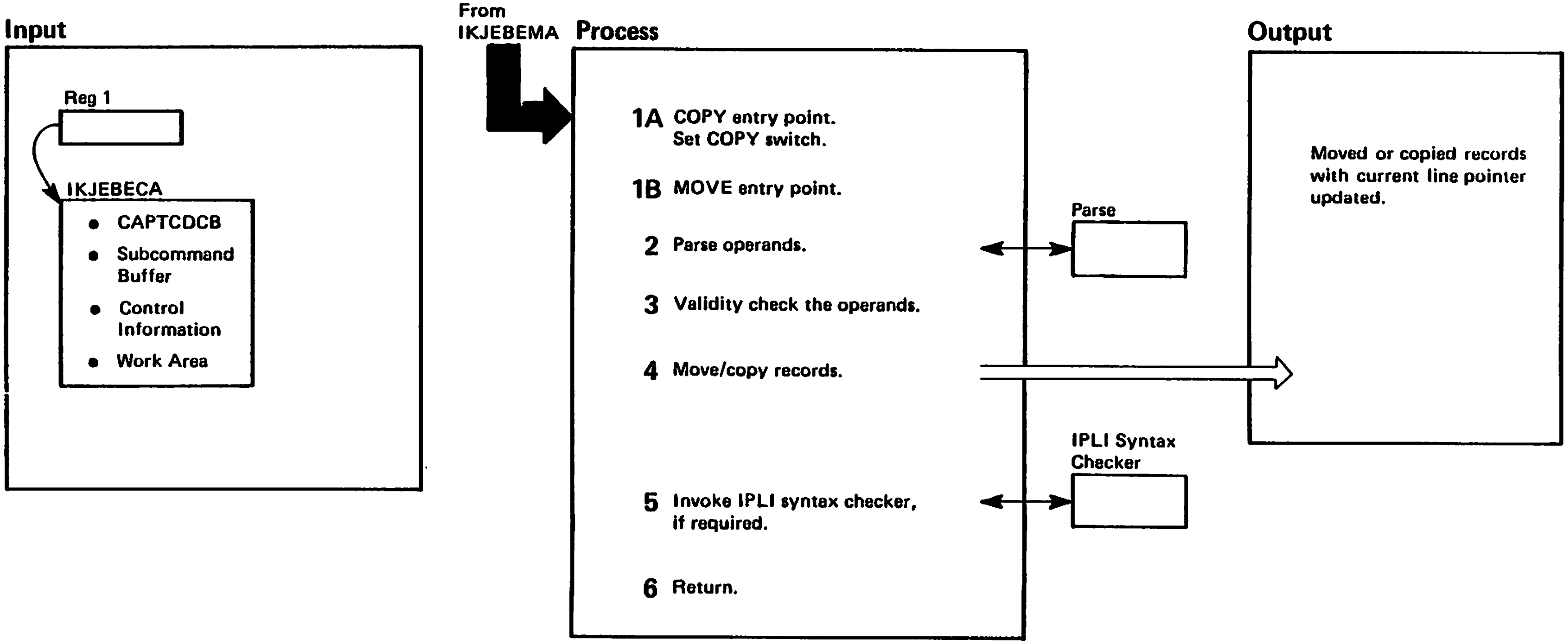
5 Return.

Diagram 1.10.1 CKPOINT Subcommand Processing (Part 2 of 2)

	Extended Description	Module	Label
1	Parse the command. Determine validity of values entered on the command.		
2	Determine options specified on the subcommand. • Stop automatic checkpointing if the CKPOINT value is zero (0). • Take a checkpoint if CKPOINT has no value.		
3	Invoke IKJEBEUT to take a checkpoint, and set the interval checkpoint count if the CKPOINT value is greater than zero.		
4	Update the actual modifications counter to reflect the above situation.		
5	Return to caller.		

Object Module: IKJEBECK

Diagram 1.11. MOVE/COPY Subcommand Processing (Part 1 of 2)



1A COPY entry point. Set COPY switch.

1B MOVE entry point.

2 Parse operands.

3 Validity check the operands.

4 Move/copy records.

5 Invoke IPLI syntax checker, if required.

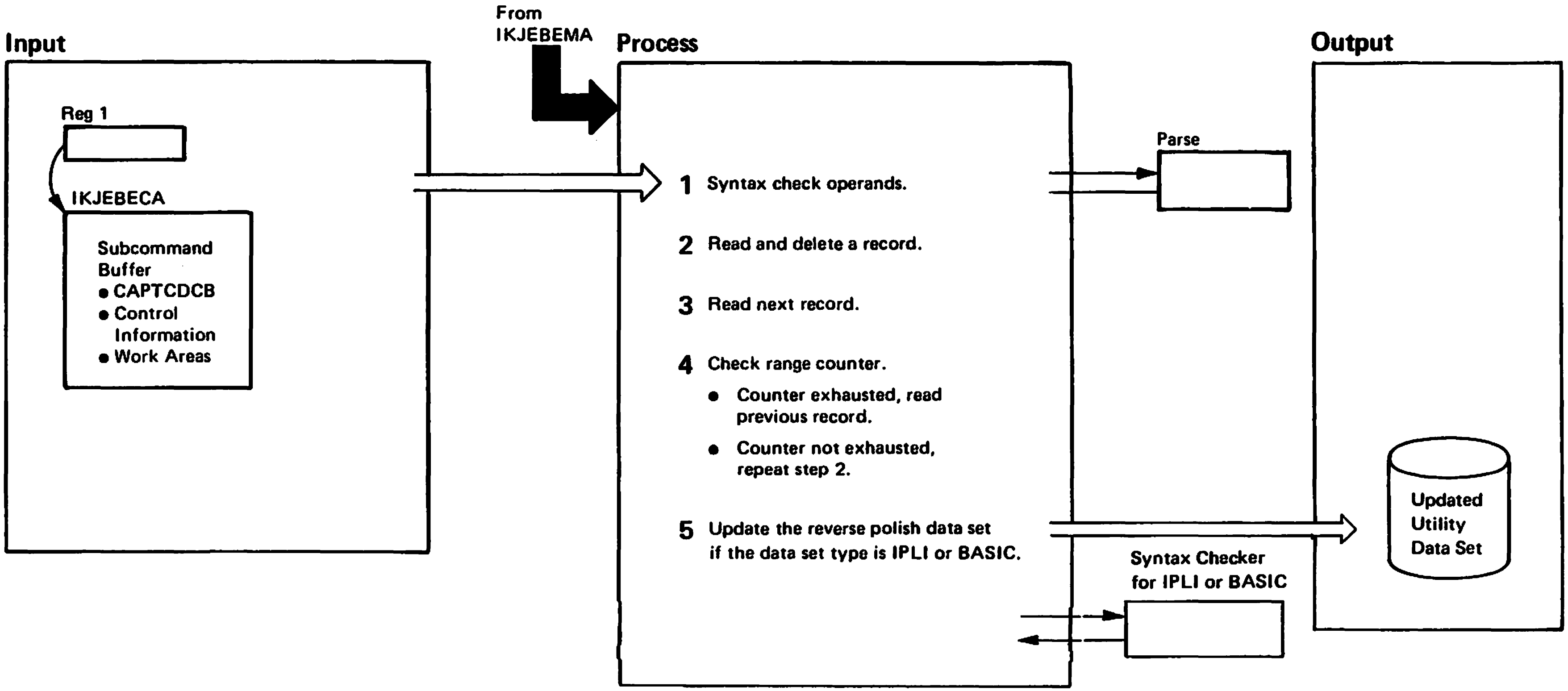
6 Return.

Diagram 1.11. MOVE/COPY Subcommand Processing (Part 2 of 2)

	Extended Description	Module	Label
1A & 1B	Set switches for later delete processing for MOVE and later allocation decision for COPY.		
2	Use parse (IKJPARS) to obtain acceptable operands for processing.		
3	Check validity of operands to ensure that no MOVE/COPY is done within itself, and that there are, in fact, records in the line range specified.		
4	Check the data set type: • VSBASIC - if a data exit routine is present, copy (IKJEBEDC) the utility data set to an in-core data set, and invoke (IKJEBEDX) the data exit routine to perform the MOVE/COPY. • Other - use MOVE to delete (IKJEBEUT) the records from where they were and use COPY to write (IKJEBEUT) them to the location specified by the operands. If the moved or copied records cause renumbering in the data set, allocate a secondary utility data set (IKJEBEUI) to ensure that no data will be lost.		
5	If the data set type is IPLI, update the reverse polish data set (IKJEBEMR).		
6	Return control to IKJEBEMA.		

Object Modules: IKJEBEDC, IKJEBEDX, IKJEBEMC, IKJEBEMR, IKJEBEUI, IKJEBEUT

Diagram 1.12. DELETE Subcommand Processing (Part 1 of 2)



1 Syntax check operands.

2 Read and delete a record.

3 Read next record.

4 Check range counter.

- Counter exhausted, read previous record.
- Counter not exhausted, repeat step 2.

5 Update the reverse polish data set if the data set type is IPLI or BASIC.

Diagram 1.12. DELETE Subcommand Processing (Part 2 of 2)

	Extended Description	Module	Label
1	Use Parse to check the syntax of the DELETE subcommand (IKJEBEDE) operands. If no operands were specified, set the line count to 1.		
2	Read and delete (IKJEBEUT) a record.		
3	Read (IKJEBEUT) the next record.		
4	Check the range counter to see if the required number of records has been deleted (IKJEBEDE). • Counter exhausted; read the previous record, and set the current line value to the corresponding record key. If no previous record exists, set the counter to 0. • Counter not exhausted; repeat step 2.		
5	If the data set type is BASIC or IPLI, use the language processor to remove lines from the reverse polish data set. Request the language processor via the standard syntax checker interface. Return control to IKJEBEMA.		

Object Modules: IKJEBEDE, IKJEBEUT

Diagram 1.13. DOWN Subcommand Processing (Part 1 of 1)

(No drawing with text.)

Extended Description **Module** **Label**

1 Pass control to the DOWN subcommand processor (IKJEBEDO) from the controller routine (IKJEBEMA). Check the operand switch (CAOPERAND) to see if the user specified an operand on the subcommand. If the user specified an operand, use Parse to check the syntax of the subcommand.

2 Depending on what the user specified, either:

- Read records toward the end of the utility data set until the number of records specified by the count operand has been read. Request the EDIT access method interface routine to locate and read the record.

  If the data set is empty (return code 4), send message IKJ52501I to the user. Set the current line pointer to zero and return control to IKJEBEMA.

  If the data set is not empty, request IKJEBEUT to locate and read the record following the current one. Continue the process until the current line pointer has been moved down the requested (count operand) number of lines.

- Read the record following the one pointed to by the current line pointer, if no operands were specified. The process is the same as above, except the record count is 1.

3 Set the current line pointer to the value of the last record read and turn on the "line to be verified" switch. If the last record read is the last record in the data set, send message IKJ52500I to the user.

4 Return control to IKJEBEMA.

Object Modules: IKJEBEDO, IKJEBEUT

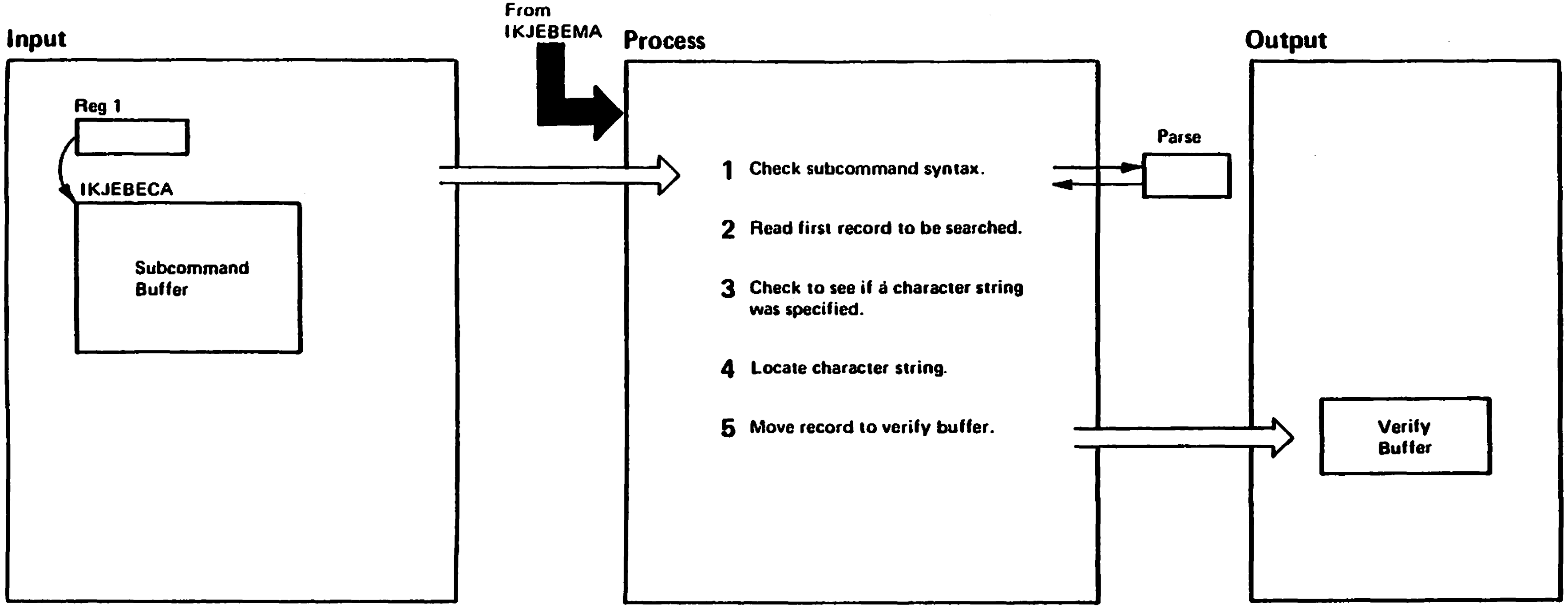
Diagram 1.14. END Subcommand Processing (Part 1 of 1)

(No drawing with text.)

	Extended Description	Module	Label
1	Pass control to the END subcommand processor (IKJEBEEN) from the controller routine (IKJEBEMA). If the utility data set has been changed since the last SAVE, issue message IKJ52555I to prompt the user to enter END or SAVE. If the user enters anything other than SAVE or END, return control to IKJEBEMA. If the user enters SAVE, perform the SAVE (IKJEBESA) operation. If the user enters END, continue. If the user entered END with the SAVE keyword, perform the SAVE operation (IKJEBESA). If the user entered END with the NOSAVE keyword, continue.		
2	Delete the EDIT Access Method (IKJEBEEX).		
3	If the data set type is BASIC or IPLI, or if the scan switch is on, use the SCAN subcommand processor (IKJEBESC) for final entry to the syntax checker.		
4	Cancel the abnormal end and attention exits.		
5	Use DAIR to free the EDIT data set.		
6	Delete the permanent resident service routines (IKJEBEMS and IKJEBELE).		
7	Return control to IKJEBEMA.		

Object Modules: IKJEBEEN, IKJEBESA, IKJEBESC, IKJEBEEX

Diagram 1.15. FIND Subcommand Processing (Part 1 of 2)



1 Check subcommand syntax.

2 Read first record to be searched.

3 Check to see if a character string was specified.

4 Locate character string.

5 Move record to verify buffer.

Diagram 1.15. FIND Subcommand Processing (Part 2 of 2)

Extended Description **Module** **Label**

1 Use Parse to check the syntax of the operands on the FIND subcommand (IKJEBEFI), if there are any. If the user did not specify any operands, and has not entered the FIND subcommand previously, prompt the user for a character string.

2 Read the first record to be searched, using IKJEBEUT. That record is:

- the next record if the user did not enter any operands and was not prompted for any.
- the current record if the user entered operands.
- the first record if the current line number (CACURNUM) is zero.

3 Check to see if the user specified a character string. If the user did, convert the string to uppercase, if required, and place the string in the FIND buffer (CAFIBFR). If the user did not specify a string, use the contents of the FIND buffer as the string.

4 Locate the requested character string.

- No offset specified - character string search (IKJEBESE) searches for the requested character string at every offset in the entire range. The search is made across line boundaries for text type data sets.

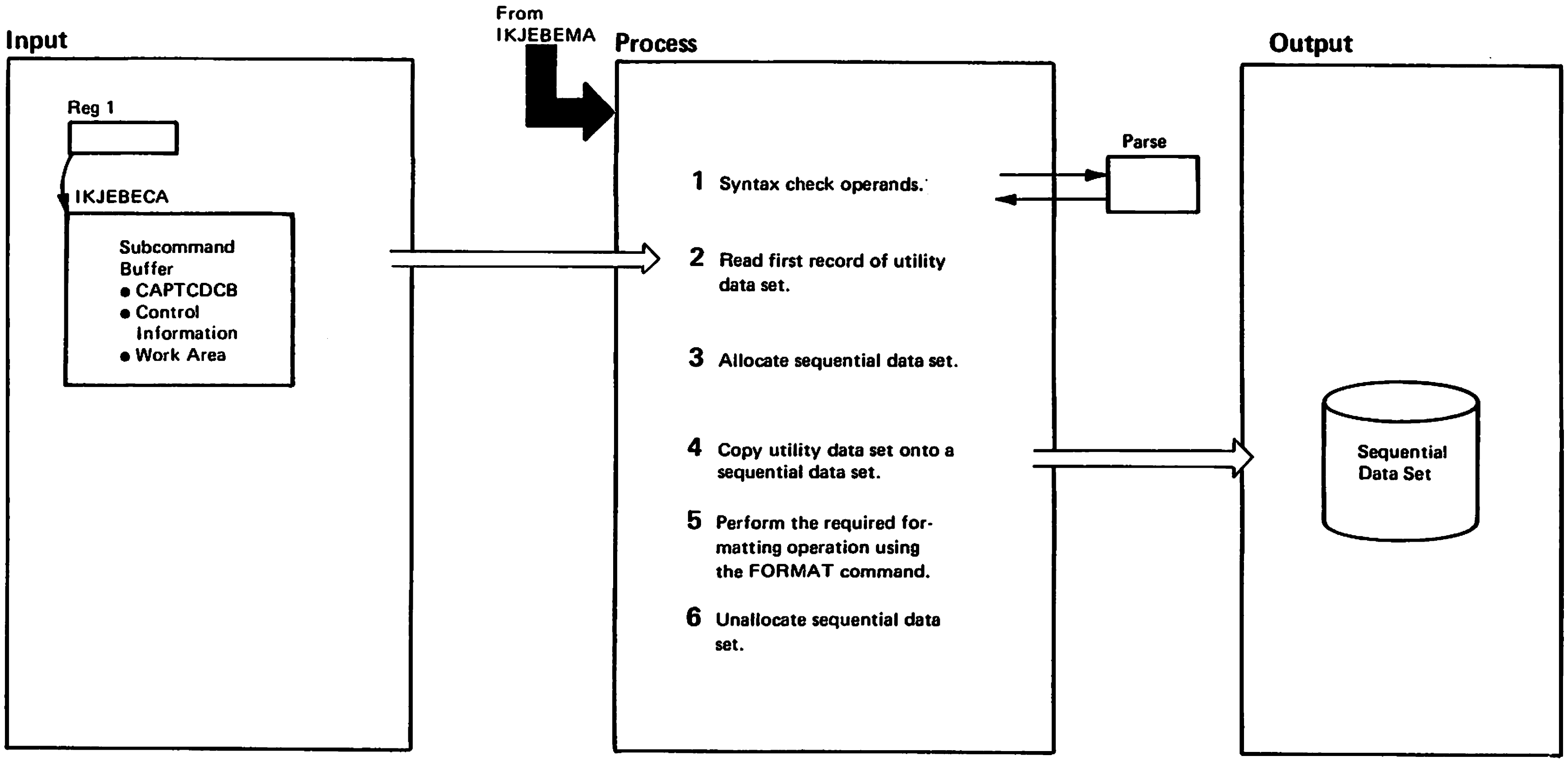
- Offset specified - IKJEBEFI reads records and checks for the character string at the specified offset for each record in the range.

If the character string is not located, issue an error message.

5 Move the record containing the requested character string into the verify buffer in CATEMPBF. Turn on the "line to be verified" switch, CALNTOVF. Store the line number in CACURNUM, the current line number. Return control to IKJEBEMA.

Object Modules: IKJEBEFI, IKJEBEUT, IKJEBELE, IKJEBESE

Diagram 1.16. FORMAT Subcommand Processing (Part 1 of 2)



1 Syntax check operands.

2 Read first record of utility data set.

3 Allocate sequential data set.

4 Copy utility data set onto a sequential data set.

5 Perform the required formatting operation using the FORMAT command.

6 Unallocate sequential data set.

Diagram 1.16. FORMAT Subcommand Processing (Part 2 of 2)

	Extended Description	Module	Label
1	Use Parse to check the syntax of the operands on the FORMAT subcommand (IKJEBEFO), if there are any.		
2	Read (IKJEBEUT) the first record of the EDIT utility data set.		
3	Test to see if the range begins at the current line pointer, a specified line, or the beginning of the data set. Allocate (IKJEBEDA) a sequential data set according to the amount of space required.		
4	Copy (IKJEBEFC) the EDIT utility data set onto the sequential data set.		
5	Perform the required formatting operations using the FORMAT command processor. Attach the system FORMAT command (a Program Product) by the FORMAT subcommand processor (IKJEBEFO) via IKJEBECI.		
6	Free (IKJEBEDA) the sequential data set. Return control to IKJEBEMA from the FORMAT subcommand processor.		

Object Modules: IKJEBEFO, IKJEBEUT, IKJEBEFC, IKJEBECI, IKJEBEDA

Diagram 1.17. FREE ALL Subcommand Processing (Part 1 of 2)

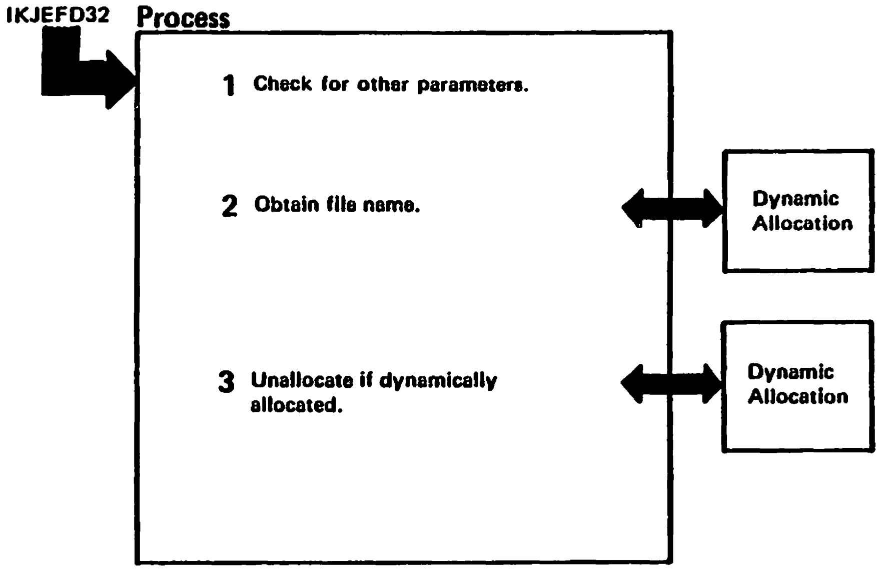
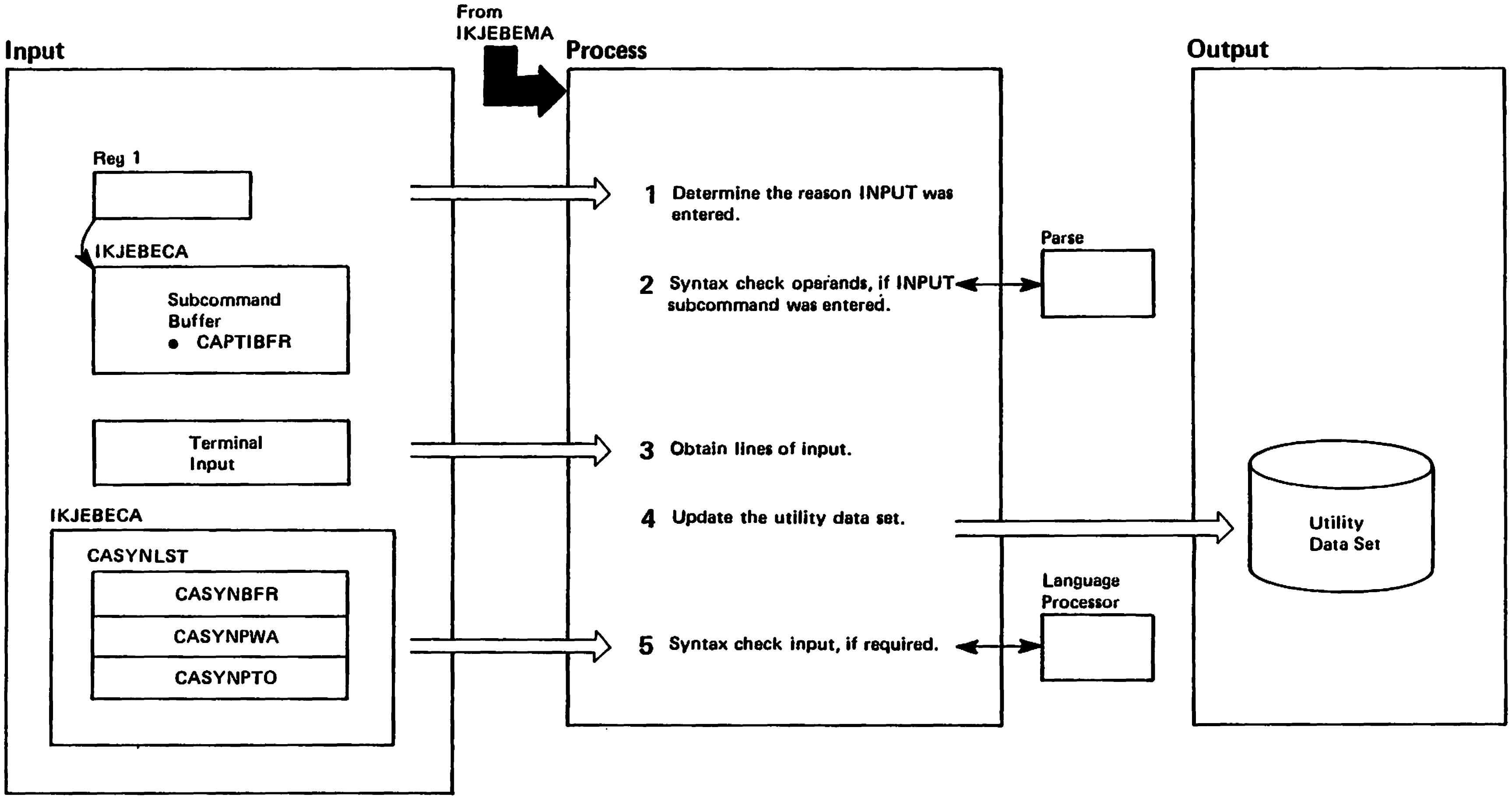


Diagram 1.17. FREE ALL Subcommand Processing (Part 2 of 2)

	Extended Description	Module	Label
1	Check to see if the user specified any other parameters on the command. If so, issue an error message and terminate the command.		
2	Invoke the RETRIEVE function of Dynamic Allocation, specifying the relative request number to get the associated file name. If the return code from dynamic allocation is not zero, use the DAIR failure message routine (IKJEFF18) to analyze the return code and send the appropriate error message to the user.		
3	If the file was dynamically allocated, use the UNALLOCATE function to unallocate the file. If the return code from dynamic allocation is not zero, use the DAIR failure message routine (IKJEFF18) to analyze the return code and send the appropriate error message to the user. Repeat step 2, specifying the next relative request number.		

Object Module: IKJEFD38

Diagram 1.18. INPUT Subcommand Processing (Part 1 of 2)



1 Determine the reason INPUT was entered.

2 Syntax check operands, if INPUT subcommand was entered.

3 Obtain lines of input.

4 Update the utility data set.

5 Syntax check input, if required.

Diagram 1.18. INPUT Subcommand Processing (Part 2 of 2)

Extended Description **Module** **Label**

1 Determine the reason INPUT (IKJEBEIP) was entered. Entry was from one of the following:

- EDIT command for a new or an old empty data set - Initialization (IKJEBEIN) passes a request to the mainline (IKJEBEMA) that the INPUT subcommand processor receive control. Input begins at line 10 and line numbers are increased by 10.
- INSERT subcommand with no operands - The INSERT subcommand (IKJEBEIS) returns control to the mainline (IKJEBEMA), indicating input mode has been entered. INPUT begins at the current line number +1 and line numbers are increased by 1.
- INPUT subcommand without operands - INPUT begins inserting data at the bottom of the data set.
- INPUT subcommand with operands - INPUT begins after a specified line if the I form is used or at the specified line if the R form is used.
- Null line entered while in EDIT mode - If input mode was previously specified, INPUT will begin after the last line entered in input mode. Or, if this is the first entry into input mode for this EDIT session, INPUT will begin at the bottom of the data set.

2 If the user entered the INPUT subcommand, use Parse to check the syntax of the operands. If the operands specify that a line is to be deleted or that a line needs to be located, use the EDIT access method (IKJEBEUT).

3 Get a line of input from the terminal. If syntax checking is in effect, or character prompting is requested, use a character string for line prompting to request a line. If the automatic line prompting facility (STAUTOLN) is used, get a line from the terminal. Format the line to be moved to the utility data set (IKJEBELE). If the user enters a null line, return control to the mainline (IKJEBEMA).

4 Use the EDIT access method (IKJEBEUT) to write a line of data to the utility data set. If the data set is numbered and there is no room to insert the line, terminate input mode and issue an error message. If the data set is NONUM, renumber it by reading in (IKJEBEUT) each of the lines, increasing the line number by one, and writing it back into the data set.

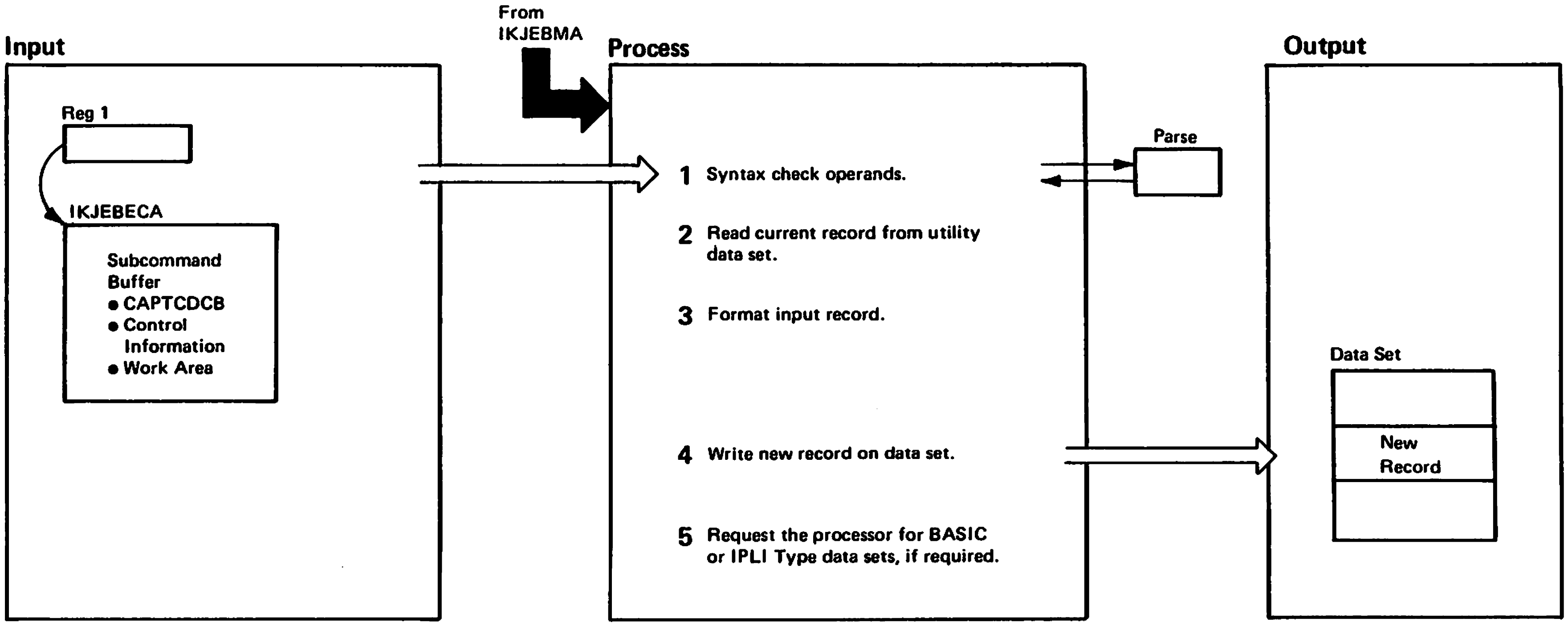
5 Use the appropriate language processor to check the syntax of the line of data. The line of data is passed to the language processor in the syntax checker parameter list (CASYNLST) set up by SCAN. If there is a syntax error, issue the message returned by the language processor, and terminate input mode.

If the data set type is IPLI or BASIC, and syntax checking is not in effect, use the language processor to update the reverse polish data set.

Continue processing with step 3.

Object Modules: IKJEBEIP, IKJEBEIS, IKJEBELE, IKJEBEMA, IKJEBEUT

Diagram 1.19. INSERT Subcommand Processing (Part 1 of 2)



1 Syntax check operands.

2 Read current record from utility data set.

3 Format input record.

4 Write new record on data set.

5 Request the processor for BASIC or IPLI Type data sets, if required.

Diagram 1.19. INSERT Subcommand Processing (Part 2 of 2)

Extended Description **Module** **Label**

1 Use Parse to check the syntax of the INSERT subcommand operands. If the user did not specify any operands, return control to IKJEBEMA and enter input mode. (IKJEBEMA requests INPUT subcommand processing on receiving a return code of 4.) If the input string is greater than 255 characters, reset it to the maximum value of 255.

2 Read the current record from the utility data set. If the data set is using the NUM option, and the record to be inserted already exists, terminate the insertion and return control to IKJEBEMA. Otherwise, provide space within the data set for the record being inserted.

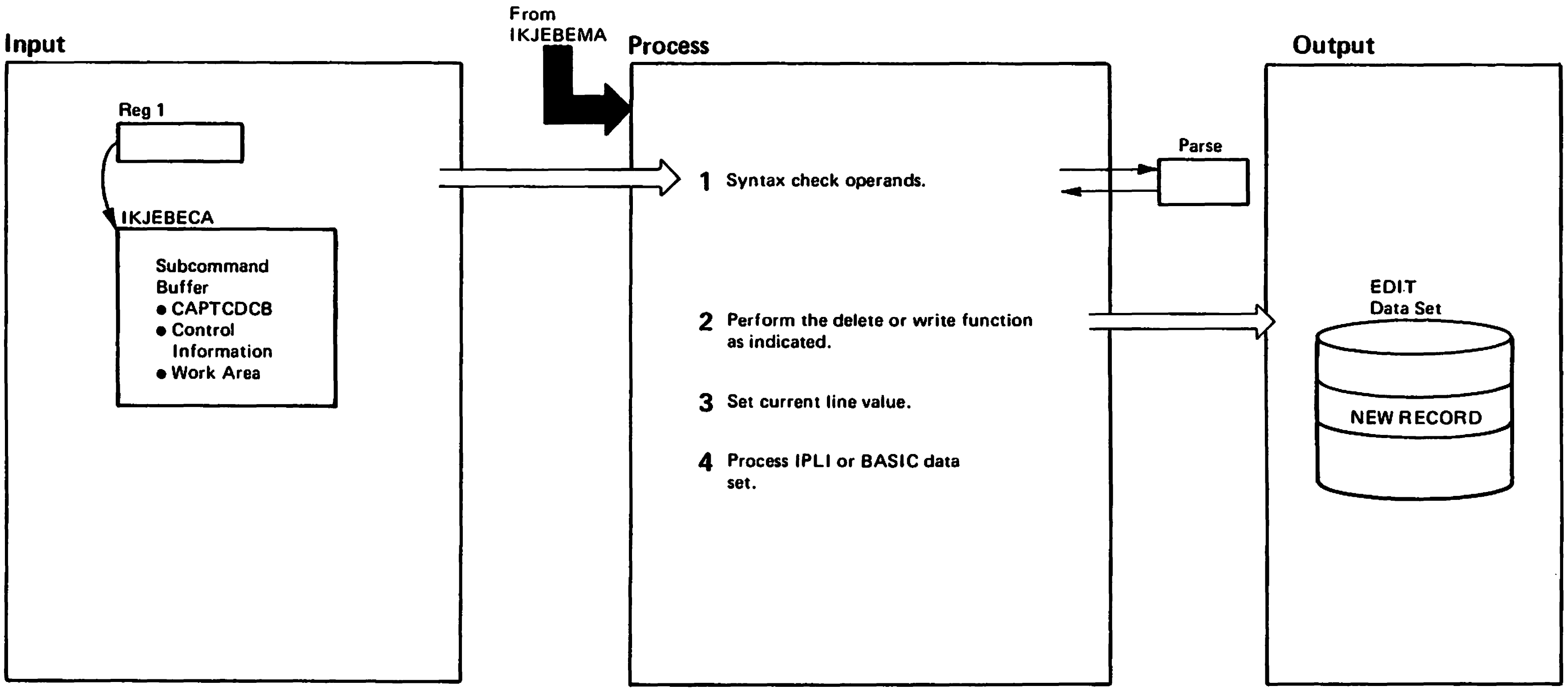
3 Format the record to be inserted.

4 Write (IKJEBEUT) the new record into the data set. Update the current line pointer.

5 If the data set type is BASIC or IPLI, request the language processor to update the reverse polish data set. Return control to IKJEBEMA from the INSERT subcommand processor.

Object Modules: IKJEBEIS, IKJEBEUT

Diagram 1.20. LINE INSERT/REPLACE/DELETE Processing (Part 1 of 2)



1 Syntax check operands.

2 Perform the delete or write function as indicated.

3 Set current line value.

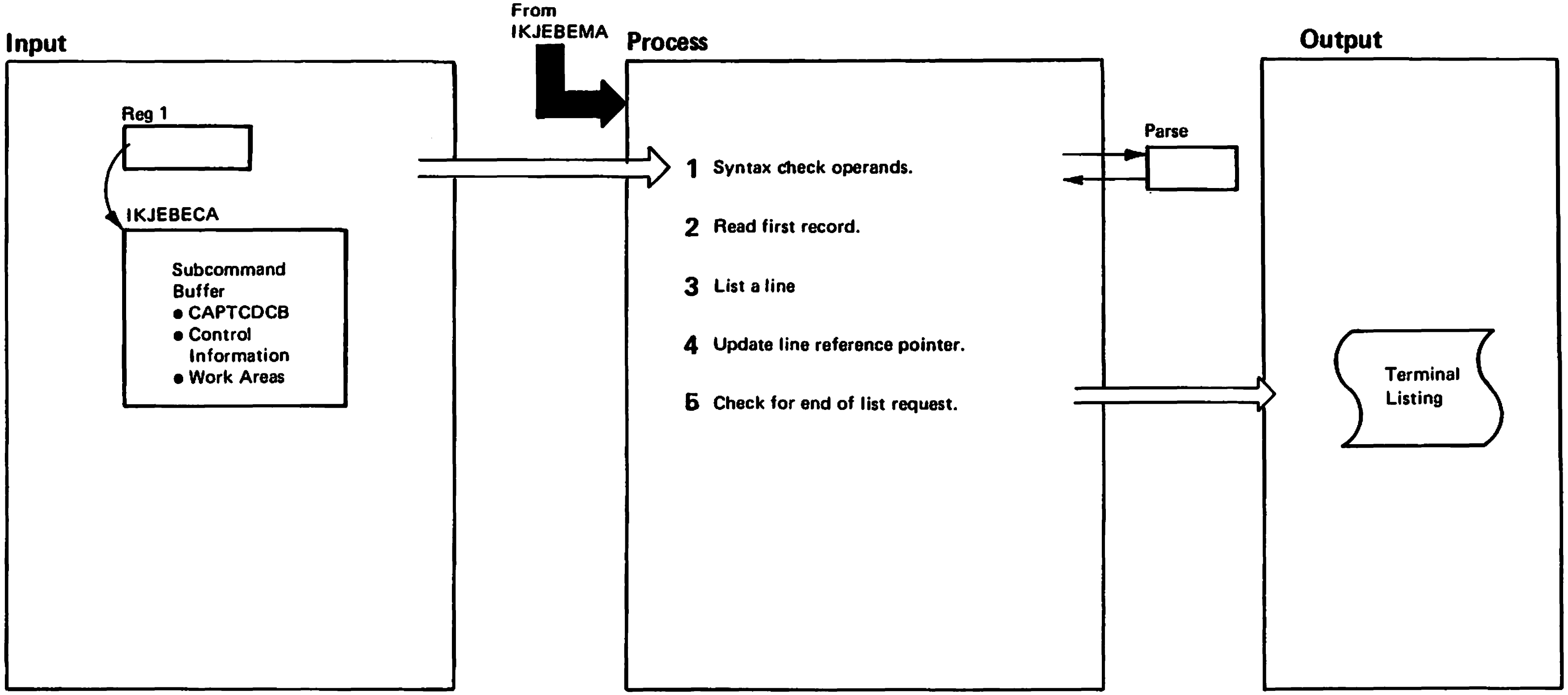
4 Process IPLI or BASIC data set.

Diagram 1.20. Line Insert/Replace/Delete Processing (Part 2 of 2)

	Extended Description	Module	Label
1	Use Parse to check the syntax of the insert/replace/delete (IKJEBELI) operands, if necessary. If a string length was specified, and it exceeds 255 characters, reset the length to 255 and indicate a line overflow. For the insert/replace functions, the input line may be edited (IKJEBELE) for tabs and uppercase.		
2	Perform the "delete a line" or "write a line" function as indicated. If text was not entered and the user requested a delete, delete the specified record (IKJEBEUT). If the user requested insert or replace, write a record to the EDIT data set (IKJEBEUT).		
3	Set the current line value to the value of the last line operation.		
4	If the data set type is BASIC or IPLI, update the reverse polish data set. Return control to IKJEBEMA.		

Object Modules: IKJEBELE, IKJEBELI, IKJEBEUT

Diagram 1.21. LIST Subcommand Processing (Part 1 of 2)



1 Syntax check operands.

2 Read first record.

3 List a line

4 Update line reference pointer.

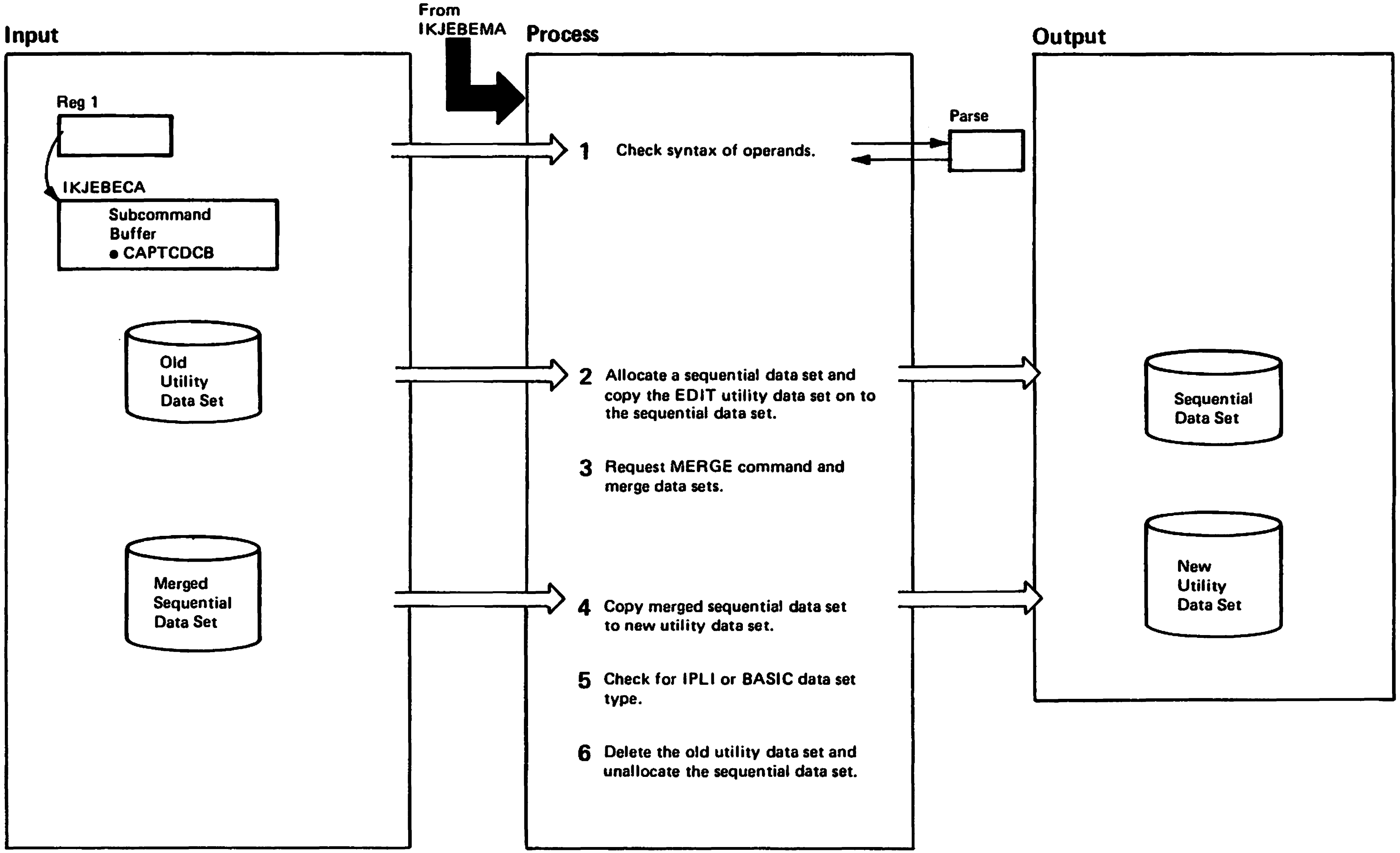
5 Check for end of list request.

Diagram 1.21. LIST Subcommand Processing (Part 2 of 2)

	Extended Description	Module	Label
1	Use Parse to check the syntax of the operands on the LIST subcommand (IKJEBELT), if necessary. If the user did not specify any operands, set the counter to 99999999 so that the entire data set will be printed.		
2	Read (IKJEBEUT) the first record as specified by the operands (current record, first record in data set, or specified record).		
3	Use PUTLINE to list a line at the terminal.		
4	Update the current line reference pointer (CACURNUM).		
5	Test for end-of-list request. • For a listing by line number, read (IKJEBEUT) the next record in the range to be listed (Step 2). If the range has been exhausted, return control to IKJEBEMA. • For a listing by counter, read (IKJEBEUT) the next record and decrease the counter by 1. If the counter is 0, return control to IKJEBEMA.		

Object Modules: IKJEBELT, IKJEBEUT

Diagram 1.22. MERGE Subcommand Processing (Part 1 of 2)



1 Check syntax of operands.

2 Allocate a sequential data set and copy the EDIT utility data set on to the sequential data set.

3 Request MERGE command and merge data sets.

4 Copy merged sequential data set to new utility data set.

5 Check for IPLI or BASIC data set type.

6 Delete the old utility data set and unallocate the sequential data set.

Diagram 1.22. MERGE Subcommand Processing (Part 2 of 2)

	Extended Description	Module	Label
1	Use Parse to check the syntax of the operands on the MERGE subcommand (IKJEBEME). If operands are missing, prompt the user to enter them.		
2	Allocate (IKJEBEDA) a sequential data set with a disposition of NEW, CATLG, DELETE. Copy (IKJEBEFC) the EDIT utility data set to the allocated sequential data set. Free (IKJEBEDA) the sequential data set with a disposition of OLD, KEEP.		
3	Build a model of the program product MERGE command based on the name of the sequential data set containing a copy of the EDIT utility data set, the subcommand operands, and the EDIT data set attributes. IKJEBECI requests the program product MERGE command (IKJEBMIN) to merge the data sets.		
4	Reallocate (IKJEBEDA) the output sequential data set. Copy (IKJEBECO) the merged sequential data set to a new utility data set.		
5	Check the data set type to see if it is IPLI or BASIC. If it is, delete (IKJEBEMR) the old reverse polish data set and build a new one.		
6	Delete (IKJEBEEX) the old utility data set and free (IKJEBEDA) the sequential data set. Turn on the "line to be verified" switch. Return control to IKJEBEMA.		

Object Modules: IKJEBECI, IKJEBECO, IKJEBEDA, IKJEBEEX, IKJEBEFC, IKJEBMIN, IKJEBEME, IKJEBEMR

Diagram 1.23. PROFILE/SEND/HELP/ALLOCATE Subcommand Processing (Part 1 of 1)

(No drawing with text.)

	Extended Description	Module	Label
1	Pass control to IKJEBEHE from IKJEBEMA.		
2	Request IKJEBECI to attach the requested system command. Pass to IKJEBECI the parameter list containing a pointer to the EDIT communications area and the buffer containing the subcommand entered from the terminal.		
3	When the system command processor finishes, return control to IKJEBEMA from IKJEBECI.		

Object Modules: IKJEBECI, IKJEBEHE

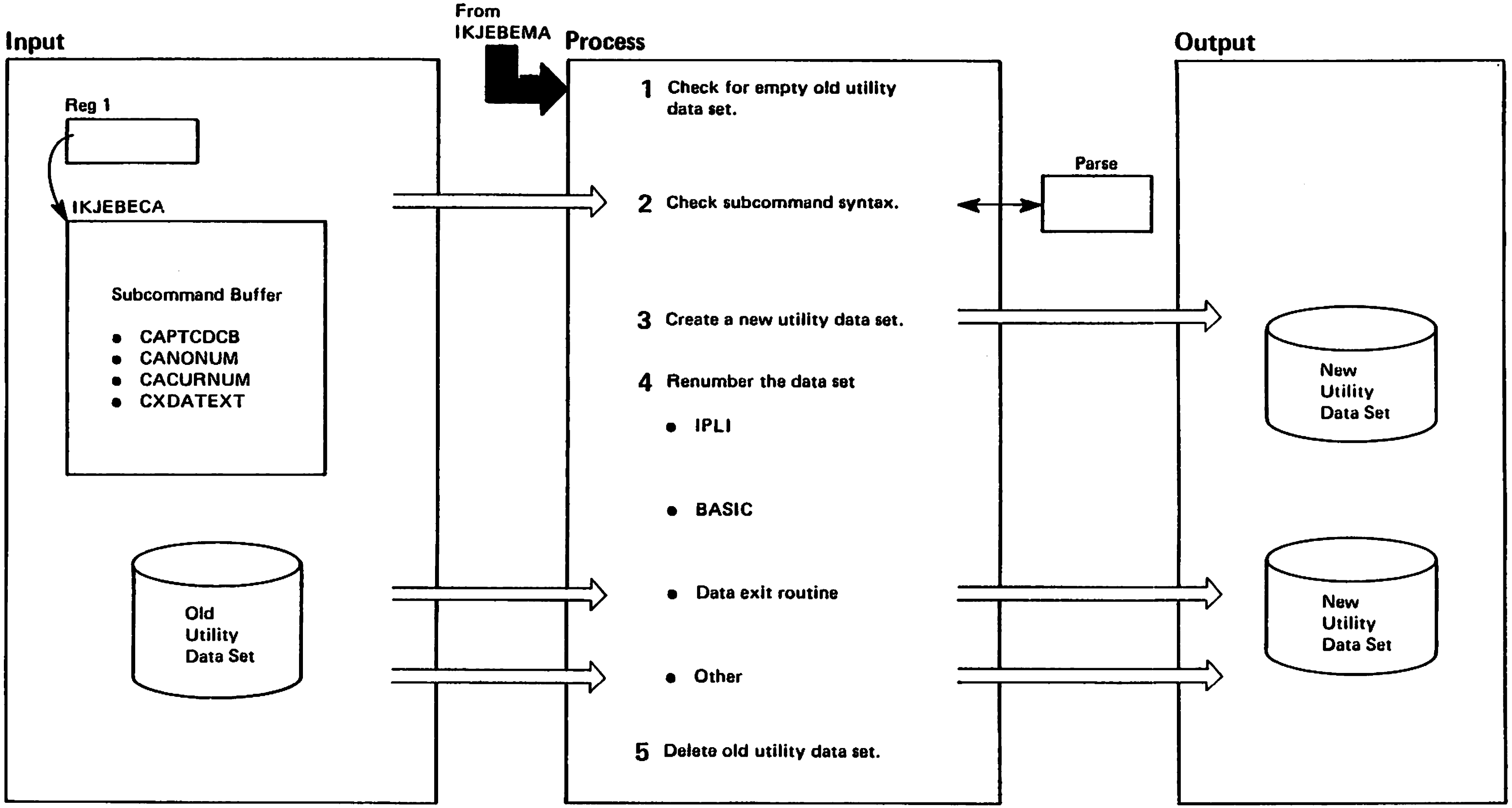
Diagram 1.24. EXEC Subcommand Processing (Part 1 of 1)

(No drawing with text.)

	Extended Description	Module	Label
1	Pass control to IKJEBECI (via LINK) from IKJEBEMA.		
2	Pass to IKJEBECI the parameter list containing a pointer to the EDIT communications area and the buffer containing the subcommand entered from the terminal. IKJEBECI attaches the EXEC command processor.		
3	When the system command processor finishes, return control to IKJEBEMA from IKJEBECI.		

Object Module: IKJEBECI

Diagram 1.25. RENUM Subcommand Processing (Part 1 of 2)



1 Check for empty old utility data set.

2 Check subcommand syntax.

3 Create a new utility data set.

4 Renumber the data set

- IPLI
- BASIC
- Data exit routine
- Other

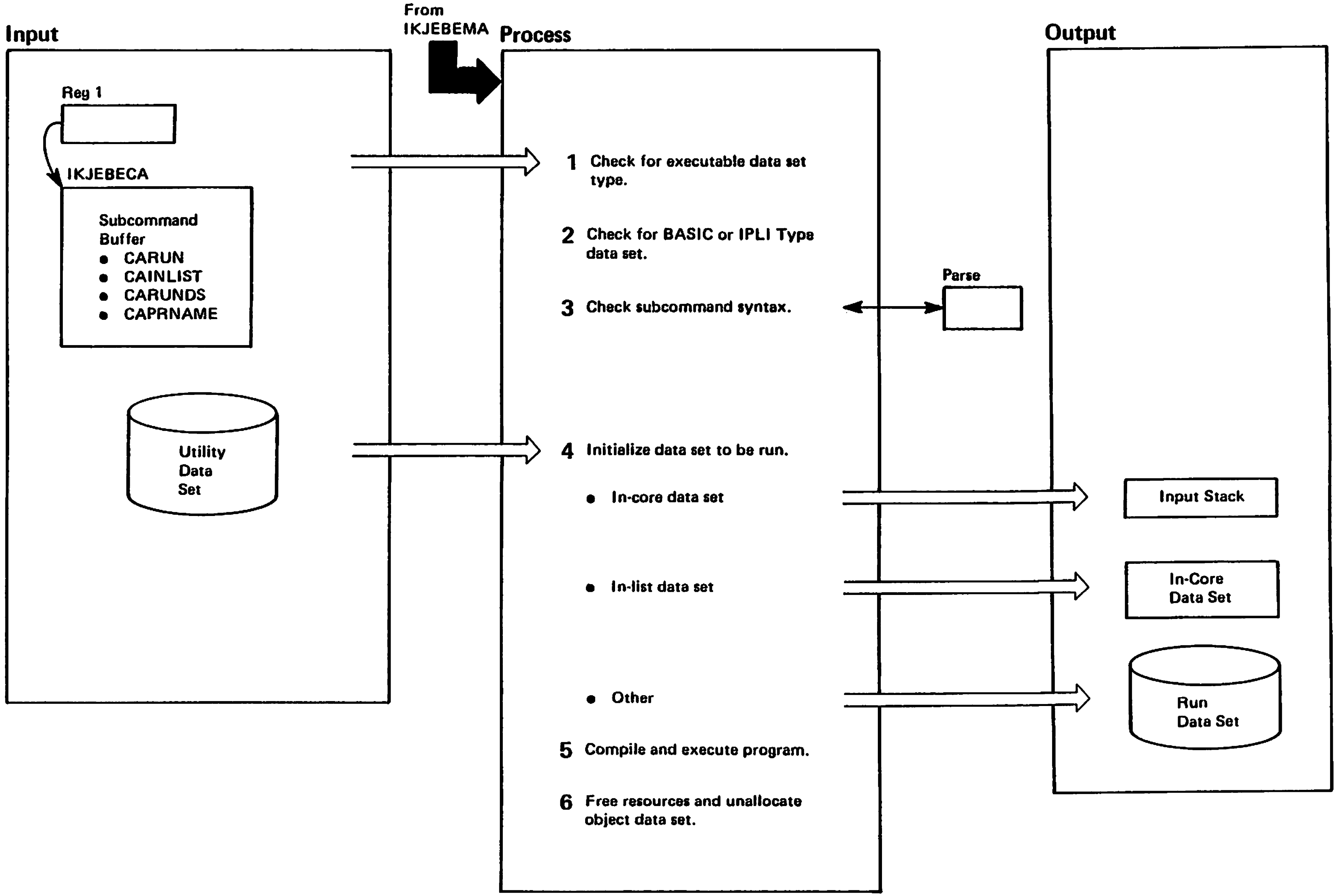
5 Delete old utility data set.

Diagram 1.25. RENUM Subcommand Processing (Part 2 of 2)

	Extended Description	Module	Label
1	If the old utility data set is empty, return to the caller.		
2	Use Parse to check the syntax of the operands on the RENUM subcommand (IKJEBERE).		
3	Create (IKJEBEUI) a new utility data set.		
4	Renumber the data set. The following are renumbering processes for data set types: *IPLI* - renumber and update (IKJEBEMR) the reverse polish data set. *BASIC* - renumber (IKJEBERN) and update (IKJEBEMR) the reverse polish data set. *Data exit routine specified* - copy (IKJEBEDC) the old utility data set to an in-core data set and renumber (IKJEBEDX) the data set. Renumber the data set (data exit routine) and copy the renumbered in-core data set to the new utility data set (IKJEBEUT). *Other* - read one record at a time (IKJEBEUT), renumber it (IKJEBERE), and write (IKJEBEUT) it on the new utility data set.		
5	Delete (IKJEBEEX) the old utility data set.		

Object Modules: IKJEBEDC, IKJEBEDX, IKJEBEEX, IKJEBEMR, IKJEBERE, IKJEBERN, IKJEBEUI, IKJEBEUT

Diagram 1.26. RUN Subcommand Processing (Part 1 of 2)



1 Check for executable data set type.

2 Check for BASIC or IPLI Type data set.

3 Check subcommand syntax.

4 Initialize data set to be run.

- In-core data set
- In-list data set
- Other

5 Compile and execute program.

6 Free resources and unallocate object data set.

Diagram 1.26. RUN Subcommand Processing (Part 2 of 2)

Extended Description **Module** **Label**

1 Check to see if the data set is executable (CARUN is on). If it is not, issue an error message and return control to IKJEBEMA.

2 Check data set type to see if it is BASIC or IPLI. If it is, request the appropriate language processor to compile and execute the data set, and return control to IKJEBEMA.

3 Use Parse to scan and check the syntax of any operands on the subcommand. Build a command buffer containing the specified operands.

4 Check the data set type and process the data set accordingly.

In-Core Data Set Type - Read (IKJEBEUT) and move (IKJEBERU) the utility data set to a 4K dynamic area. Use the STACK service routine to place a copy of the data set on the input stack. Move a data set name of * into the command buffer.

In-List Data Set Type - Copy (IKJEBEDC) the utility data set to an in-core data set. Move a data set name of * and the INLIST keyword and its associated subfield into the command buffer.

Other Data Set Types - Allocate (IKJEBEDA) a run data set, and copy (IKJEBEFC) the utility data set to the RUN data set. Move the data set name into the command buffer.

5 Request (IKJEBECI) the appropriate prompter and compile and execute the program.

6 Free resources used for the source data set:

In-core - Delete the data set on the input stack.

In-list - Free storage occupied by the in-core data set.

Other - Free the source and object data sets. The RUN subcommand processor (IKJEBERU) returns control to IKJEBEMA.

Object Modules: IKJEBECI, IKJEBEDA, IKJEBEDC, IKJEBEFC, IKJEBERU, IKJEBEUT

Diagram 1.27. SAVE Subcommand Processing (Part 1 of 2)

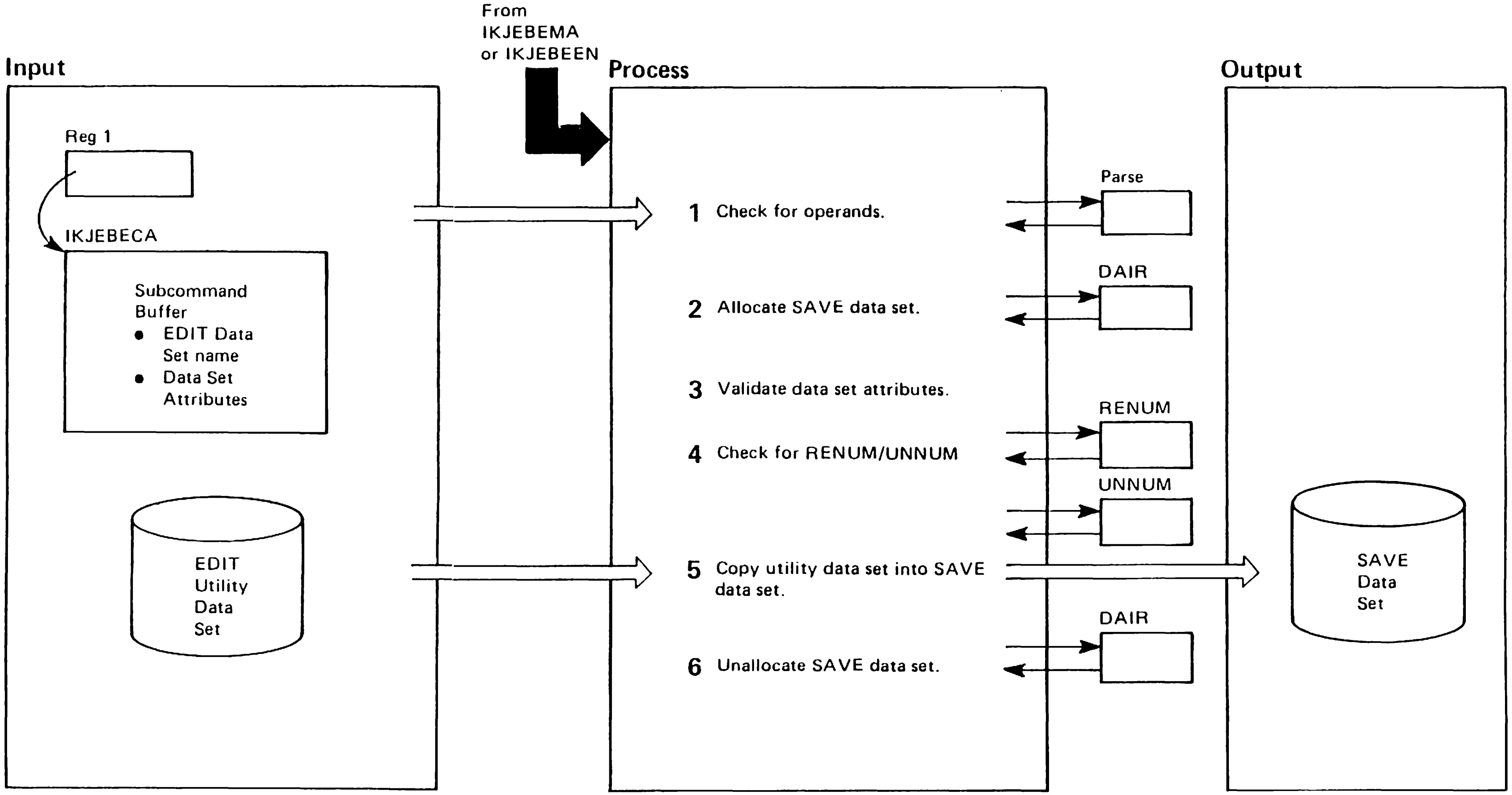
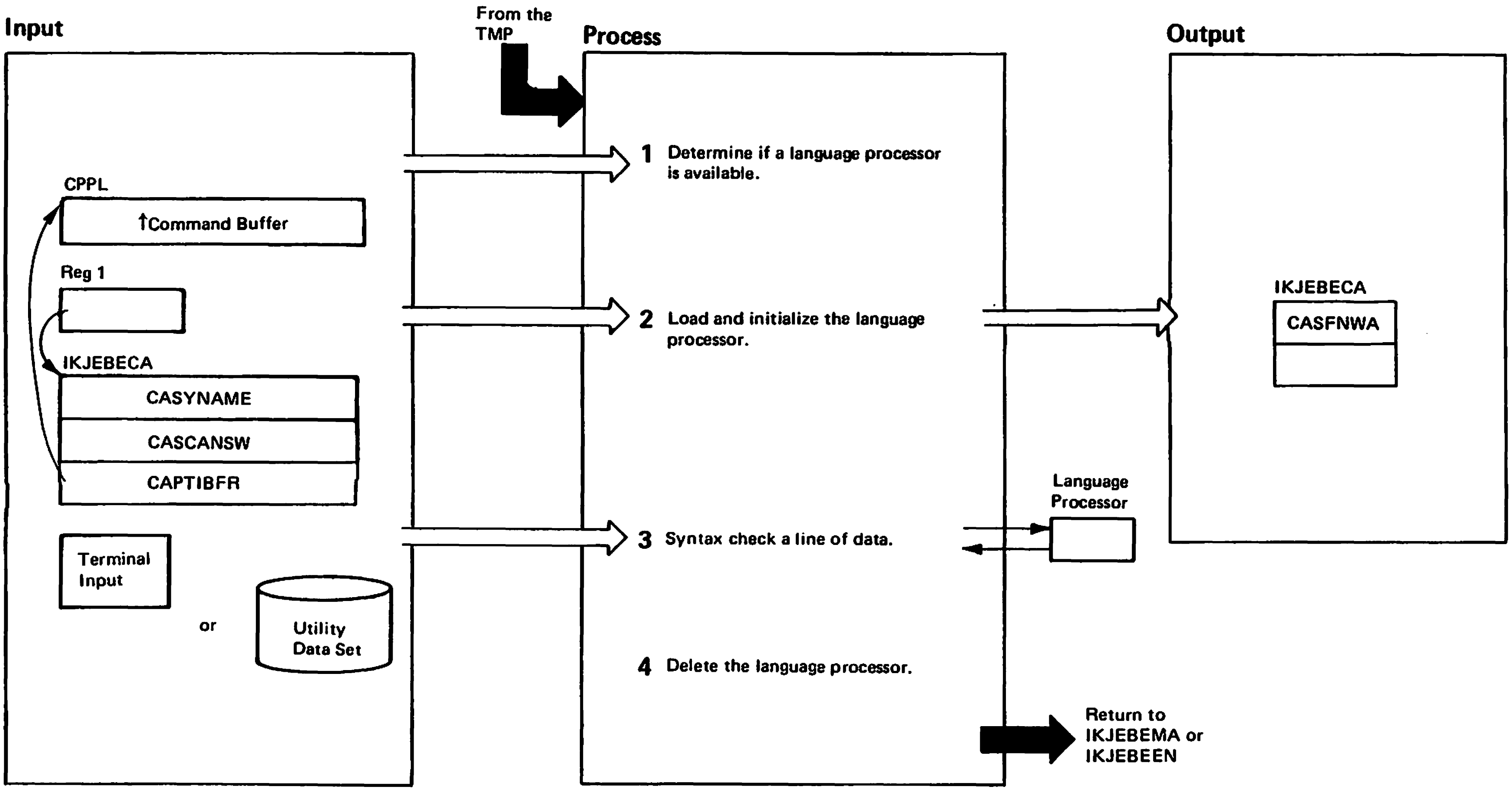


Diagram 1.27. SAVE Subcommand Processing (Part 2 of 2)

	Extended Description	Module	Label
1	Check for operands on the subcommand. If the user specified an operand, use Parse to check it. If the user did not specify any operands, or if data set name was '*', use the EDIT data set name. Use the Default service routine to fully qualify an unqualified name, and to determine if the data set is in the catalog. If NEW was used on the EDIT command, and the data set is in the catalog (already exists), prompt the user for action. If REUSE was specified on the EDIT command, do not prompt the user, and reuse the existing data set.		
2	Determine if the SAVE data set is sequential or partitioned, and use DAIR to allocate the data set accordingly.		
3	Validate data set attribute for those supported with the data set type. • LRECL, BLKSIZE, and RECFM must be valid for the data set type. • Block size must be less than or equal to the track length. • Check the DSCB for unsupported SAVE formats: undefined, variable spanned, track overflow, and machine or ANSI control characters.		
4	If the user specified either RENUM (IKJEBERE) or UNNUM (IKJEBEUN), call that function.		
5	Copy (IKJEBEFC) the EDIT utility data set to the SAVE data set.		
6	Use DAIR to unallocate the SAVE data set. Return control to the caller.		

Object Modules: IKJEBEFC, IKJEBESA, IKJEBERE, IKJEBEUN

Diagram 1.28. SCAN Subcommand Processing (Part 1 of 2)



1 Determine if a language processor is available.

2 Load and initialize the language processor.

3 Syntax check a line of data.

4 Delete the language processor.

Diagram 1.28. SCAN Subcommand Processing (Part 2 of 2)

Extended Description	Module	Label

1 Determine if a language processor (syntax checker) is available by checking CASYNAME in the EDIT Communications Area for a language processor name. The language processor is specified during the EDIT program initialization (IKJEBEIN).

2 Load and initialize the language processor when syntax checking is requested. If the user entered the EDIT command keyword SCAN, or if the data set type is IPLI or BASIC, the language processor is loaded (IKJEBEMA) and initialized (IKJEBESC) during the EDIT command processor initialization. Set CASCANSW to indicate that syntax checking is in effect.

If the user entered the SCAN subcommand (IKJEBESC) and syntax checking is not in effect, load and initialize the language processor.

Information is passed to the language processor in the Syntax Check Communications Area (CASYNWA in the EDIT Communications Area).

3 Check the syntax of a line of data (CASCANSW is on) received in input mode by passing it to the appropriate language processor. Pass data for IPLI and BASIC data set types to the language processor to update the reverse polish data set, whether SCAN is on or not.

If the user entered the SCAN subcommand, request (IKJEBESC) the EDIT Access Method (IKJEBEUT) to read each record in the range into storage and pass the data to the language processor.

4 When the user enters SCAN OFF, delete (IKJEBESC) the language processor and turn off CASCANSW to indicate that syntax checking is no longer in effect. For IPLI and BASIC type data sets, retain the language processor to maintain the reverse polish data set.

If the END subcommand processor is entered (IKJEBEEN) to terminate the EDIT program, and the language processor has not yet been deleted, request the SCAN subcommand processor (IKJEBESC) to delete the language processor.

Object Modules: IKJEBEEN, IKJEBEMA, IKJEBESC

Diagram 1.29. SUBMIT Subcommand Processing (Part 1 of 1)

(No drawing with text.)

Extended Description **Module** **Label**

1 Pass control to IKJEBESU from IKJEBEMA. Check to see if the subcommand has any operands.

YES - Link to Parse for syntax checking.

NO - Assume "SUBMIT*" was entered.

2 Begin to set up the command buffer by moving the subcommand name in for the command name.

3 Check for an "*" in the dslist.

YES - Allocate (IKJEBEDA) a new system data set (DISP=(NEW,DELETE)). Copy (IKJEBEFC) the utility data set into the system data set. Mark (IKJEBEDA) the data set "not in use." Copy the data set names from the PDL into the command buffer, replacing any "*" with the name of the system data set just created.

NO - Copy data set names from the PDL into the command buffer.

4 Attach (IKJEBECI) the SUBMIT command. Delete (IKJEBEDA) the system data set. Release any storage gotten by Parse (IKJRLSA). Return to IKJEBEMA.

Object Modules: IKJEBECI, IKJEBEDA, IKJEBEFC, IKJEBESU

Diagram 1.30. TABSET Subcommand Processing (Part 1 of 1)

(No drawing with text.)

	Extended Description	Module	Label
1	Pass control to the TABSET subcommand processor (IKJEBETA) from the controller routine (IKJEBEMA). If the user specified any operands, use Parse to check the syntax of the operands.		
2	Depending on what the user specified, either: • Indicate that translation of tabulation is not to be performed, if the OFF keyword was specified. • Set new values for tabulation characters. If the ON keyword was specified, turn on the tabulation switch. If an integer list was specified, store the values in the tabulation table (CATABS) in ascending order. If the user specified the IMAGE keyword with tab set characters, use GETLINE to obtain the tab set characters and store in the tabulation table.		
3	Return control to IKJEBEMA.		

Object Module: IKJEBETA

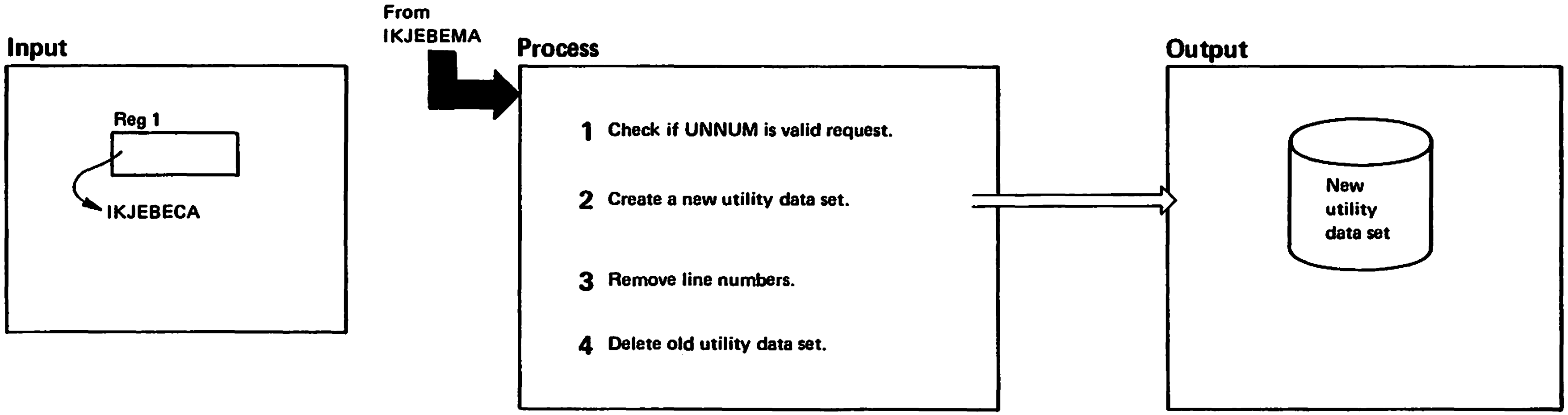
Diagram 1.31. TOP Subcommand Processing (Part 1 of 1)

(No drawing with text.)

	Extended Description	Module	Label
1	Pass control to the TOP subcommand processor (IKJEBETO) from the controller routine (IKJEBEMA).		
2	Request the EDIT Access Method interface (IKJEBEUT) to find and read the first record. Set the current line pointer to zero. If a record exists for line zero, turn on the "line to be verified" switch (CALNOTOVF).		
	If the data set is empty (return code 4), send message IKJ52501I to the user. Set the current line pointer to zero.		
3	Return control to IKJEBEMA.		

Object Modules: IKJEBETO, IKJEBEUT

Diagram 1.32. UNNUM Subcommand Processing (Part 1 of 2)



1 Check if UNNUM is valid request.

2 Create a new utility data set.

3 Remove line numbers.

4 Delete old utility data set.

Diagram 1.32. UNNUM Subcommand Processing (Part 2 of 2)

	Extended Description	Module	Label
1	If the data set is not numbered, return to caller. If the data set type must have line numbers, issue error message and return to caller. If the data set type requires a data exit routine for renumbering, issue error message and return to caller.		
2	Create (IKJEBEUI) a new utility data set.		
3	Remove line numbers by reading one record at a time (IKJEBEUT), setting the line number field to blanks if the record format is fixed or shifting data to overlay the line number field if the record format is variable, and writing (IKJEBEUT) the line on the new utility data set.		
4	Delete (IKJEBEEX) the old utility data set.		

Object Modules: IKJEBEEX, IKJEBEUI, IKJEBEUN, IKJEBEUT

Diagram 1.33. UP Subcommand Processing (Part 1 of 1)

(No drawing with text.)

Extended Description **Module** **Label**

1 Pass control to the UP subcommand processor (IKJEBEUP) from the controller routine (IKJEBEMA). Check the operand switch (CAOPERAND) to see if the user specified an operand on the subcommand. If the user specified an operand, use parse to check the syntax of the subcommand.

2 Depending upon what the user has specified, either:

- Read records toward the beginning of the utility data set until the number of records specified by the count operand has been read. Request the EDIT Access Method interface routine (IKJEBEUT) to locate and read the first record.

  If the data set is empty (return code 4), send message IKJ52502I to the user. Set the current line pointer to zero.

  If the data set is not empty, request IKJEBEUT to locate and read the record previous to the current one. Continue the process until the current line pointer has been moved up the requested (count operand) number of lines.

- If the user did not specify any operands, read the record before the one currently being pointed to. Processing is the same as above, except the record count is 1.

3 Set the current line pointer to the value of the last record read, and turn on the "line to be verified" switch.

4 Return control to IKJEBEMA.

Object Modules: IKJEBEUP, IKJEBEUT

Diagram 1.34. VERIFY Subcommand Processing (Part 1 of 1)

(No drawing with text.)

	Extended Description	Module	Label
1	Pass control to the VERIFY subcommand processor (IKJEBEVE) from the controller routine (IKJEBEMA). Check the operand switch (CAOPERAND) to see if the user specified an operand on the subcommand. If the user specified an operand, use parse to check the syntax of the subcommand.		
2	Depending on what the user specified, either: • Indicate that a record will be displayed when the text of the current line or the value of the current line pointer is changed, if no operands or the ON keyword was specified. Set the VERIFY switch (CAVRFYSW) to 1. • Indicate that a record will not be displayed, if the OFF keyword was specified. Set the VERIFY switch (CAVRFYSW) to 0.		
3	Return control to IKJEBEMA.		

Object Module: IKJEBEVE

Program Organization

This section contains a list of the EDIT modules and their functions, and charts showing the flow of control from one module to another.

Modules and Their Functions

IKJEBEAA	EDIT Utility Access Method
IKJEBEAE	EDIT ESTAE routine
IKJEBEAT	EDIT Attention handling routine
IKJEBEBO	BOTTOM subcommand processor
IKJEBECH	CHANGE subcommand processor
IKJEBECK	CKPOINT subcommand processor
IKJEBECI	Command invoker
IKJEBECO	Initial copy routine
IKJEBEDA	Data set allocation/free routine
IKJEBEDC	In-core copy routine for VS BASIC
IKJEBEDE	DELETE subcommand processor
IKJEBEDO	DOWN subcommand processor
IKJEBEDX	Data set interface routine for VS BASIC
IKJEBEEN	END subcommand processor
IKJEBEEX	EDIT Access Method final processing routine
IKJEBEFC	Final copy routine
IKJEBEFI	FIND subcommand processor
IKJEBEFO	FORMAT subcommand processor
IKJEBEHE	PROFILE/SEND/HELP/ALLOCATE subcommand processor
IKJEBEIA	Initialization message processing routine
IKJEBEIN	Initialization routine
IKJEBEIP	INPUT subcommand processor
IKJEBEIS	INSERT subcommand processor
IKJEBELE	Line editing routine
IKJEBELI	Line insert/replace/delete subcommand processor
IKJEBELT	LIST subcommand processor
IKJEBEMA	EDIT main line control routine
IKJEBEMC	MOVE/COPY subcommand processor
IKJEBEME	MERGE subcommand processor
IKJEBEMR	Merge data set translation routine
IKJEBEMS	Message selection routine
IKJEBEM1	Contains text for EDIT messages
IKJEBEM2	Contains text for EDIT messages
IKJEBEM3	Contains text for EDIT messages
IKJEBEM4	Contains text for EDIT messages
IKJEBEM5	Contains text for EDIT messages
IKJEBEM6	Contains text for EDIT messages
IKJEBEM7	Contains text for EDIT messages

IKJEBEPD	Processor data table
IKJEBEPS	Processor data table search routine
IKJEBERE	RENUM subcommand processor
IKJEBERN	BASIC renumber routine
IKJEBERU	RUN subcommand processor
IKJEBESA	SAVE subcommand processor
IKJEBESC	SCAN subcommand processor
IKJEBESE	String search routine
IKJEBESU	SUBMIT subcommand processor
IKJEBETA	TABSET subcommand processor
IKJEBETO	TOP subcommand processor
IKJEBEUI	EDIT Access Method initialization routine
IKJEBEUN	UNNUM subcommand processor
IKJEBEUP	UP subcommand processor
IKJEBEUT	EDIT Access Method interface routine
IKJEBEVE	VERIFY subcommand processor

Module Control Flow

Note: The first six characters in an EDIT module name are IKJEBE. The remaining two characters are used in Figures 1, 2, and 3.

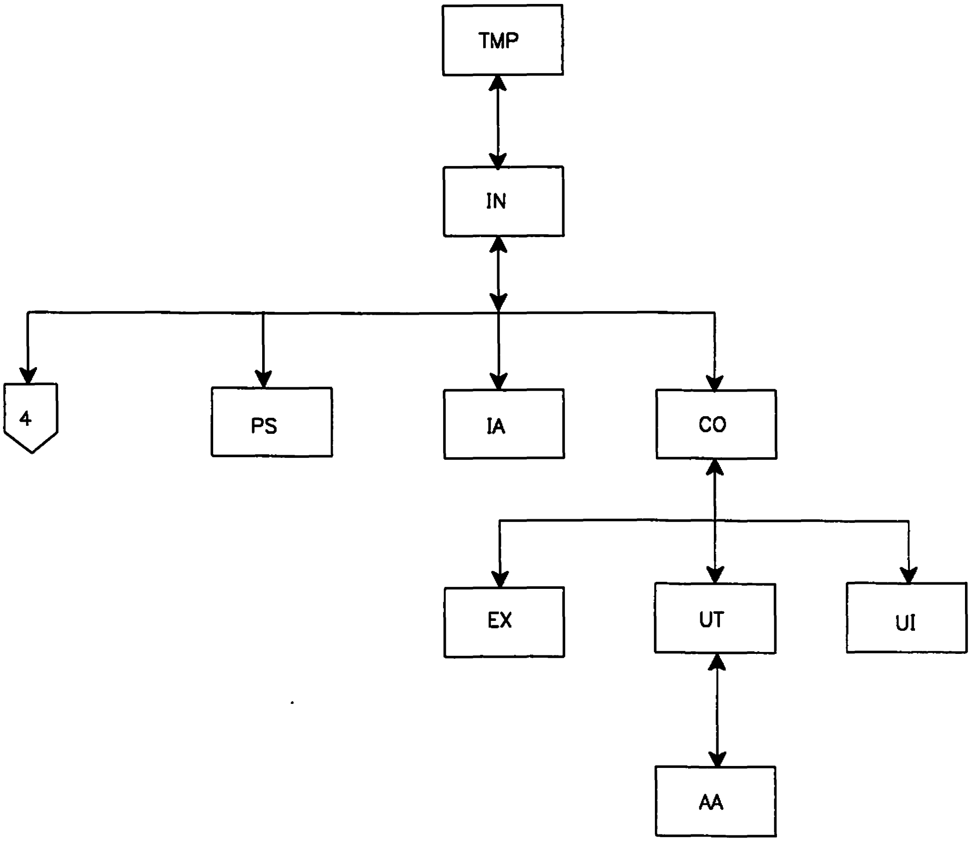


Figure 1-1. Mainline Module Control Flow

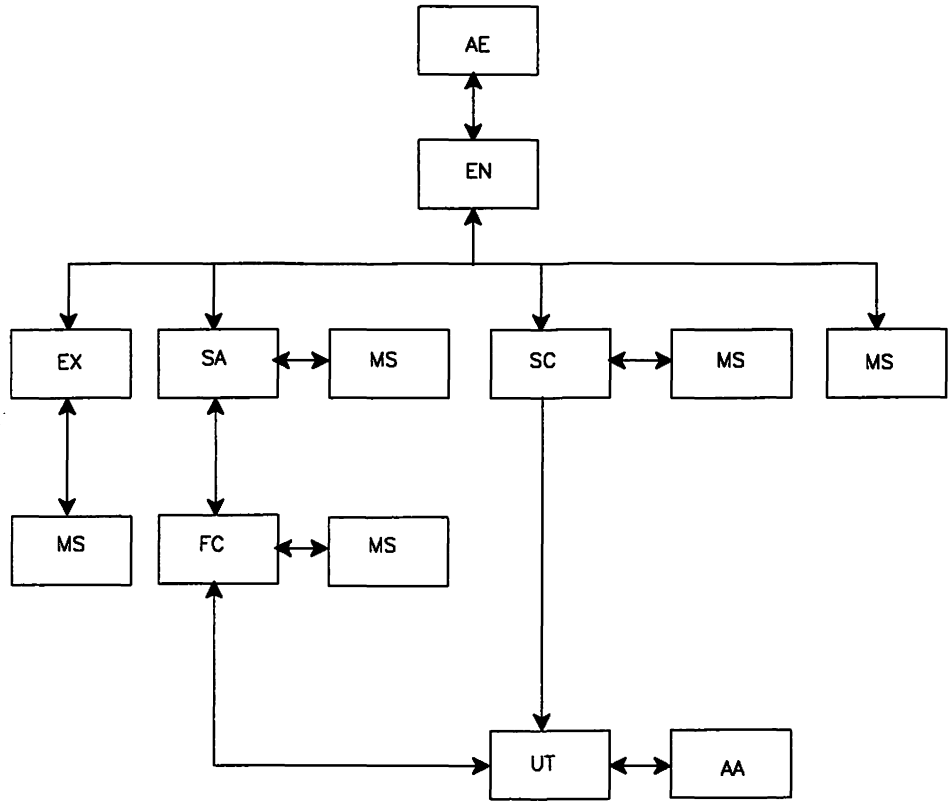


Figure 1-2. ESTAE Exit Module Control Flow

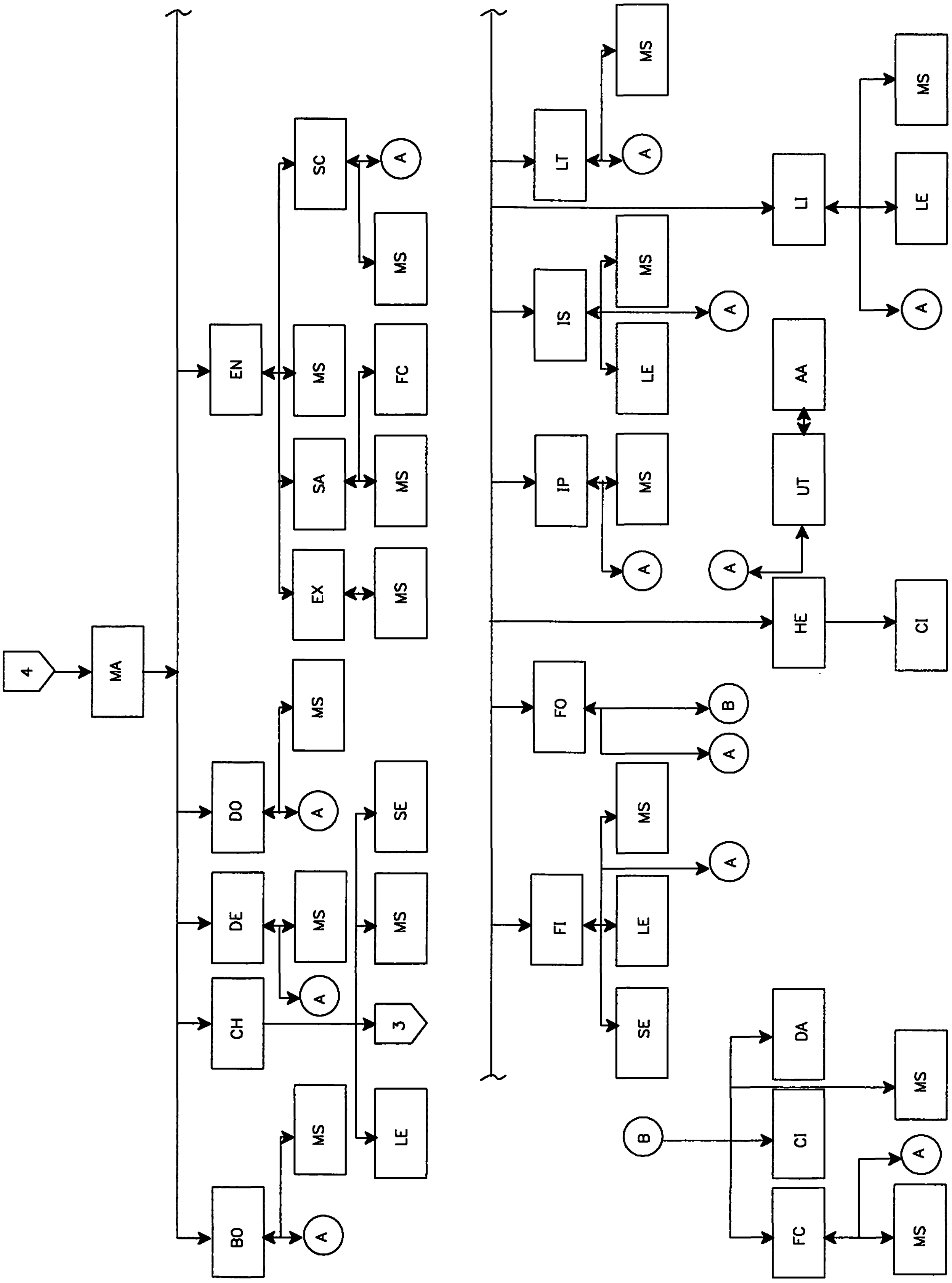
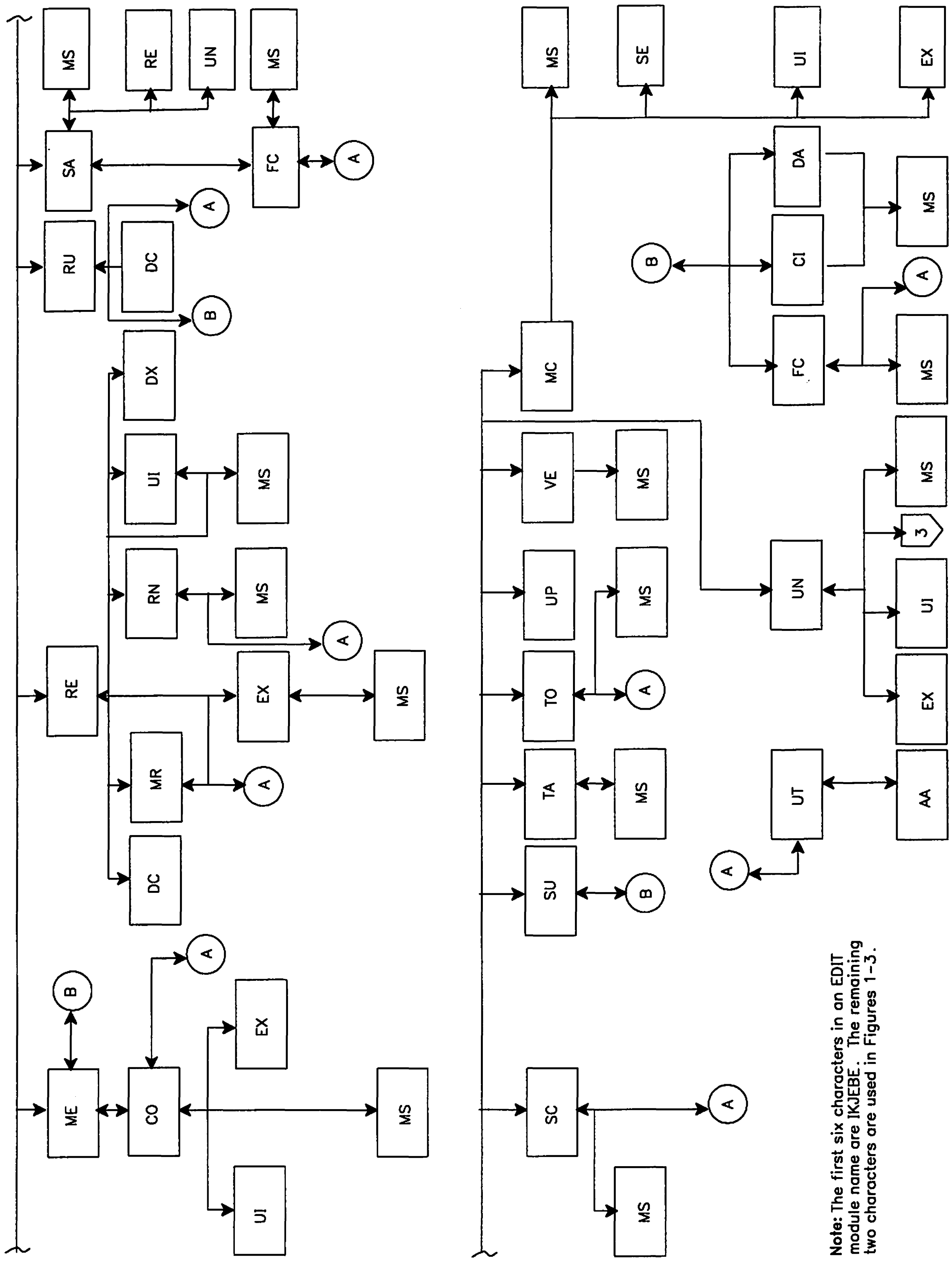


Figure 1-3 (Part 1 of 2). EDIT Subcommands Control Flow



Note: The first six characters in an EDIT module name are IKJEBE. The remaining two characters are used in Figures 1-3.

Figure 1-3 (Part 2 of 2). EDIT Subcommands Control Flow

Diagnosing an EDIT Problem

As with other TSO command processors, you may diagnose a problem with the EDIT command by issuing the TEST command. For an example of how to use the TEST command to debug an EDIT problem, refer to *TSO Extensions System Diagnosis: Guide and Index.*

TSO Extensions System Diagnosis: Guide and Index also provides information about reading messages and dumps, issuing traces and traps, and calling IBM to report a problem you are unable to fix.

Chapter 2. EXEC Command Processing

The EXEC command executes TSO/E commands and CLISTs. For a complete description of EXEC command processing and diagnosis, see the "CLIST Processing and Diagnosis" chapter in *TSO Extensions System Diagnosis: Terminal Monitor Program and Service Routines.*

Chapter 3. FREE Command Processing

This section describes the logic of the **FREE** command. It emphasizes the flow of data and control information through buffers and tables, and contains detailed functional descriptions through the use of method of operations diagrams. Figure 3-1 is the visual table of contents for the FREE command.

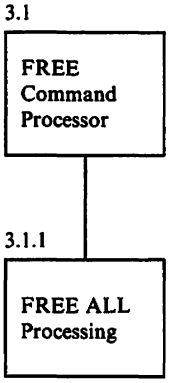
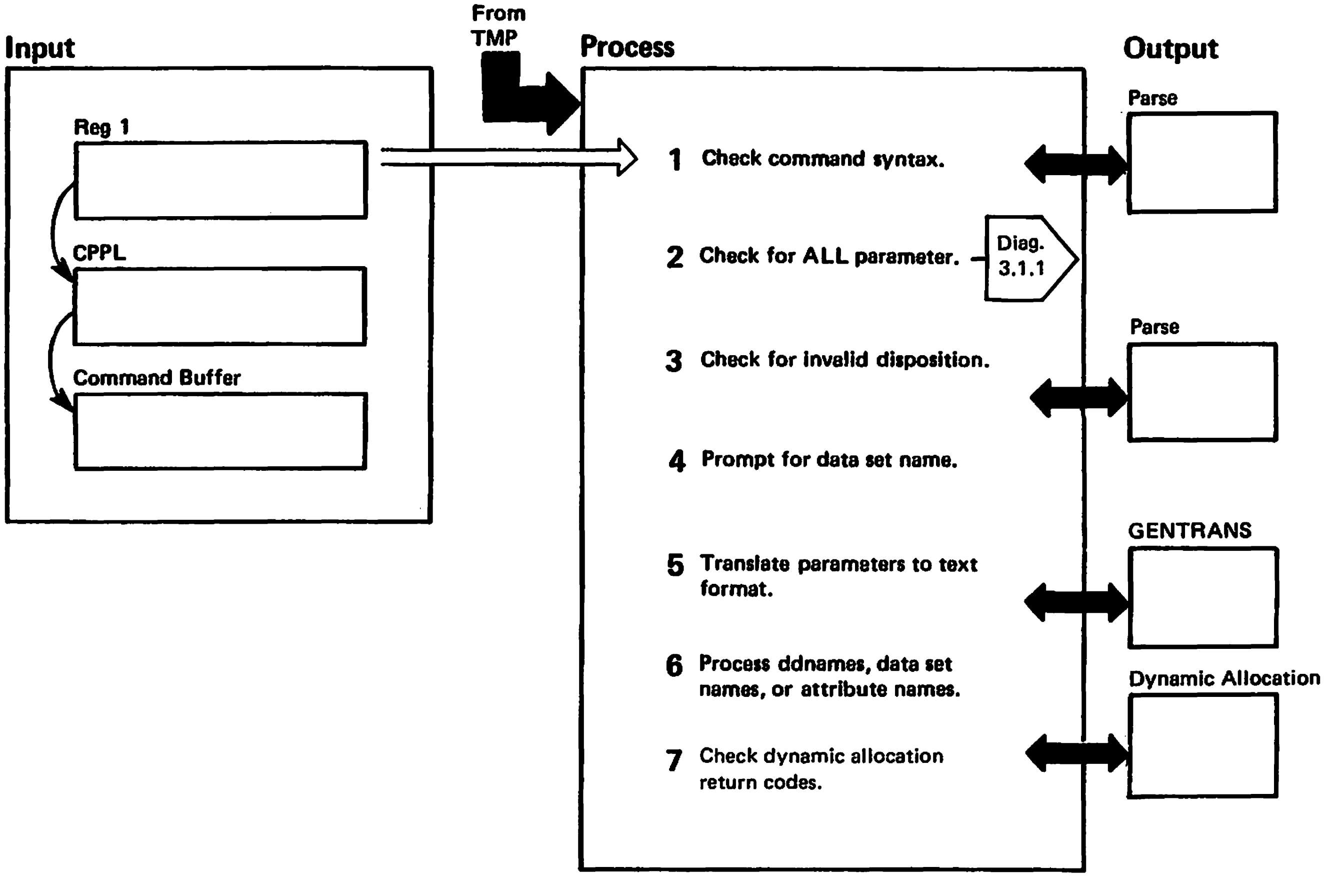


Figure 3-1. FREE Command Processing Visual Table of Contents

Diagram 3.1. FREE Command Processing (Part 1 of 2)



1 Check command syntax.

2 Check for ALL parameter.

3 Check for invalid disposition.

4 Prompt for data set name.

5 Translate parameters to text format.

6 Process ddnames, data set names, or attribute names.

7 Check dynamic allocation return codes.

Diagram 3.1. FREE Command Processing (Part 2 of 2)

Extended Description **Module** **Label**

1 Use parse to syntax check the command. Check the parse return code; if it is non-zero return to the TMP.

2 Check to see if the ALL parameter was specified. If so, unallocate all dynamically allocated file names.

3 Check to see if DEST, HOLD, or NOHOLD was specified with a data set disposition of KEEP, DELETE, CATALOG, or UNCATALOG. If yes, issue an error message and return to the TMP.

4 Check to see if a data set name, file name, or attribute list name was entered. If no, pass control to parse and prompt the user for a data set name. When prompting is complete, overlay the original PDE with the new PDE from prompt.

5 Use GENTRANS to translate the parameters to text format. The pointer to the text unit is returned from GENTRANS in the IKJZB831 parameter list. If the return code is non-zero, return control to the TMP.

6 Use the unallocate function of dynamic allocation to unallocate files, data sets, or attribute lists.

7 Check the dynamic allocation error code and information reason code.

- If both codes are zero, a file, data set, or attribute list was unallocated.
- If either code was non-zero, unable to unallocate. Use the DAIR failure message routine IKJEFF18 to analyze the error code and send the appropriate error message to the user.

  FREE command processing terminates.

Control is returned to the TMP.

Object Module: IKJEFD20

Diagram 3.1.1. FREE ALL Processing (Part 1 of 2)

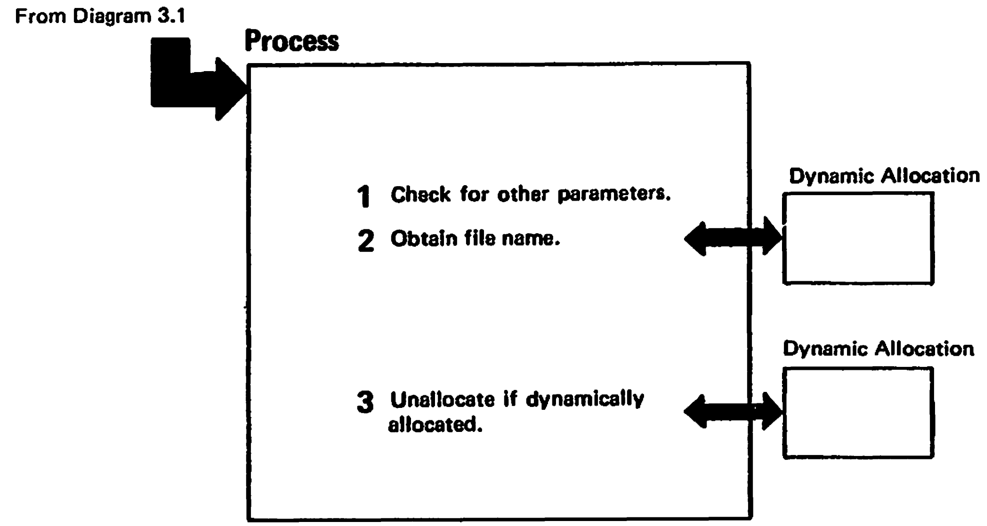


Diagram 3.1.1. FREE ALL Processing (Part 2 of 2)

	Extended Description	Module	Label
1	Check to see if any other parameters were specified on the command. If so, issue an error message and terminate the command.		
2	Invoke the RETRIEVE function of dynamic allocation specifying the relative request number to obtain the associated file name. If the error code from dynamic allocation is non-zero, use the DAIR failure message routine IKJEFF18 to analyze the error code and send the appropriate error message to the user.		
3	Determine if the file was dynamically allocated using information returned by dynamic allocation. If so, use the UNALLOCATE function of dynamic allocation to unallocate the file. If the return code from dynamic allocation is non-zero, use the DAIR failure message routine IKJEFF18 to analyze the return code and send the appropriate error message to the user. Repeat step 2, specifying the next relative request number.		

Object Module: IKJEFD38

Chapter 4. HELP Command Processing

This section describes the logic of the **HELP** command. It emphasizes the flow of data and control information through buffers and tables, and contains detailed functional descriptions through the use of method of operations diagrams. Figure 4-1 is the visual table of contents for the HELP command.

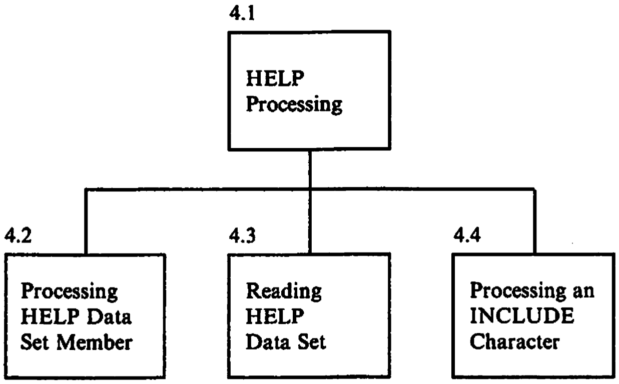
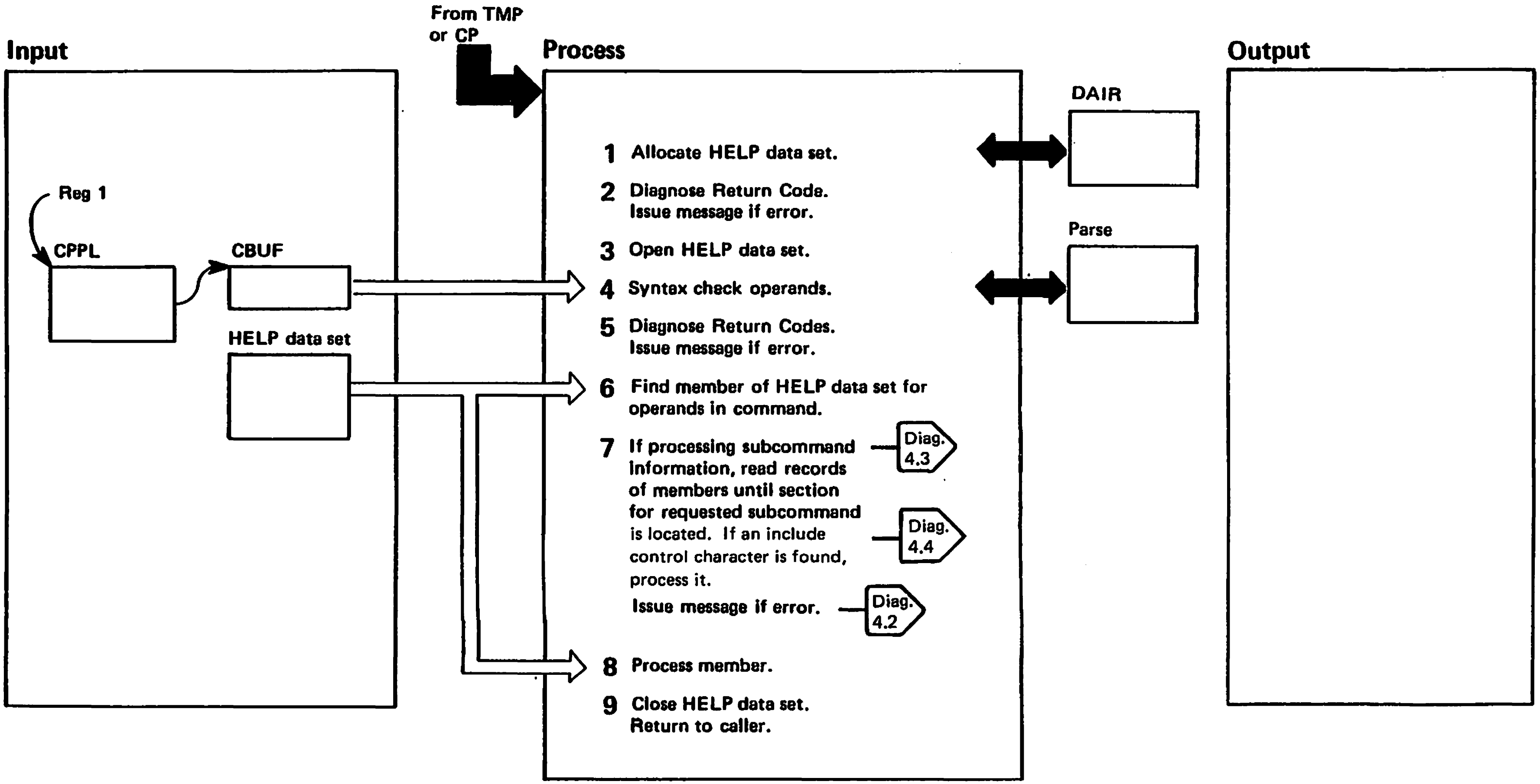

4.1
HELP Processing
4.2
Processing HELP Data Set Member
4.3
Reading HELP Data Set
4.4
Processing an INCLUDE Character

Figure 4-1. HELP Command Processing Visual Table of Contents

Diagram 4.1. HELP Processing (Part 1 of 2)



1 Allocate HELP data set.

2 Diagnose Return Code. Issue message if error.

3 Open HELP data set.

4 Syntax check operands.

5 Diagnose Return Codes. Issue message if error.

6 Find member of HELP data set for operands in command.

7 If processing subcommand information, read records of members until section for requested subcommand is located. If an include control character is found, process it.
Issue message if error.

8 Process member.

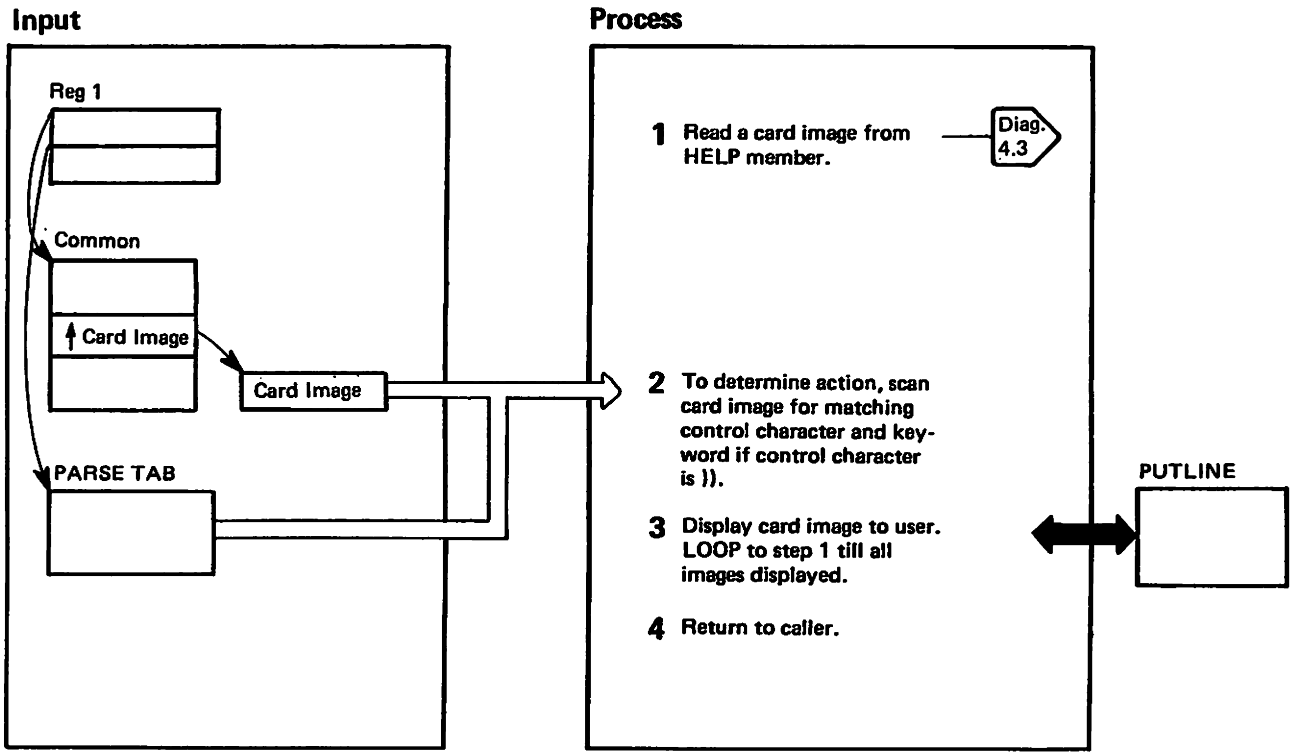
9 Close HELP data set. Return to caller.

Diagram 4.1. HELP Processing (Part 2 of 2)

	Extended Description	Module	Label
1	Using DAIR (the dynamic allocation interface routine), allocate the HELP data set.		
2	Check return code: a. If non-zero, use IKJEFF18 (DAIRFAIL) to diagnose error and send message to user, return caller. b. If zero and DSORG is not PO (partitioned), issue message to user, return to caller.		
3	Open HELP data set. If open fails, issue message and return to caller.		
4	Use parse to check syntax of command.		
5	If parse was unsuccessful, issue messages and return to caller.		
6	FIND member of HELP data set. a. If not in subcommand mode (command attached by the TMP is HELP or H), and HELP entered with no operands, find member 'COMMANDS'. b. If not in subcommand mode and HELP entered with operands, use first operand as member name. c. If in subcommand mode and HELP is first operand, find member 'HELP'. d. If in subcommand mode and HELP is not first operand, use the name of the command attached by TMP (from ECT) as the member name.		
7	If case d) of step 6, read record from member of HELP data set. See Diagram 4.3. Search each record for subcommand name indicator '=' and then for subcommand name requested on command. Keep reading records until end-of-file (error) or subcommand name found. If an include control character is found, process it. See Diagram 4.4.		
8	Process the HELP data set member. See Diagram 4.2.		
9	Close and free the HELP data set. Return to caller.		

Object Module: IKJEFH01

Diagram 4.2. Processing HELP Data Set Member (Part 1 of 2)



1 Read a card image from HELP member.

2 To determine action, scan card image for matching control character and keyword if control character is)).

3 Display card image to user. LOOP to step 1 till all images displayed.

4 Return to caller.

Diagram 4.2. Processing HELP Data Set Member (Part 2 of 2)

Extended Description **Module** **Label**

1 Obtain a card image from the HELP data set member. See Diagram 4.3.

2 Scan card image for a matching control character. (If the matching control character is)), also scan for a matching keyword.) If the matching control character is)I, then process the include request. (See Diagram 4.4.)

The control characters are:

)S - list of commands or subcommands
)F - functional information about command or subcommand
)X - syntactical information about command or subcommand
)M - message ID information
)O - command operand information
)P - positional parameter information
))'keyword' - information about 'keyword'
)I - include a member

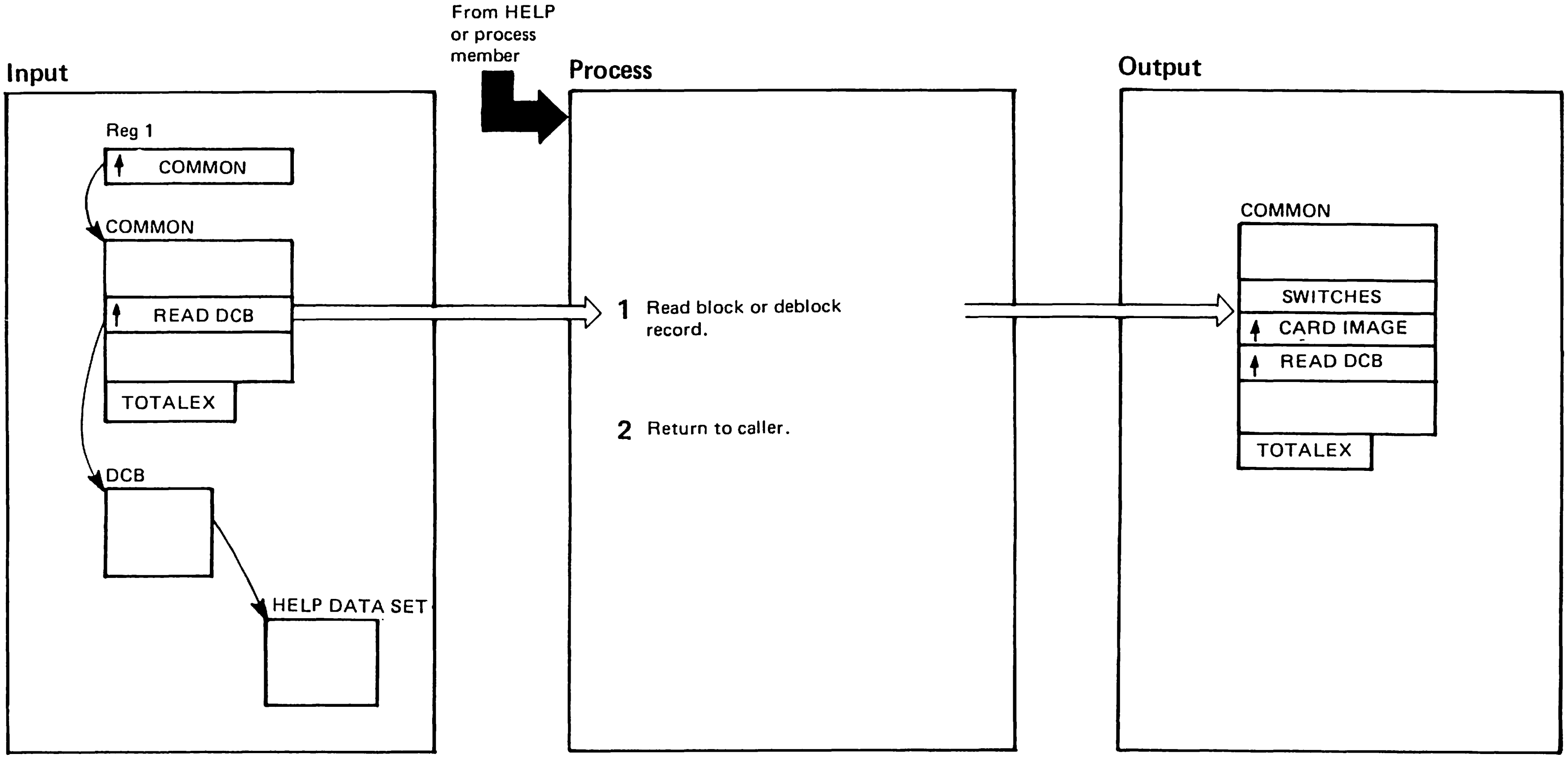
Compare the control information on the card image with the control character information maintained by the HELP command processor that indicates the information requested by the user. If a match is found, proceed to step 3 below. If no match is found, repeat step 1.

3 If scan finds match, use PUTLINE via IKJEFF02 to display card image to user.

4 If all information has been displayed, return to caller; else process step 1 again.

Object Module: IKJEFH02

Diagram 4.3. Reading HELP Data Set (Part 1 of 2)



1 Read block or deblock record.

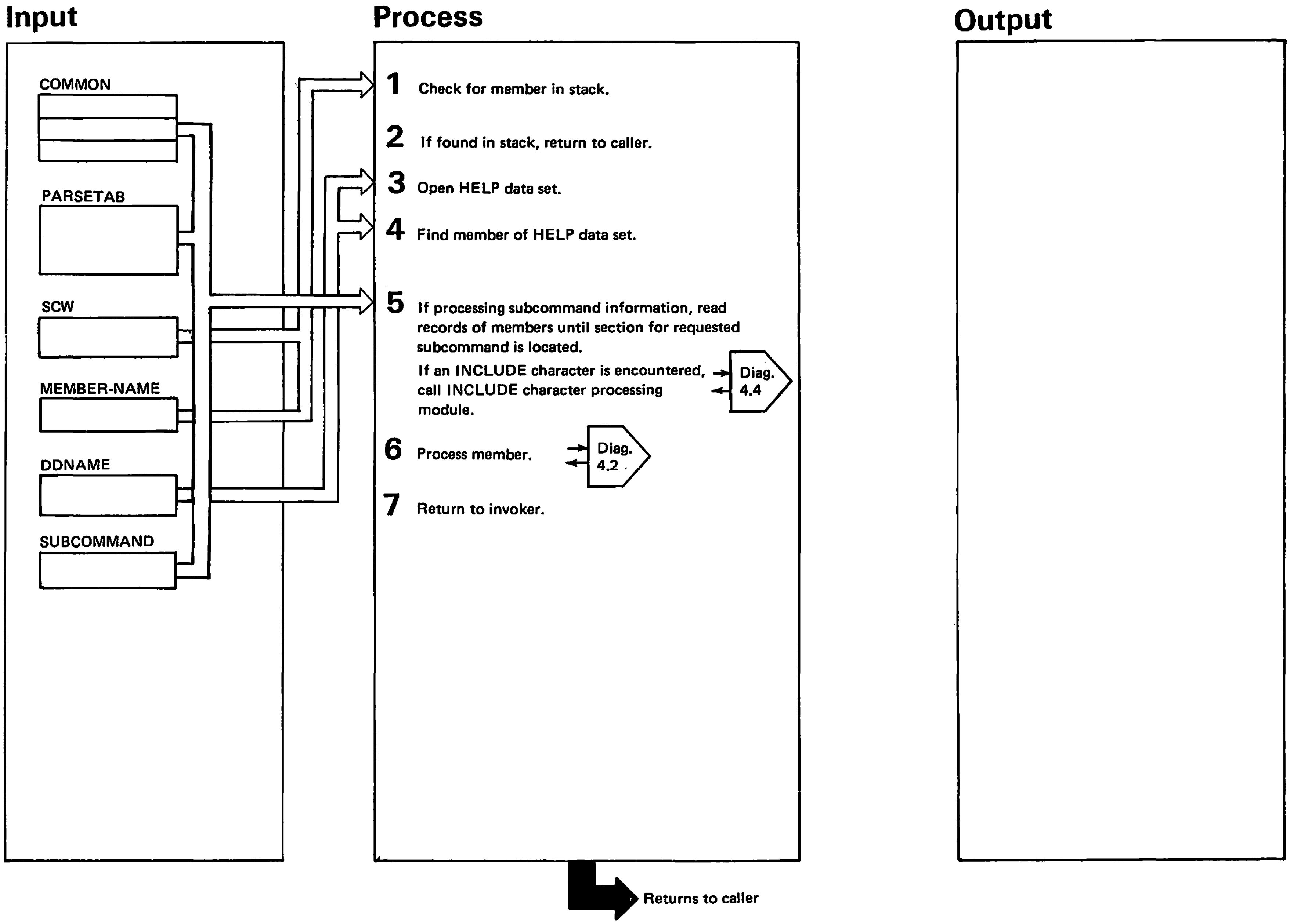
2 Return to caller.

Diagram 4.3. Reading HELP Data Set (Part 2 of 2)

	Extended Description	Module	Label
1	The HELP data is blocked. Read a block and deblock a record or deblock a card image record. In case of I/O error or end of data, set switches.		
2	Return to caller.		

Object Module: IKJEFH03

Diagram 4.4. Processing an INCLUDE Character (Part 1 of 2)



1 Check for member in stack.

2 If found in stack, return to caller.

3 Open HELP data set.

4 Find member of HELP data set.

5 If processing subcommand information, read records of members until section for requested subcommand is located.

If an INCLUDE character is encountered, call INCLUDE character processing module.

6 Process member.

7 Return to invoker.

Diagram 4.4. Processing an INCLUDE Character (Part 2 of 2)

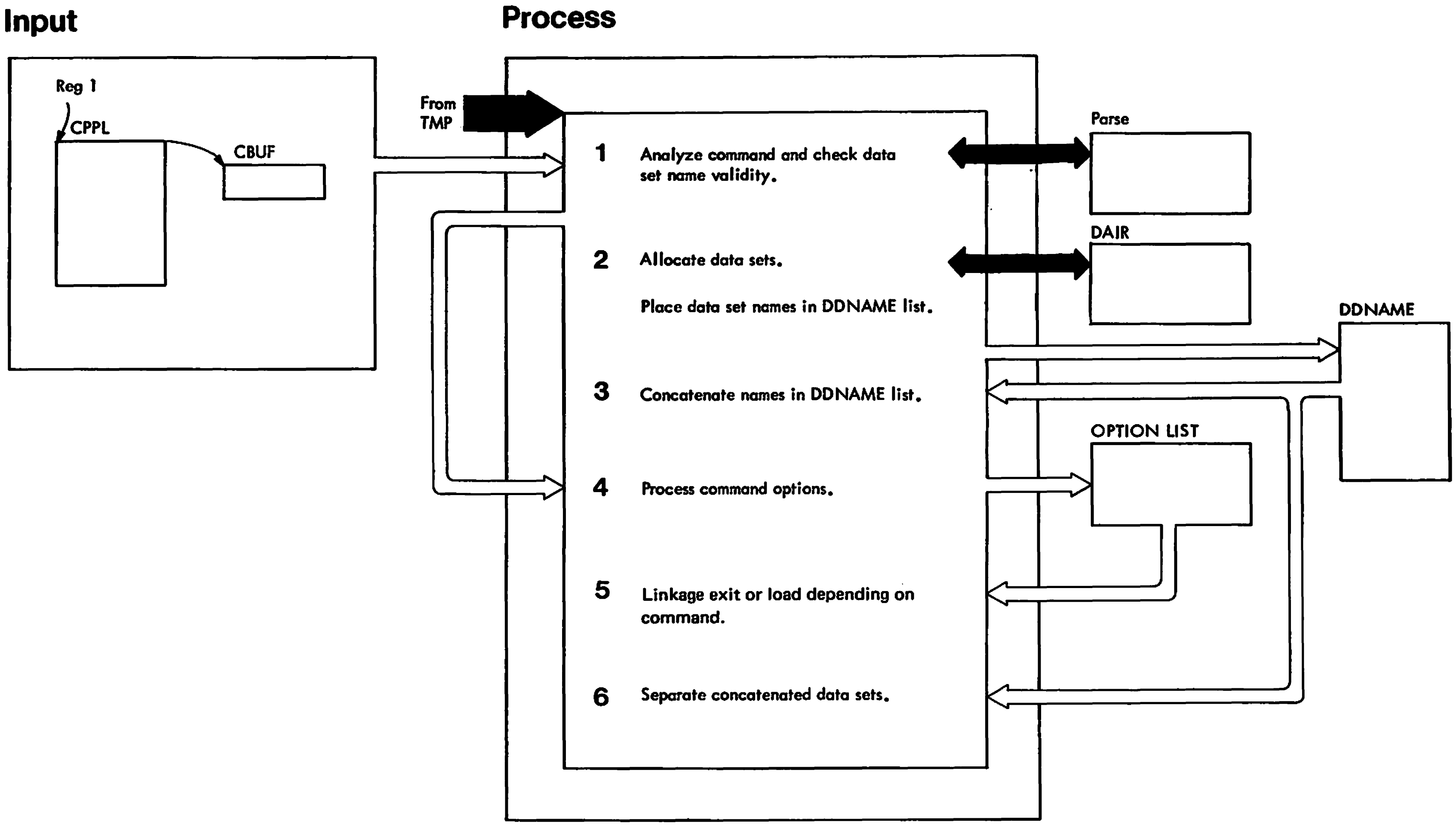
	Extended Description	Module	Label
1	Check list (stack) of member names for the member being included. This prevents recursive includes.		
2	Finding a member name in the stack indicates a recursive include. If it is a recursive include, return to the invoker. If it is not a recursive include, continue processing.		
3	Open the HELP data set.		
4	Issue a FIND for the member. If found, place the member in the stack and continue. If not found, close the data set and return to the invoker.		
5	If in subcommand mode, scan the member searching for the subcommand indicator (=). If a subcommand indicator is found, check if it is the subcommand requested. If it is the requested subcommand, then process. If not, continue search. If an include character is found while searching for a subcommand, set up all parameters and process the include character (Diagram 4.4). Otherwise, process.		
6	If process is indicated, then process the member (Diagram 4.2).		
7	If the data set was opened, close it. If the member was placed in stack, remove it. Return to the invoker.		

Object Module: IKJEFH04

Chapter 5. LINK/LOADGO Command Processing

This section describes the logic of the **LINK/LOADGO** commands. It emphasizes the flow of data and control information through buffers and tables, and contains detailed functional descriptions through the use of method of operations diagrams.

Diagram 5.1. LINK and LOADGO Processing (Part 1 of 2)



1 Analyze command and check data set name validity.

2 Allocate data sets.

Place data set names in DDNAME list.

3 Concatenate names in DDNAME list.

4 Process command options.

5 Linkage exit or load depending on command.

6 Separate concatenated data sets.

Diagram 5.1. LINK and LOADGO Processing (Part 2 of 2)

	Extended Description	Module	Label
1	Use parse to analyze syntax of commands. Check data set names for valid qualifiers. Set LKLD to indicate whether command is LINK or LOADGO.		
2	Use DAIR to allocate data sets. Place each data set name in DDNAME list.		
3	Use DAIR to concatenate ddnames in ddname list. (DDNMS)		
4	Using parse output, process command options. Prompt for missing operands, set defaults. Place results in option list. (OPLEN)		
5	Link to either the linkage editor or the loader depending on LKLD switch passing the option list and ddname list through OUTPARM.		
6	On return, separate concatenated data sets and return to the TMP.		

Chapter 6. LISTALC Command Processing

This section describes the logic of the **LISTALC** command. It emphasizes the flow of data and control information through buffers and tables, and contains detailed functional descriptions through the use of method of operations diagrams. Figure 6-1 is the visual table of contents for the LISTALC command.

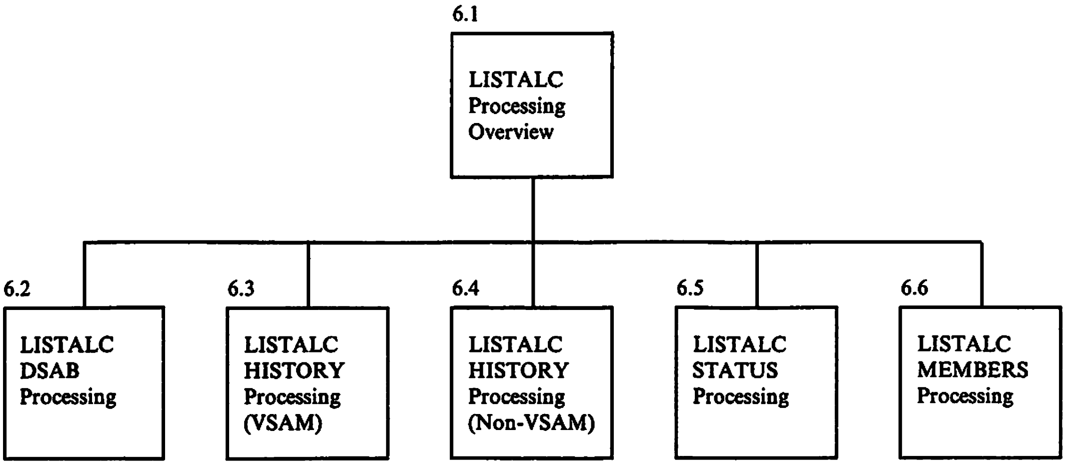
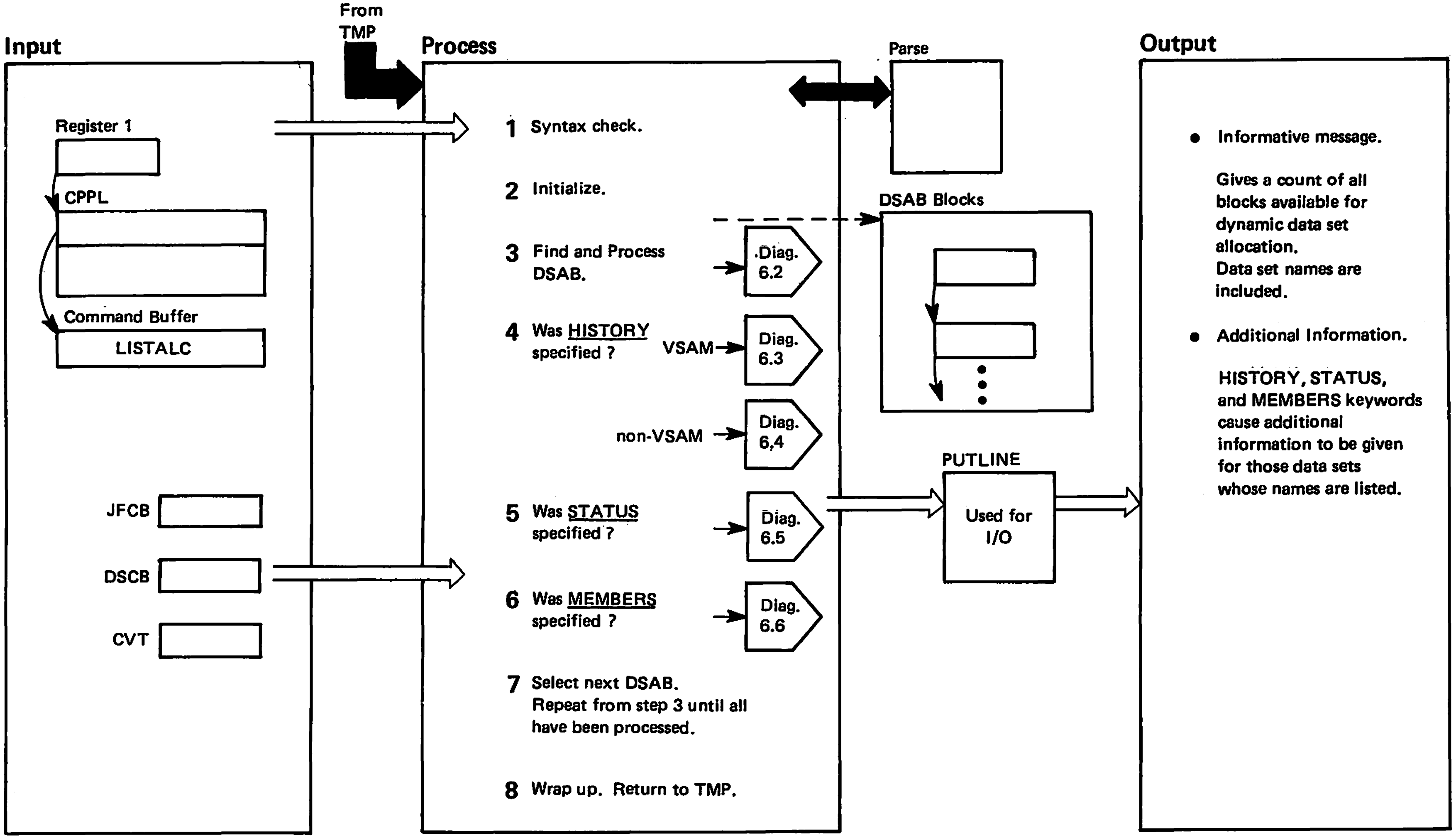

6.1
LISTALC Processing Overview
6.2
LISTALC DSAB Processing
6.3
LISTALC HISTORY Processing (VSAM)
6.4
LISTALC HISTORY Processing (Non-VSAM)
6.5
LISTALC STATUS Processing
6.6
LISTALC MEMBERS Processing

Figure 6-1. LISTALC Command Processing Visual Table of Contents

Diagram 6.1. LISTALC Processing Overview (Part 1 of 2)



1 Syntax check.

2 Initialize.

3 Find and Process DSAB.

4 Was HISTORY specified ?

5 Was STATUS specified ?

6 Was MEMBERS specified ?

7 Select next DSAB. Repeat from step 3 until all have been processed.

8 Wrap up. Return to TMP.

- Informative message.

  Gives a count of all blocks available for dynamic data set allocation. Data set names are included.

- Additional Information.

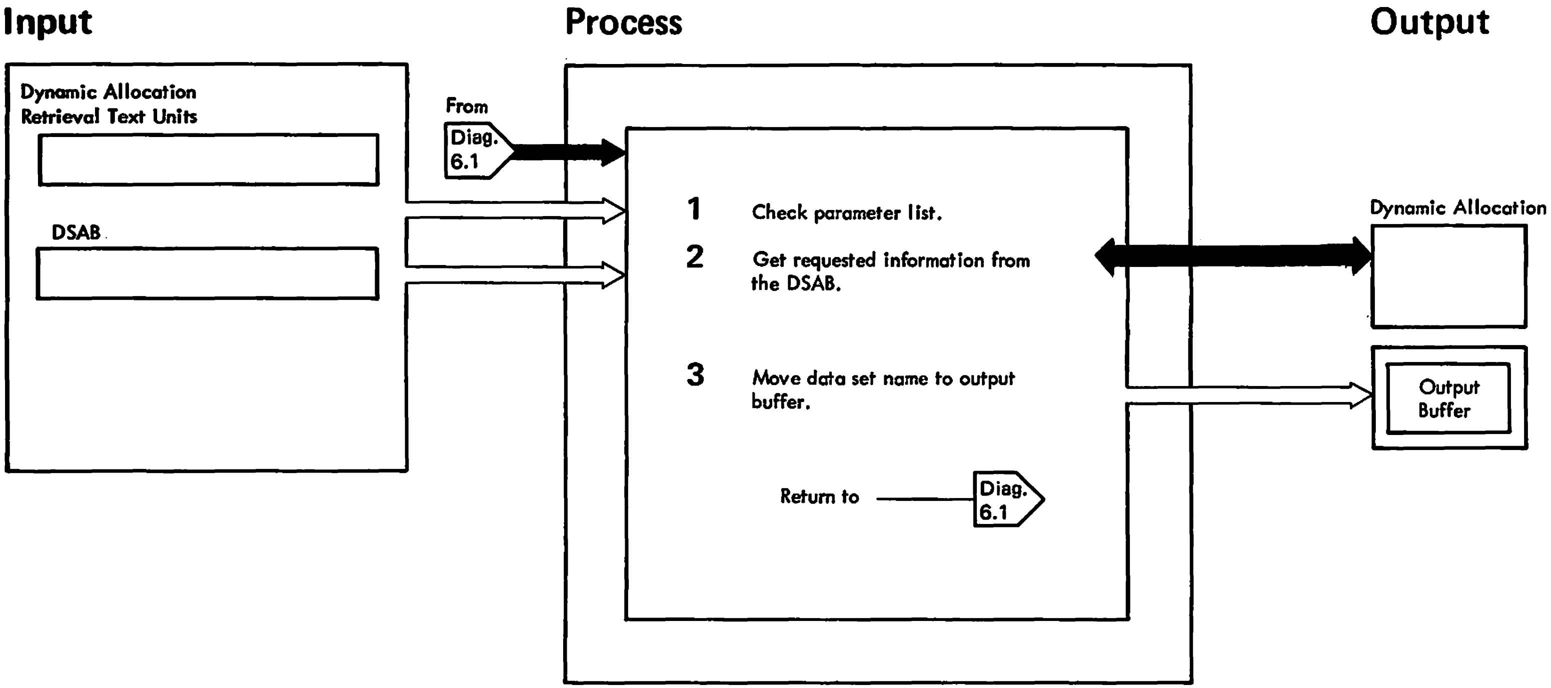
  HISTORY, STATUS, and MEMBERS keywords cause additional information to be given for those data sets whose names are listed.

Diagram 6.1. LISTALC Processing Overview (Part 2 of 2)

	Extended Description	Module	Label
1	The parse routine syntax checks the command. Upon return, the parse return code is checked. *Possible messages: IKJ58304I, IKJ58305I*		
2	Set option byte to reflect options selected by user. If HISTORY, MEMBERS, or SYSNAMES were specified, get work area and place address in OBTWA. Store JFCB work area address. Store DCB address. *Possible message: IKJ58303I*		
3	Obtain a pointer to the data set attribute block (DSAB) chain through SVC99. After the DSAB is located, check for HISTORY and STATUS and print applicable headings. Then check the DSAB to see if it is available for allocation. The DSAB is considered available if the data set is not in use and not permanently allocated. This condition is indicated on the output line by an asterisk (*) preceding the data set name.		
4	After basic processing of a DSAB, check to see if HISTORY was requested. If yes, process HISTORY information. See Diagram 6.3 (VSAM) or 11.4 (Non-VSAM).		
5	If STATUS was specified, process STATUS information, see Diagram 6.5.		
6	Write HISTORY and/or STATUS information, if applicable. *Possible messages: IKJ58301I, IKJ58300I*		
7	After all processing of the DSAB is complete, process the next DSAB. If no DSABs remain to be processed, return control to the TMP.		

Object Module: IKJEHAL1

Diagram 6.2. LISTALC DSAB Processing (Part 1 of 2)



1 Check parameter list.

2 Get requested information from the DSAB.

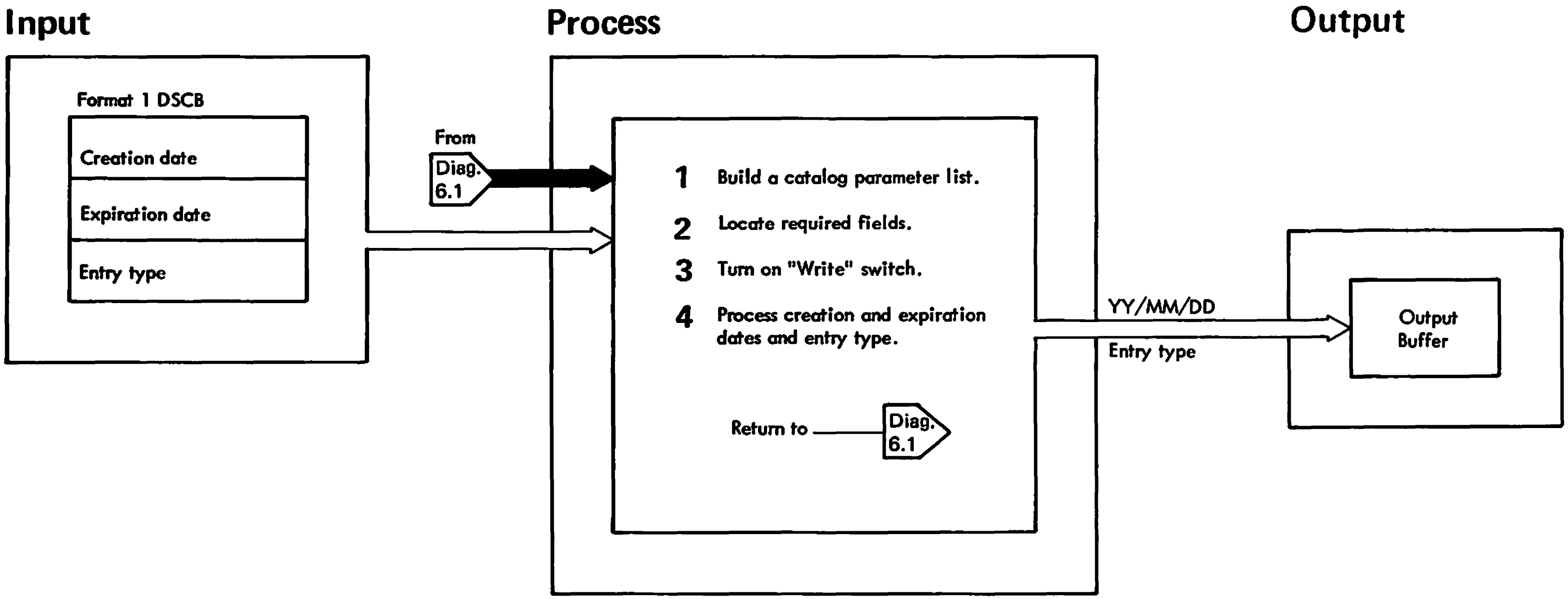
3 Move data set name to output buffer.

Diagram 6.2. LISTALC DSAB Processing (Part 2 of 2)

	Extended Description	Module	Label
1	Check the dynamic allocation parameter list (IEFZB4D0) to see if it has been initialized for use by dynamic allocation. If it is initialized continue processing. If not, build dynamic allocation text units describing the data to be returned about each allocated data set.		
2	Use dynamic allocation to get information requested by the text units from the DSAB.		
3	Get data set name from the DSAB and move it to the output buffer. If a data set name is not available, put the appropriate message in the output buffer.		

Object Module: IKJEHAL1

Diagram 6.3. LISTALC HISTORY Processing (VSAM) (Part 1 of 2)



1 Build a catalog parameter list.

2 Locate required fields.

3 Turn on "Write" switch.

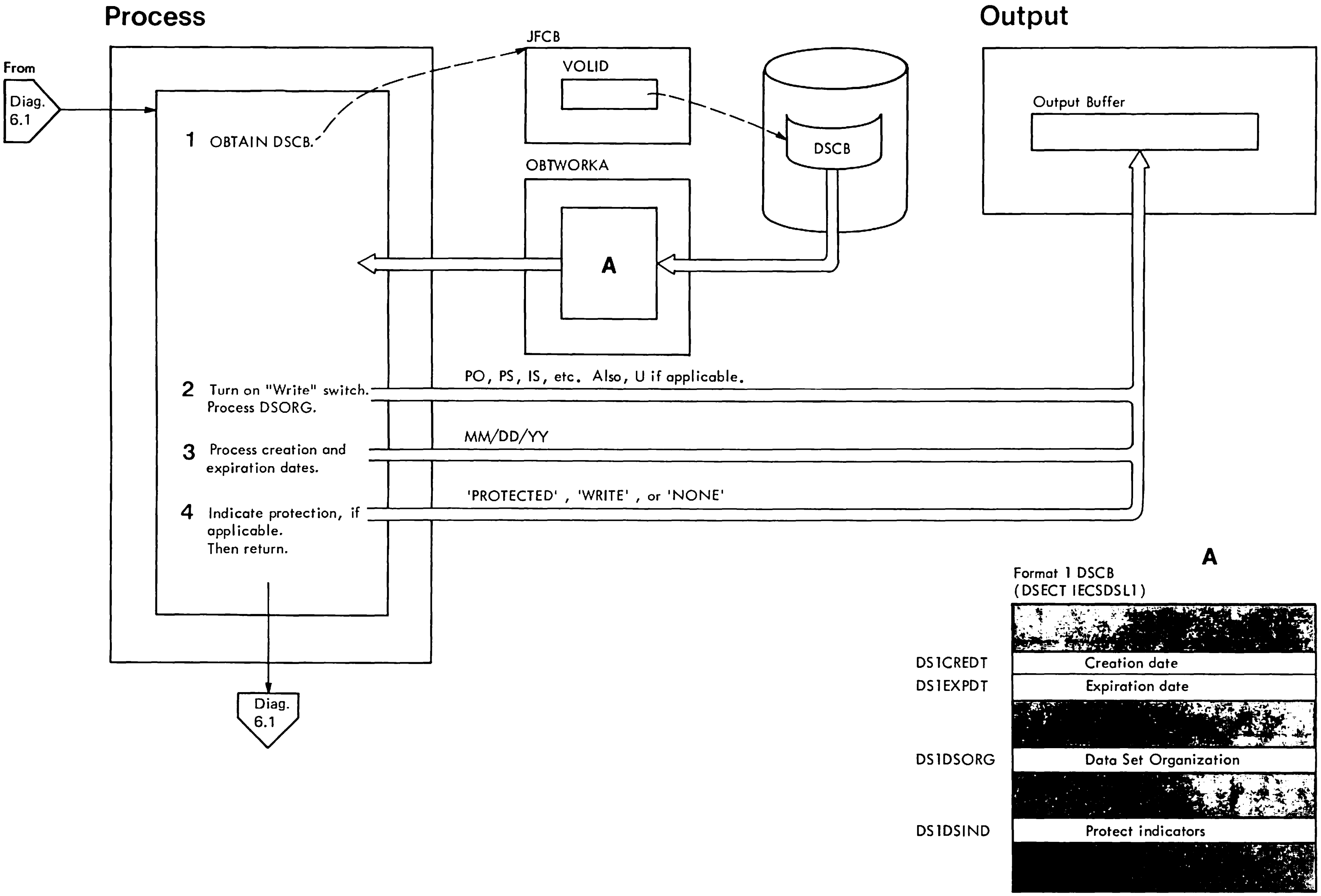
4 Process creation and expiration dates and entry type.

Diagram 6.3. LISTALC HISTORY Processing (VSAM) (Part 2 of 2)

	Extended Description	Module	Label
1	Build a catalog parameter list using information and the data set name obtained from the DSAB by SVC99. The parameter list specifies the named data set to be retrieved from the VSAM catalog, the entry type (indicates VSAM or non-VSAM data set) and the expiration and creation dates for the data set are to be returned in the work area.		
2	Locate the required fields in the CTGPL and the CTGFL to return expiration date, creation date, and entry type.		
3	Pass control to IKJEHVHS, the VSAM HISTORY processing routine, which turns on the "Write" switch to indicate that the buffer should be written when all options have been processed.		
4	Convert creation and expiration dates into MM/DD/YY format and move them and the entry type to the output buffer.		

Object Module: IKJEHAL1
CSECT: IKJEHHST

Diagram 6.4. LISTALC HISTORY Processing (Non-VSAM) (Part 1 of 2)



1 OBTAIN DSCB.

2 Turn on "Write" switch. Process DSORG.

3 Process creation and expiration dates.

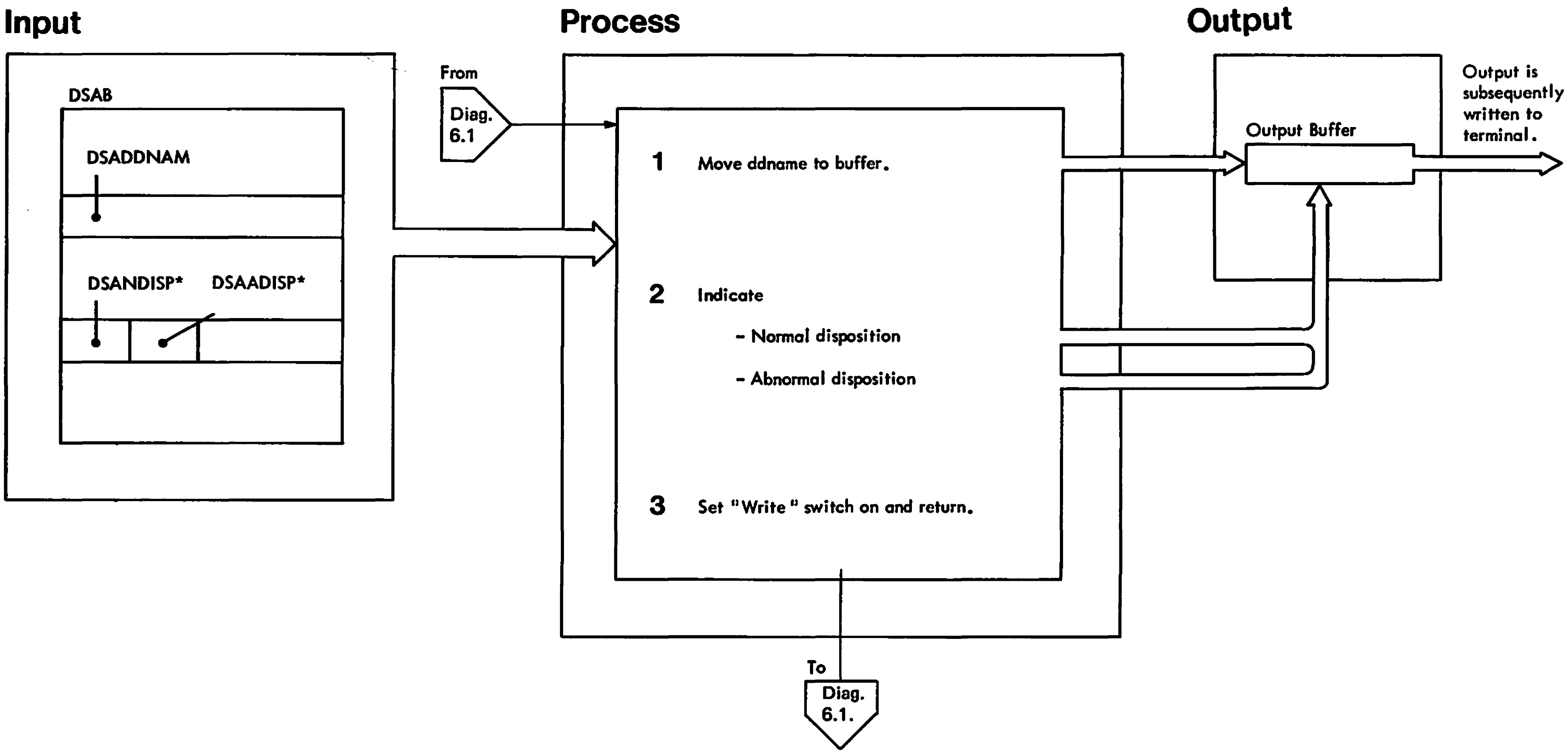
4 Indicate protection, if applicable. Then return.

Diagram 6.4. LISTALC HISTORY Processing (Non-VSAM) (Part 2 of 2)

	Extended Description	Module	Label
1	Check the DSADDNAM field in the DSAB for blanks. If the field contains blanks, it it part of a concatenation. Issue LOCATE to find the volume serial for the OBTAIN; otherwise, issue a RDJFCB to find the volume serial of the volume containing the DSCB. Issue an OBTAIN macro instruction.		
2	Then pass control to IKJEHHST, the HISTORY processing routine, which turns on the "Write" switch and checks for data set organization. The organization indicator (PO, PS, etc.) is then placed in the buffer, along with the unmovable (U), if applicable.		
3	Creation and expiration date are converted into MM/DD/YY format and placed in the buffer.		
4	A check is made for password protection. If none, a check is made for write protection. The applicable indication is placed in the buffer. Control is then returned to IKJEHAL1, where a check is made for STATUS processing. (If none, the write switch is checked and found "on," then the buffer is written via PUTLINE; if STATUS processing is applicable, the STATUS information is placed in the buffer prior to checking the write switch.)		

Object Module: IKJEHAL1
CSECT: IKJEHHST

Diagram 6.5. LISTALC STATUS Processing (Part 1 of 2)



1 Move ddname to buffer.

2 Indicate

- Normal disposition
- Abnormal disposition

3 Set "Write" switch on and return.

Diagram 6.5. LISTALC STATUS Processing (Part 2 of 2)

	Extended Description	Module	Label
1	A check is made to see if HISTORY information is contained in the output buffer area. If so, this will affect the pointer to the proper location in the buffer. The ddname in field DSADDNAM is moved to the output buffer area. The ddname can be blanks if the data set was concatenated.		
2	Then a test is made for normal disposition. The appropriate indication is placed in the output buffer. The output buffer pointer is then updated to point to the next location. A test is now made for disposition in the event of an abnormal termination. The appropriate disposition is moved to the buffer.		
3	The 'Write' switch is turned on and control is returned to IKJEHAL1.		

Diagram 6.6. LISTALC MEMBERS Processing (Part 1 of 2)

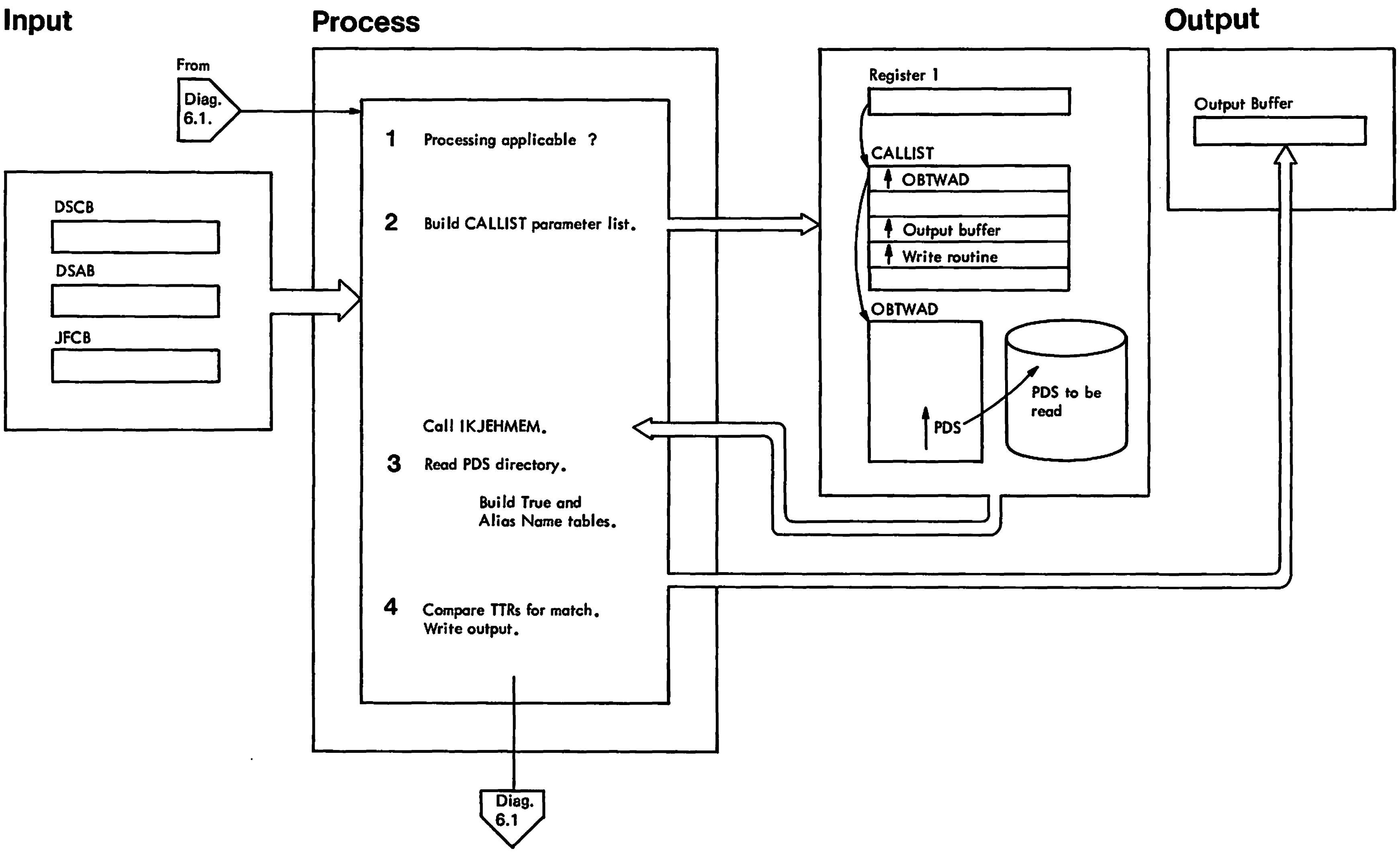


Diagram 6.6. LISTALC MEMBERS Processing (Part 2 of 2)

Extended Description **Module** **Label**

1 IKJEHMMR makes a number of checks, prior to passing control to IKJEHMEM, to ensure that processing is applicable.

- DSORG in the DSCB is checked to ensure that it is PO.
- This user's user ID must be the first qualifier in the data set name.
- The ddname cannot be blank (which would indicate concatenation).
- The dynamic concatenation bit in the DSAB is checked. If on, DSADDNAM is compared to DCBDDNAM. If unequal, MEMBERS processing can continue. (If equal, this is at least the second data set of a concatenation cluster.)

2 A RDJFCB is issued (unless HISTORY was also specified, in which case it is not required), the CALLIST parameter list for IKJEHMEM is constructed, and control is passed to IKJEHMEM.

3 IKJEHMEM initializes the true and alias name tables, then reads the PDS directory into them. Name blocks are obtained and chained dynamically, as required.

4 A true name is moved to the output buffer. The true name TTR is compared with all of the alias name TTRs. Applicable aliases are moved to the buffer. (The calling routine's write routine is used to write the buffer.) This action is repeated until all true names have been processed. Alias names that do not match any true name are then grouped by TTR and written. A message is provided to indicate that no true name exists for them.

Control then returns to IKJEHMMR, where the return code is checked and control is passed to IKJEHAL1.

Possible message: IKJ58301I

Object Module: IKJEHMEM
CSECT: IKJEHMMR

Chapter 7. LISTBC Command Processing

This section describes the **LISTBC** command and provides diagnostic aids for isolating and fixing a problem in the LISTBC command processor.

Overview

SEND provides the facility to send short messages from a TSO user, background job, or operator to a TSO user. The message is displayed on the terminal if that user is logged on. The sender may request that a message sent to a user who is not logged on be stored for the receiver. Such messages can be retrieved using the LISTBC command. The sender may request that a message be stored even if the receiver is logged on.

The LISTBC (and SEND) function allows messages to be stored in a separate log for each user. PARMLIB settings allow the installation to supply a data set name for this log. Exits are provided to allow the installation to modify the data set name based on the parameters passed to the SEND command. This data set name may refer to member of an existing PDS of each TSO user. If the data set name, as defined in the PARMLIB and modified by the pre-save exit, is not found in the catalog, then it will be created for the user during LISTBC processing.

Instead of individual logs, the installation may choose to continue the use of SYS1.BRODCAST as a message repository. This may be done by setting the LOGNAME field in PARMLIB to 'SYS1.BRODCAST'.

LISTBC displays the notices found in SYS1.BRODCAST. LISTBC displays messages that have been stored for the user in the user's private log, if one has been defined. At the option of the installation, it will also list any messages for the user found in SYS1.BRODCAST. For LISTBC, as for SEND, the PARMLIB and exits are used to determine the data set name of the user's private log. An exit allows the installation to process the message text before display. After reading and processing the messages, LISTBC will remove the messages from the user's log and/or SYS1.BRODCAST.

Log Storage Implications

User logs impact direct storage access device utilization. If each user has a new file dedicated to the SEND log, a minimum of one track will be used for each user. The installation may choose to set LOGNAME to a member of a file most users already have or exits can be used to implement other access methods that require less than one track per user.

Diagnosing a LISTBC Problem

This section describes diagnostic information provided to solve a problem with the LISTBC command processor. Before going further you may wish to create a search argument and use it to search a problem data base to see if there is already a fix for the problem. For information about searching a problem data base, see "Creating a Search Argument" in *TSO Extensions System Diagnosis: Guide and Index.*

LISTBC Abend Codes

LISTBC does not issue any abend codes. The LISTBC command issues an SVC dump when it detects an error. It does not request a dump for the following system abend codes:

- '913'
- '13E'
- 'x22'
- 'x37'.

LISTBC Return Codes

LISTBC provides the following decimal return codes if user logs are being used:

- 0 -- Messages and notices displayed.
- 4 -- Messages only displayed.
- 8 -- Notices only displayed.
- 12 -- No notices or messages to display.
- 16 -- Messages and notices not displayed. Installation exit denied access.
- 20 -- Messages and notices not displayed. Command not invoked authorized.
- 92 -- Messages and notices not displayed. System error.

LISTBC provides the following decimal return codes if user logs are *not* being used:

- 0 -- LISTBC processing was successful.
- 12 -- LISTBC processing was not successful.

Note: Because LISTBC propagates reason codes issued during processing of standard format exits, you may receive return codes other than those listed. For a description of standard format exit reason and return codes, see *TSO Extensions Customization.*

Services Used by LISTBC

The LISTBC command uses the following services during processing:

- Parse, DAIR, ESTAE, MODESET, DYNALLOC (SVC 99)
- PUTLINE, LINK, TESTAUTH, STACK, TCLEARQ, OPEN
- READ, WRITE, CLOSE, BLDL, SETRP, VRADATA
- ENQ, DEQ, SDUMP, TSO Service Facility.

LISTBC Dump Information

When an abend occurs, LISTBC places the following information in the system diagnostic work area (SDWA):

- Load module name (LISTBC)
- Active module name
- Level of the active module
- Recovery module name (IKJEES71)
- Recovery label name (IKJEES71)
- Component ID (28502)
- Component ID base (5665)
- A functional description of the LISTBC command.

LISTBC places the following information in the SDWA variable recording area (SDWAVRA):

- Name of the abending module
- Name of the recovery work area, IKJEESLR, and its address.

For information about finding the SDWA and SDWAVRA in a dump, see *TSO Extensions System Diagnosis: Guide and Index.*

TSO Extensions System Diagnosis: Guide and Index also provides information about reading messages, issuing traces and traps and calling IBM to report a problem you are unable to fix.

Exit Considerations

LISTBC invokes standard format exits that can change processing of the command. For a description of these exits, see *TSO Extensions Customization.*

TSO provides a common recovery routine for the standard format exits. When an abend occurs while attempting to invoke an exit, or during exit processing, the recovery routine furnishes the following information in the SDWA:

- Load module name (IKJRTR01)
- Active module name (IKJRTR01 or the active exit name)
- Level of the active module
- Recovery module name (RTRESTAE)
- Recovery label name (RTRESTAE)
- Component ID (28502)
- Component ID base (5665)

The common recovery routine furnishes the following information in the SDWAVRA when issuing a dump:

- Name of the exit handler parmlist, followed by its address.
- Name of the abending exit.

For unauthorized exits, a SNAP dump is issued; an SVC dump is issued for authorized exits.

Chapter 8. LISTDS Command Processing

This section describes the logic of the **LISTDS** command. It emphasizes the flow of data and control information through buffers and tables, and contains detailed functional descriptions through the use of method of operations diagrams. Figure 8-1 is the visual table of contents for the LISTDS command.

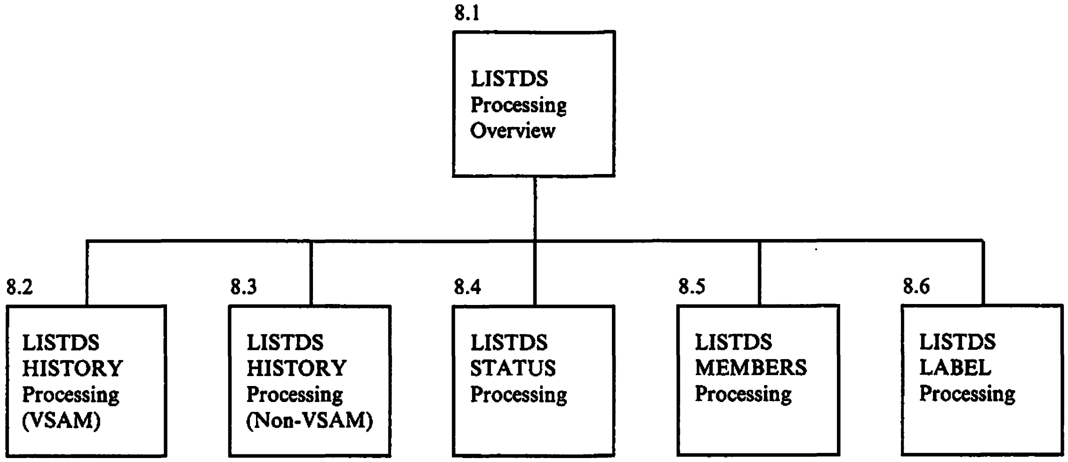
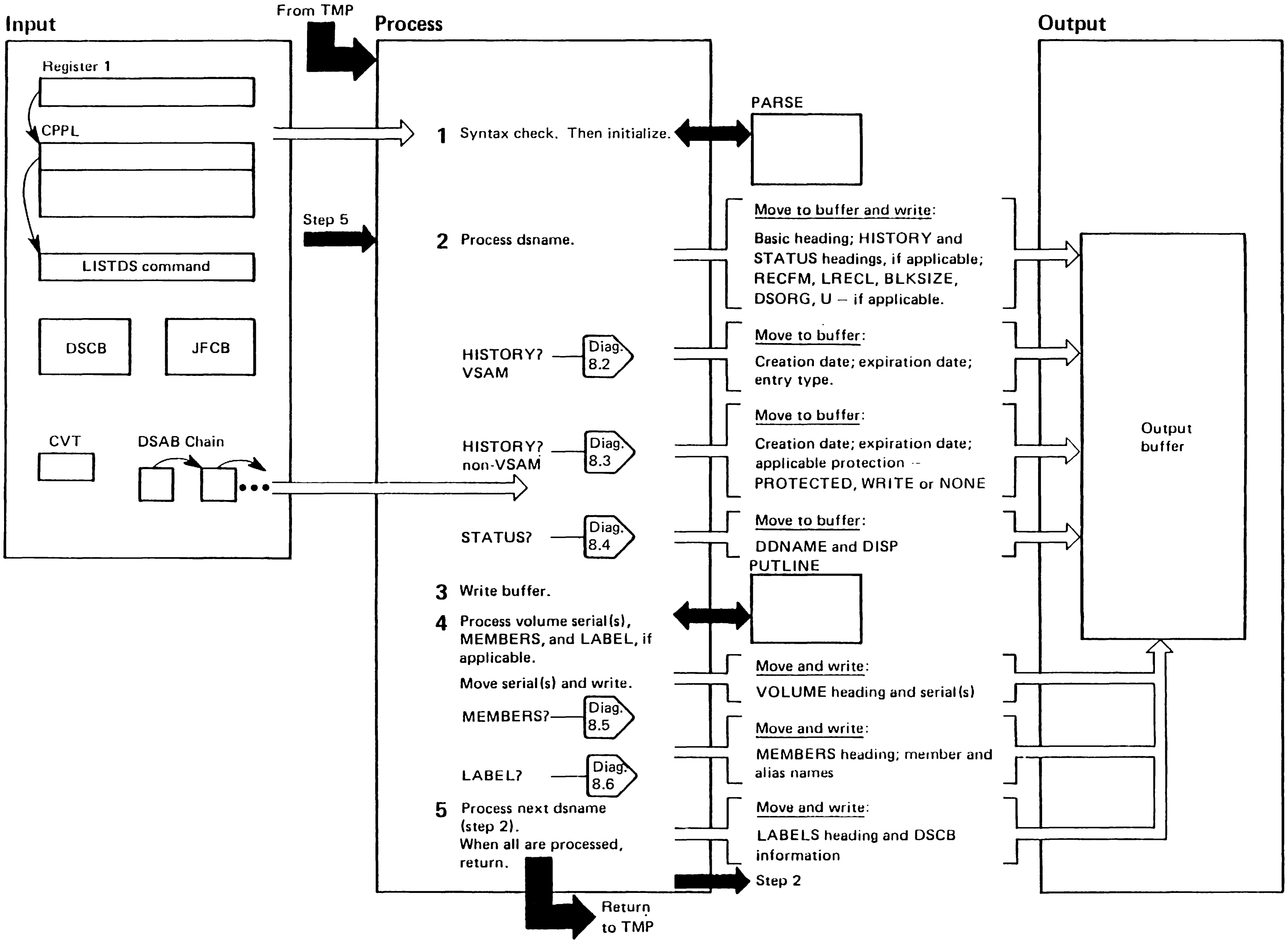

8.1
LISTDS Processing Overview
8.2
LISTDS HISTORY Processing (VSAM)
8.3
LISTDS HISTORY Processing (Non-VSAM)
8.4
LISTDS STATUS Processing
8.5
LISTDS MEMBERS Processing
8.6
LISTDS LABEL Processing

Figure 8-1. LISTDS Command Processing Visual Table of Contents

Diagram 8.1. LISTDS Processing Overview (Part 1 of 2)



1 Syntax check. Then initialize.

2 Process dsname.

Move to buffer and write:
Basic heading; HISTORY and STATUS headings, if applicable; RECFM, LRECL, BLKSIZE, DSORG, U – if applicable.

Move to buffer:
Creation date; expiration date; entry type.

Move to buffer:
Creation date; expiration date; applicable protection -- PROTECTED, WRITE or NONE

Move to buffer:
DDNAME and DISP

3 Write buffer.

4 Process volume serial(s), MEMBERS, and LABEL, if applicable.

Move serial(s) and write.

Move and write:
VOLUME heading and serial(s)

Move and write:
MEMBERS heading; member and alias names

Move and write:
LABELS heading and DSCB information

5 Process next dsname (step 2).
When all are processed, return.

Diagram 8.1. LISTDS Processing Overview (Part 2 of 2)

Extended Description **Module** **Label**

1 The parse routine receives control to check the command buffer for incorrect or unspecified parameters. Upon return from parse, the return code is checked, and then appropriate option bits are set to reflect the specified options. If MEMBERS is requested, load module IKJEHMEM is loaded into storage. If STATUS is specified, the DSAB is located. The LEVEL keyword or an '*' indicates that the data set name is generic.

Possible messages: IKJ58511I, IKJ58512I

2 The first entry in the dsname list is pointed to. The NXDSNAME subroutine examines the dsname and fully qualifies it, if necessary.

Possible messages: IKJ58503I, IKJ58509I, IKJ58502I, IKJ58513I

Then the fully-qualified name is moved to the output buffer and written.

Non-VSAM Data Sets: The LOCATE macro is used in an attempt to locate the dsname through the catalog. If LOCATE passes back a non-zero return code, DAIR is used (with an X'08' operation code) in an attempt to see if the data set is otherwise accessible. If the data set cannot be located, the user is informed and processing continues with the next dsname.

Possible messages: IKJ58503I, IKJ58506I, IKJ58507I, IKJ58508I, IKJ58510I, IKJ58512I

Depending on which way the data set was found (using LOCATE or DAIR), set up is performed prior to issuing an OBTAIN. If LOCATE found the data set, the LOCATE switch is set. If DAIR found the data set, the DDNAME is moved to a DCB and the RDJFCB macro is issued to get the JFCB.

Then the OBTAIN macro is issued to bring the DSCB into storage. If the return code is not equal zero, processing continues with the next dsname in the list. Otherwise, heading information is moved to the buffer and written.

VSAM Data Sets: The catalog information routine is used to supply a list of names. VSAM LOCATE is used to indicate whether the data set is VSAM or not. LOCATE also supplies the required attributes if the data set is VSAM.

Possible message: IKJ58504

HISTORY is processed, if applicable. See Diagram 8.2 (VSAM) or 8.3 (non-VSAM)
STATUS is processed, if applicable. See Diagram 8.4.

3-4 First the buffer is written, then volume serials are placed, one by one, in the buffer and printed.

Possible message: IKJ58504

Then a check is made for MEMBERS and LABEL processing.
See Diagrams 8.5 and 8.6 respectively.

5 After MEMBERS and/or LABEL have been processed, additional dsnames, if any, are processed. When all have been processed, control returns to the TMP.

Object Module: IKJEHDS1
CSECT: IKJEHBSC

Diagram 8.2. LISTDS HISTORY Processing (VSAM) (Part 1 of 2)

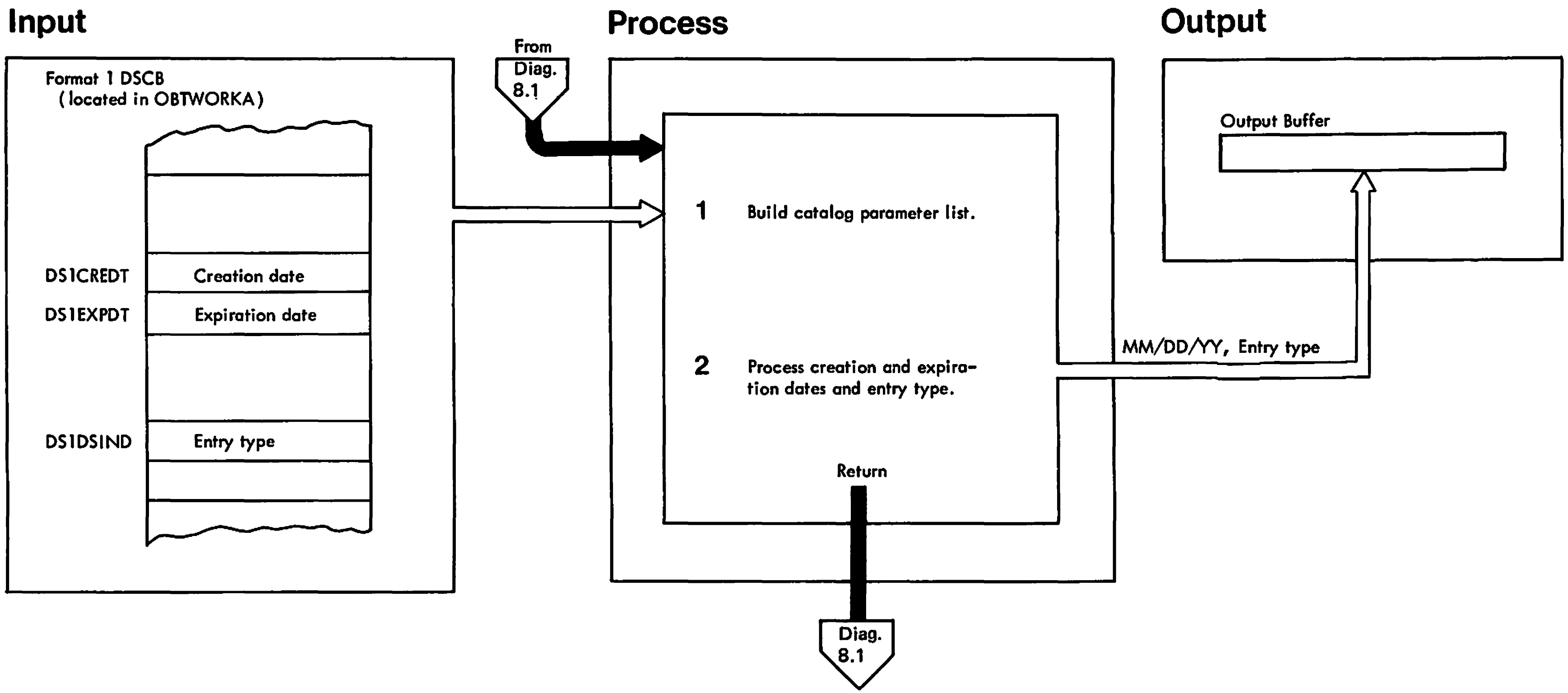


Diagram 8.2. LISTDS HISTORY Processing (VSAM) (Part 2 of 2)

	Extended Description	Module	Label
1	Build a catalog parameter list using information and the data set name obtained from the DSAB by SVC99. The parameter list specifies the data set name to be retrieved from the VSAM catalog, the entry type, creation and expiration dates, logical record length, volume serials, and physical blocksize for the data set.		
2	Upon entry from the IKJEHDS1 routine, IKJEHHIS gets a save area, then sets up to convert the creation date from YMMDD format to MM/DD/YY. The creation date is converted and placed in the buffer. If no date was found, a default of 00/00/00 is placed in the buffer. This process is repeated for the expiration date. The entry type code is also moved into the output buffer.		

Object Module: IKJEHDS1
CSECT: IKJEHHIS

Diagram 8.3. LISTDS HISTORY Processing (Non-VSAM) (Part 1 of 2)

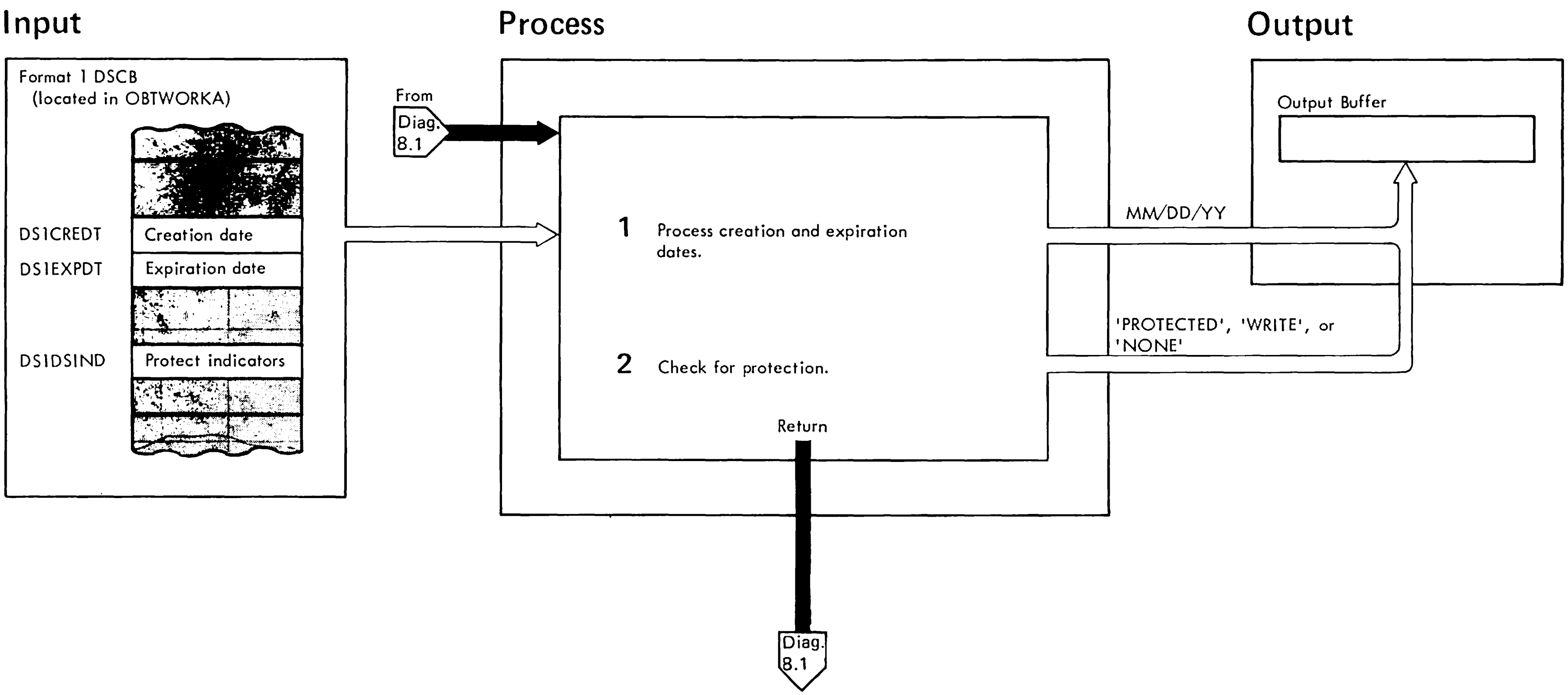


Diagram 8.3. LISTDS HISTORY Processing (Non-VSAM) (Part 2 of 2)

Extended Description **Module** **Label**

1 Upon entry from the IKJEHDS1 routine, IKJEHHIS gets a save area, then sets up to convert the creation date from YMMDD format MM/DD/YY.

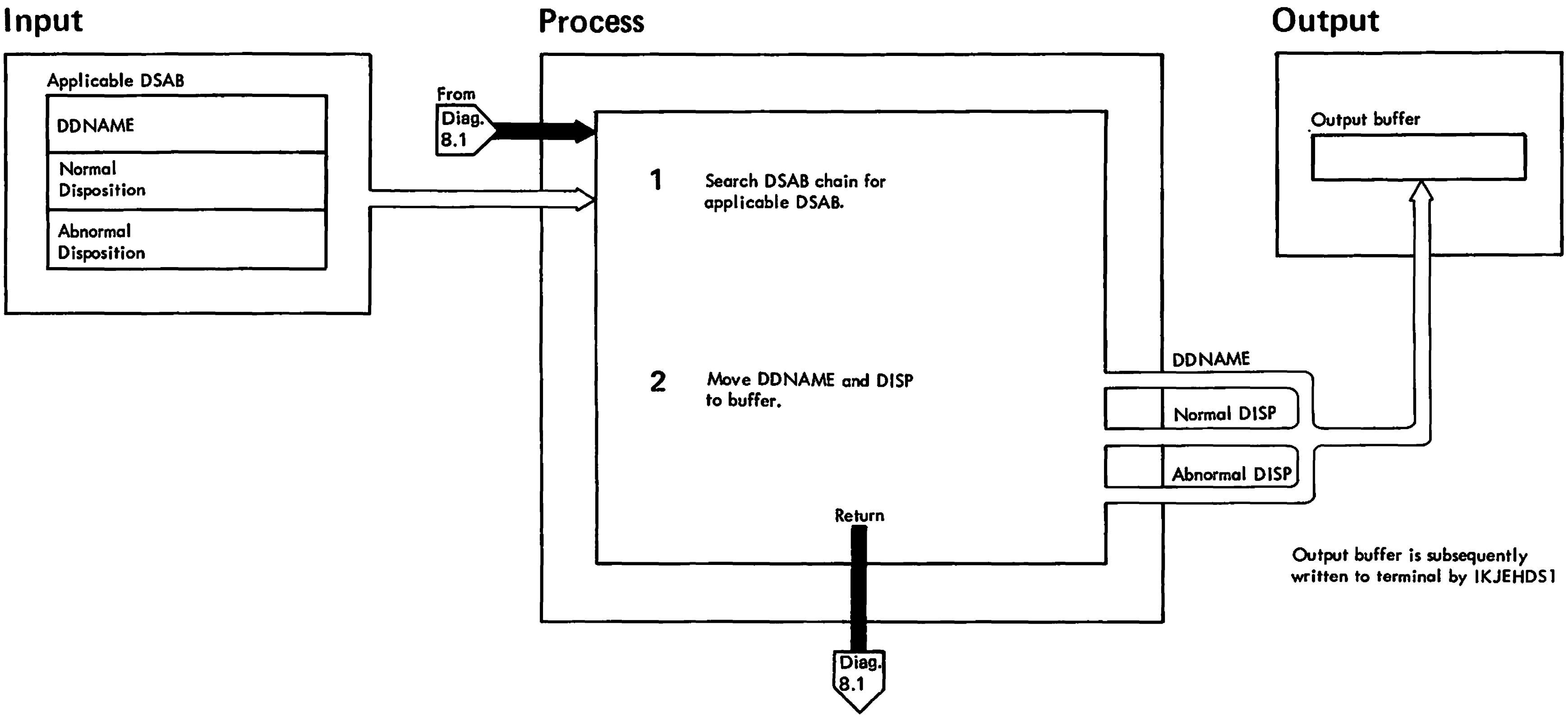
The creation date is converted and placed in the buffer. If no date was found, a default of 00/00/00 is placed in the buffer. This process is repeated for the expiration date.

2 Then a check is made for password protection. If password protection applies, 'PROTECTED' is placed in the buffer. Otherwise, a check is made for WRITE protection. If WRITE protection applies, 'WRITE' is placed in the buffer. Otherwise 'NONE' is placed in the buffer.

Then the space obtained for the save area is freed and control returns to the IKJEHDS1 routine. See Diagram 8.1.

Object Module: IKJEHDS1
CSECT: IKJEHHIS

Diagram 8.4. LISTDS STATUS Processing (Part 1 of 2)



1 Search DSAB chain for applicable DSAB.

2 Move DDNAME and DISP to buffer.

Output buffer is subsequently written to terminal by IKJEHDS1

Diagram 8.4. LISTDS STATUS Processing (Part 2 of 2)

Extended Description **Module** **Label**

1 SVC99 searches the DSAB chain for a data set name that matches the name in the data set list. The search is done to obtain allocation information about the data set name. If no match is found, control is returned to IKJEHDS1.

2 When a dsname match is found, a check is made to see if HISTORY is also specified (in which case, the buffer is filled partially with HISTORY information that has not yet been written). If yes, the offset to the output buffer area is adjusted accordingly.

Then the DDNAME is moved to the output buffer.

The status bits are then tested for normal disposition, and the appropriate word (KEEP, DELETE, CATLG, or UNCATLG) is placed in the buffer.

After the output buffer offset is adjusted, the status bits are tested for disposition in the event of an abnormal termination. The appropriate disposition is placed in the buffer.

Control returns to the IKJEHDS1 routine. See Diagram 8.1.

Object Module: IKJEHDS1
CSECT: IKJEHSTA

Diagram 8.5. LISTDS MEMBERS Processing (Part 1 of 2)

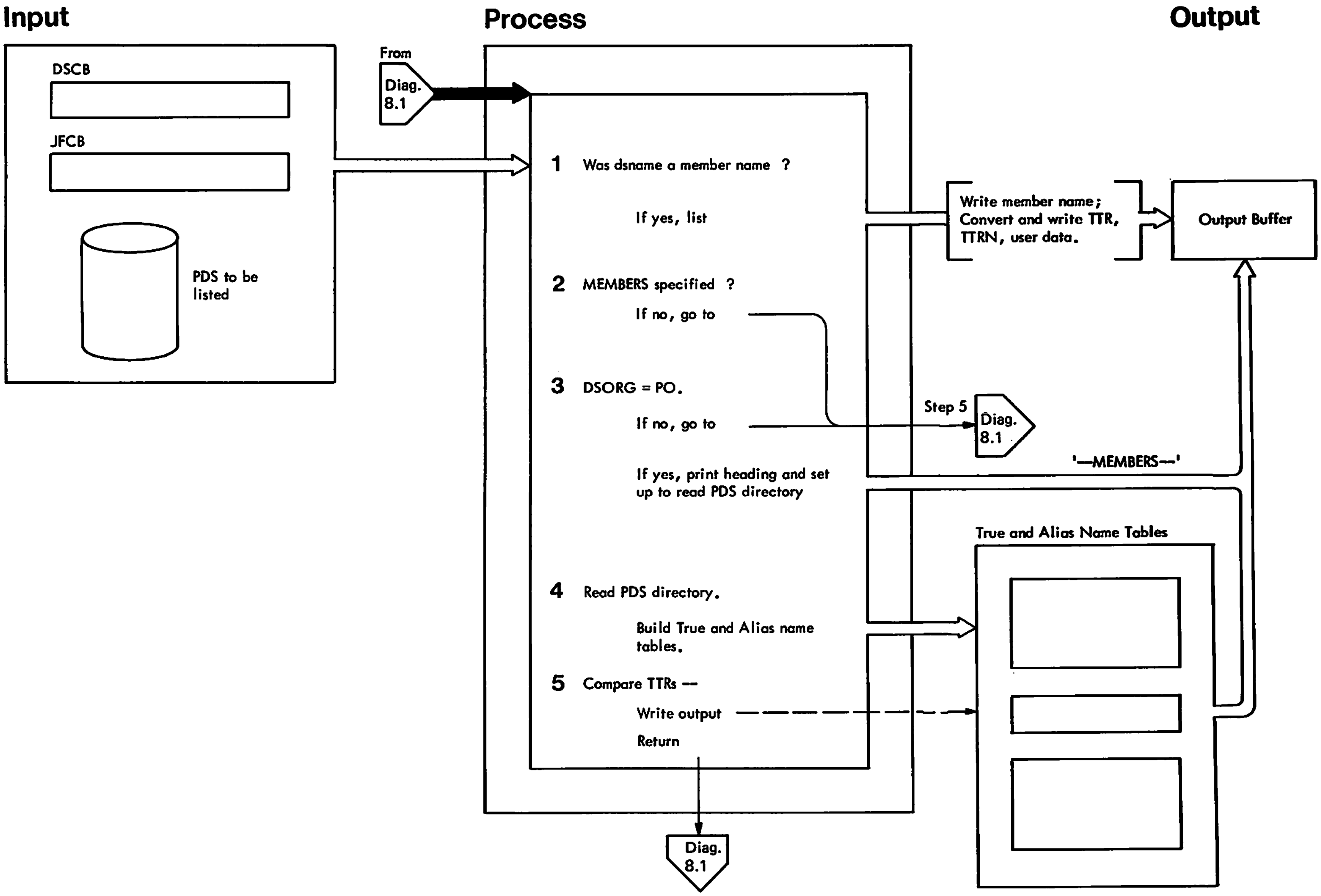


Diagram 8.5. LISTDS MEMBERS Processing (Part 2 of 2)

Extended Description **Module** **Label**

1 After volume information has been printed, a check is made to see whether the current dsname is a member name. If not a member name, a check for label processing is first made, then a check is made to see if the MEMBERS keyword is specified (step 2).

If the dsname was a member name, routine MNAMROUT is given control to print specific information for the member. If necessary, MNAMROUT issues a RDJFCB and passes control to DAIR to allocate the data set.

2 Then LABEL processing takes place, if applicable (see Diagram 8.6). After this, a check is made to see if MEMBERS was specified. If no, processing continues from step 5 of Diagram 8.1. Otherwise control is passed to the MEMBERS interface routine, MEMROUT.

3 A check is made to ensure that the organization is partitioned. If not, control is returned to step 5 of Diagram 8.1. Otherwise the MEMBERS heading is written. Then a check is made to see if the JFCB has already been read. If yes, processing continues from step 4, below. Otherwise, DAIR is used to allocate the data set. Then the DDNAME is placed in DCBDDNAM of OBTDCB and an RDJFCB is issued prior to reading the PDS directory. Then control passes to IKJEHMEM.

Possible messages: IKJ58502I, IKJ58514I

4 IKJEHMEM initializes tables to contain the true and alias names, then reads the PDS directory into the tables. Name blocks are obtained and chained dynamically, as required.

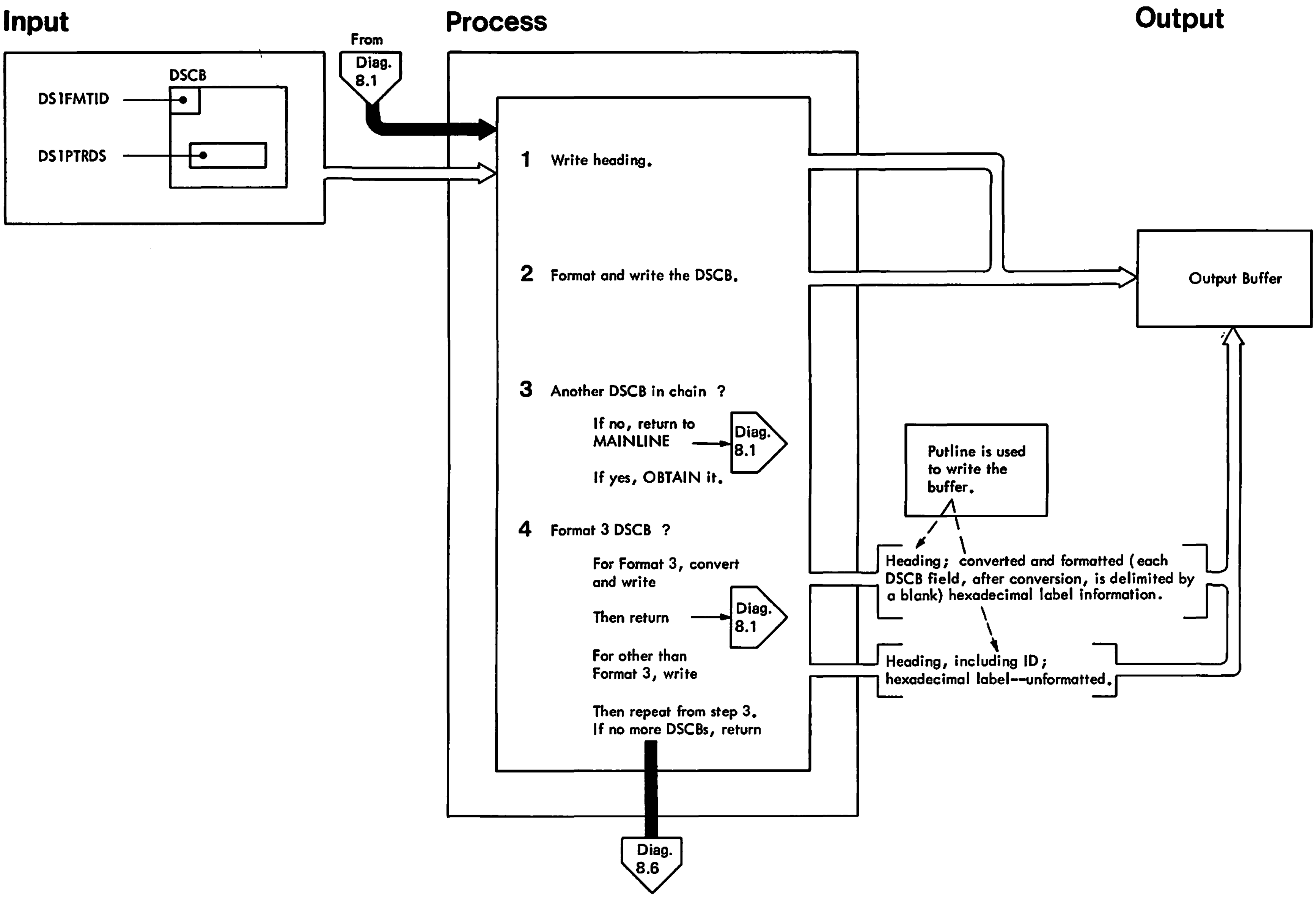
5 Then a true name is moved to the output buffer. The true name TTR is compared to all of the alias name TTRs. Applicable aliases are moved to the buffer (MEMROUT's write routine, which uses PUTLINE, is used to write the buffer.) This action is repeated until all true names have been processed. Alias names not matching any true name are then grouped by TTR and written. A message is provided to indicate that no true name exists for them. Control returns to MEMROUT.

Possible message: IKJ58501I

MEMROUT checks the return code, then returns control to step 5 of Diagram 8.1 to process the next name in the list.

Object Modules: IKJEHDS1, IKJEHMEM

Diagram 8.6. LISTDS LABEL Processing (Part 1 of 2)


Input
Process
Output
From
Diag. 8.1
DSCB
DS1FMTID
DS1PTRDS
Diag. 8.1
Diag. 8.1
Diag. 8.6
Output Buffer
Putline is used to write the buffer.
Heading; converted and formatted (each DSCB field, after conversion, is delimited by a blank) hexadecimal label information.
Heading, including ID; hexadecimal label--unformatted.

1 Write heading.

2 Format and write the DSCB.

3 Another DSCB in chain ?

If no, return to MAINLINE

If yes, OBTAIN it.

4 Format 3 DSCB ?

For Format 3, convert and write

Then return

For other than Format 3, write

Then repeat from step 3. If no more DSCBs, return

Diagram 8.6. LISTDS LABEL Processing (Part 2 of 2)

	Extended Description	Module	Label
1	After getting a save area, IKJEHLBL uses PUTLINE to write the heading for the DSCB.		
2	IKJEHLBL refers to DS1FMTID for the address of the DSCB information to be converted. The DSCB information is then converted from binary to hexadecimal and written one line at a time. Formatting consists of separating each field by a blank.		
3	DS1PTRDS is then checked to determine if any DSCBs are chained to the format 1 DSCB just processed. If none, control returns to MEMBCHK in the IKJEHDS1 routine. If another DSCB is found, an OBTAIN is issued for it. *Possible message: IKJ58505I*		
4	A check for a Format 3 DSCB is made. (A Format 3 DSCB is formatted as in step 2 and 3 above.) If a Format 3 DSCB is found, control returns to the IKJEHDS1 routine after the DSCB is processed. If the DSCB is other than a Format 3 DSCB, no formatting takes place. That is, the information is converted to hexadecimal and dumped 36 bytes at a time. Then a check is made for another DSCB. If none, storage is freed and control returns to MEMBCHK in the IKJEHDS1 routine.		

Object Module: IKJEHDS1
CSECT: IKJEHLBL

Chapter 9. LOGON Command Processing

This section describes the logic of the **LOGON** command. It emphasizes the flow of data and control information through buffers and tables, and contains detailed functional descriptions through the use of method of operations diagrams. Figure 9-1 is the visual table of contents for the LOGON command.

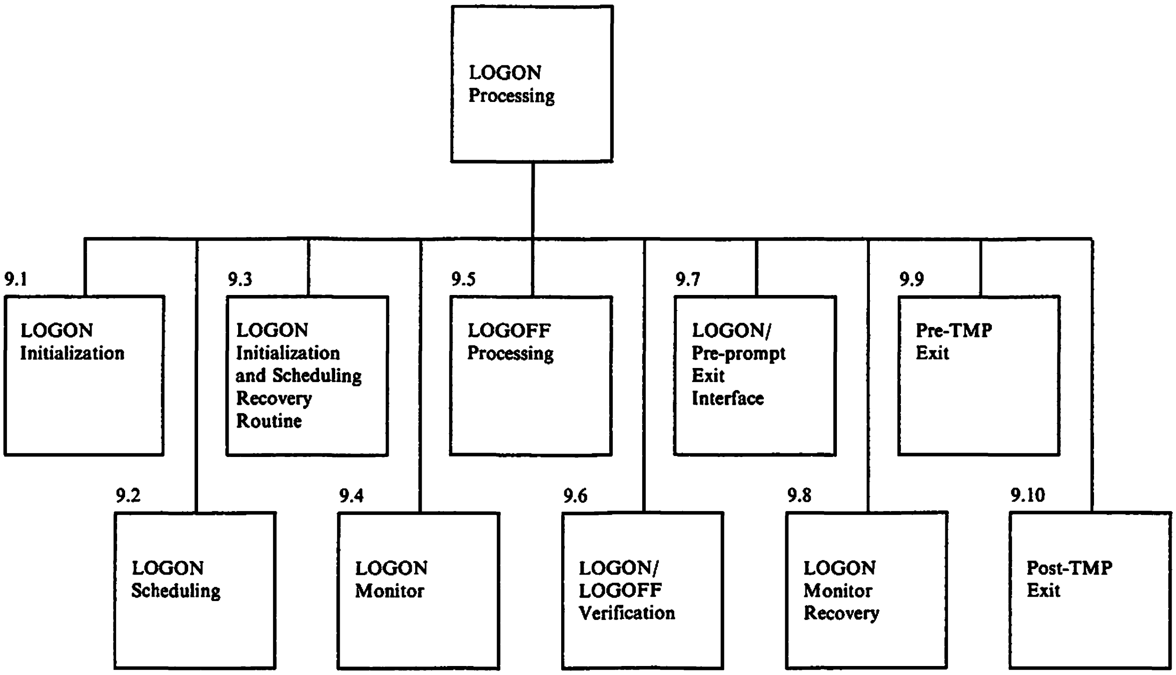


Figure 9-1. LOGON Command Processing Visual Table of Contents

LOGON Scheduling

Started task control (STC) transfers control to the LOGON pre-initialization routine, IKJEFLA, which resides below 16 megabyes in virtual storage.

IKJEFLA places the address of the logon address table, IKJEFTBL, into the TSO vector table and passes control to LOGON initialization, IKJEFLA1, which resides above 16 megabytes in virtual storage.

IKJEFLA1 does the following:

- Initializes the various control blocks required for LOGON and the terminal session.
- Establishes the ESTAE recovery routine, IKJEFLS.
- Searches the master scheduler JCL, MSTRJCL, for the system broadcast data set, SYSLBC and the system user attribute data set, SYSUADS. (If the level of RACF is 1.8 or higher and all user information has been converted to the RACF data base, SYSUADS need not be present on the system.)
- Calls the LOGON scheduler, IKJEFLB.
- For an initial LOGON, IKJEFLB receives control from IKJEFLA1. For a re-LOGON or LOGOFF, IKJEFLB receives control from the job scheduling subroutine, JSS.

IKJEFLB attaches the LOGON prompting monitor, IKJEFLC, and then waits for notification of the appropriate processing depending on whether the request is a LOGON or LOGOFF.

For a LOGON request:

- IKJEFLC calls LOGON verification, IKJEFLE, which parses the command to obtain the LOGON data and verify this data against either the user attribute data set or the RACF data base, depending on how the user was defined to the system.
- IKJEFLC notifies IKJEFLB that it should pass control to JSS to schedule the terminal session.
- IKJEFLC calls IKJEFLH to wait for JSS to finish.
- JSS passes control to the pre-TMP exit, IKJEFLJ.
- IKJEFLJ notifies IKJEFLH that IKJEFLH and then IKJEFLC should terminate.
- After IKJEFLJ terminates, the TMP is invoked for the user's terminal session.

For a re-LOGON or LOGOFF:

- The TMP terminates.
- JSS passes control to the post-TMP exit, IKJEFLK.
- After JSS completes, it transfers control to IKJEFLB, which attaches IKJEFLC.
- IKJEFLC calls the LOGOFF processor, IKJEFLL. IKJEFLC then calls IKJEFLE, which parses the command.

- If the command is a re-LOGON:
  - IKJEFLE obtains the LOGON data and verifies this data against either the user attribute dataset or calls IKJEFLE2 to verify the data against the RACF data base.
  - IKJEFLC notifies IKJEFLB that it should transfer control to JSS to schedule the terminal session. JSS passes control to the pre-TMP exit, IKJEFLJ.
  - IKJEFLC passes control to IKJEFLH to wait for JSS to complete.
  - IKJEFLJ notifies IKJEFLH that IKJEFLH and then IKJEFLC should terminate.
  - After IKJEFLJ terminates, the TMP is invoked for the user terminal session.
- If the command is a LOGOFF:
  - IKJEFLE notifies IKJEFLC that the current terminal session should terminate.
  - IKJEFLC notifies IKJEFLB to terminate and transfer control to started task control, STC.

See Figure 9-2 for a diagram of LOGON scheduling flow and Figure 9-3 for an overview of LOGON scheduling control blocks.

LOGON Module Addressing and Residency Changes

The following is a breakdown of the LOGON control modules into two categories:

1. The modules that reside in 24-bit addressable storage.
2. The modules that reside in 31-bit addressable storage.

AMODE(24) RMODE(24)

These modules reside in 24-bit addressable storage and can only access 24-bit addresses.

Module	Description
IKTXINIT	VTIOC Initialization Routine
IKTLOGR	LOGON Reconnect Routine
IKTIIOM	I/O Manager Initialization Routine
IKTLOGFF	Extended Logoff Routine
IKTRPLXT	OPNDST RPL Asynchronous Exit Routine
IKTXLOG	Extended LOGON Routine
IKJEFLA	LOGON Pre-Initialization Routine
IKJEFLIO	LOGON UADS I/O Routine
IKJEFTBL	LOGON Address Table

AMODE(31) RMODE(ANY)

Each of these modules has a 31-bit addressing mode and can reside anywhere in virtual storage. However, the dynamic area for each of these modules resides below 16 megabytes in virtual storage.

Module	Description
IKJEFLA1	LOGON Initialization
IKJEFLB	LOGON Scheduler
IKJEFLC	LOGON Monitor
IKJEFLCM	LOGON Message CSECT for IKJEFLC
IKJEFLE	LOGON/LOGOFF Verification
IKJEFLE2	LOGON/LOGOFF Verification
IKJEFLE3	LOGON/LOGOFF Verification
IKJEFLG	Attention Processor
IKJEFLGB	ESTAI Recovery and Retry
IKJEFLGH	Message Text for IKJEFLG
IKJEFLGN	LOGON Message CSECT for IKJEFLGM
IKJEFLH	LOGON Information Routine
IKJEFLI	Installation Exit Interface
IKJEFLJ	Pre-TMP Exit
IKJEFLK	Post-TMP Exit
IKJEFLL	LOGOFF Processing
IKJEFLLM	LOGOFF Message for IKJEFLL
IKJEFLPA	Time and Date Processor
IKJEFLS	ESTAE Recovery and Retry
IKJEFLGM	Message Issuer Routine
IKJEFLEA	Parse/Scan Interface
IKJEFLJA	LOGON Full Screen Processor
IKJEFLJH	LOGON Full Screen HELP
IKJEFLJU	LOGON Defaults Processor
IKJEFRAF	TSO RACF Routine
IKJEFRRF	TSO RACF Routine Recovery

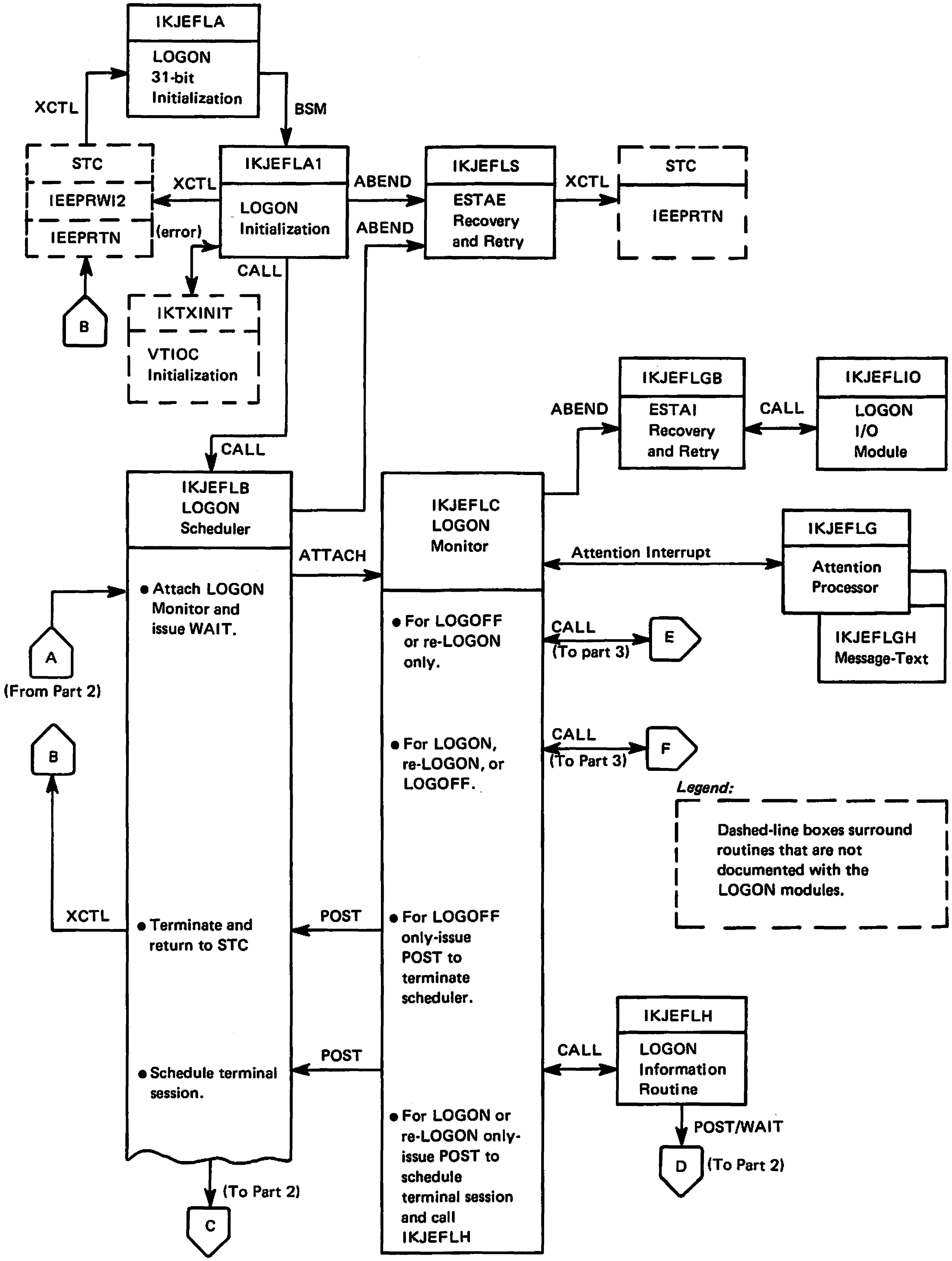


Figure 9-2 (Part 1 of 3). LOGON Schedule Module Flow

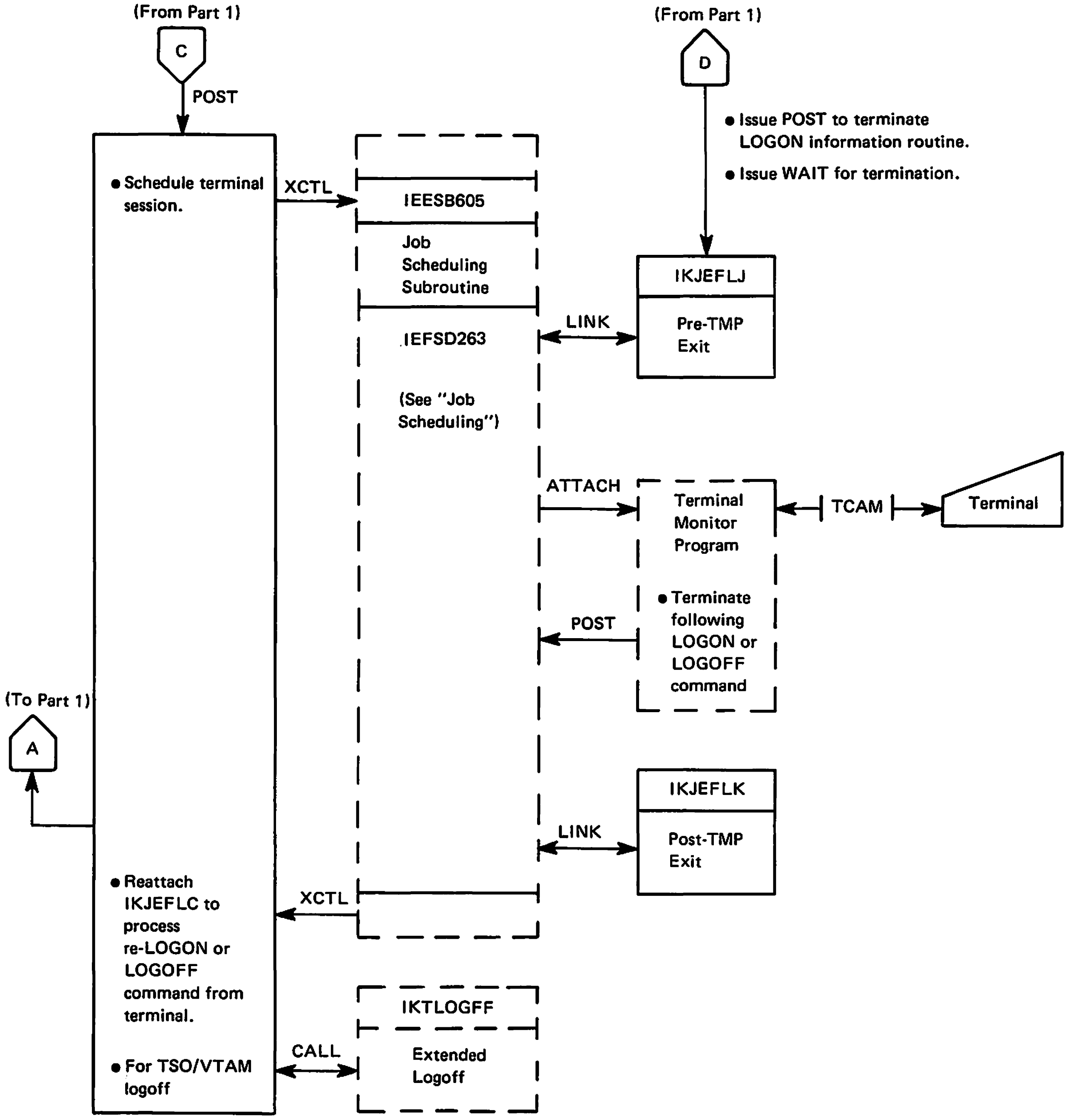

(From Part 1)
C
POST
• Schedule terminal session.
XCTL
IEESB605
Job Scheduling Subroutine
IEFSD263
(See "Job Scheduling")
(From Part 1)
D
• Issue POST to terminate LOGON information routine.
• Issue WAIT for termination.
IKJEFLJ
Pre-TMP Exit
LINK
ATTACH
Terminal Monitor Program
• Terminate following LOGON or LOGOFF command
POST
TCAM
Terminal
(To Part 1)
A
IKJEFLK
Post-TMP Exit
LINK
XCTL
• Reattach IKJEFLC to process re-LOGON or LOGOFF command from terminal.
• For TSO/VTAM logoff
CALL
IKTLOGFF
Extended Logoff

Figure 9-2 (Part 2 of 3). LOGON Schedule Module Flow

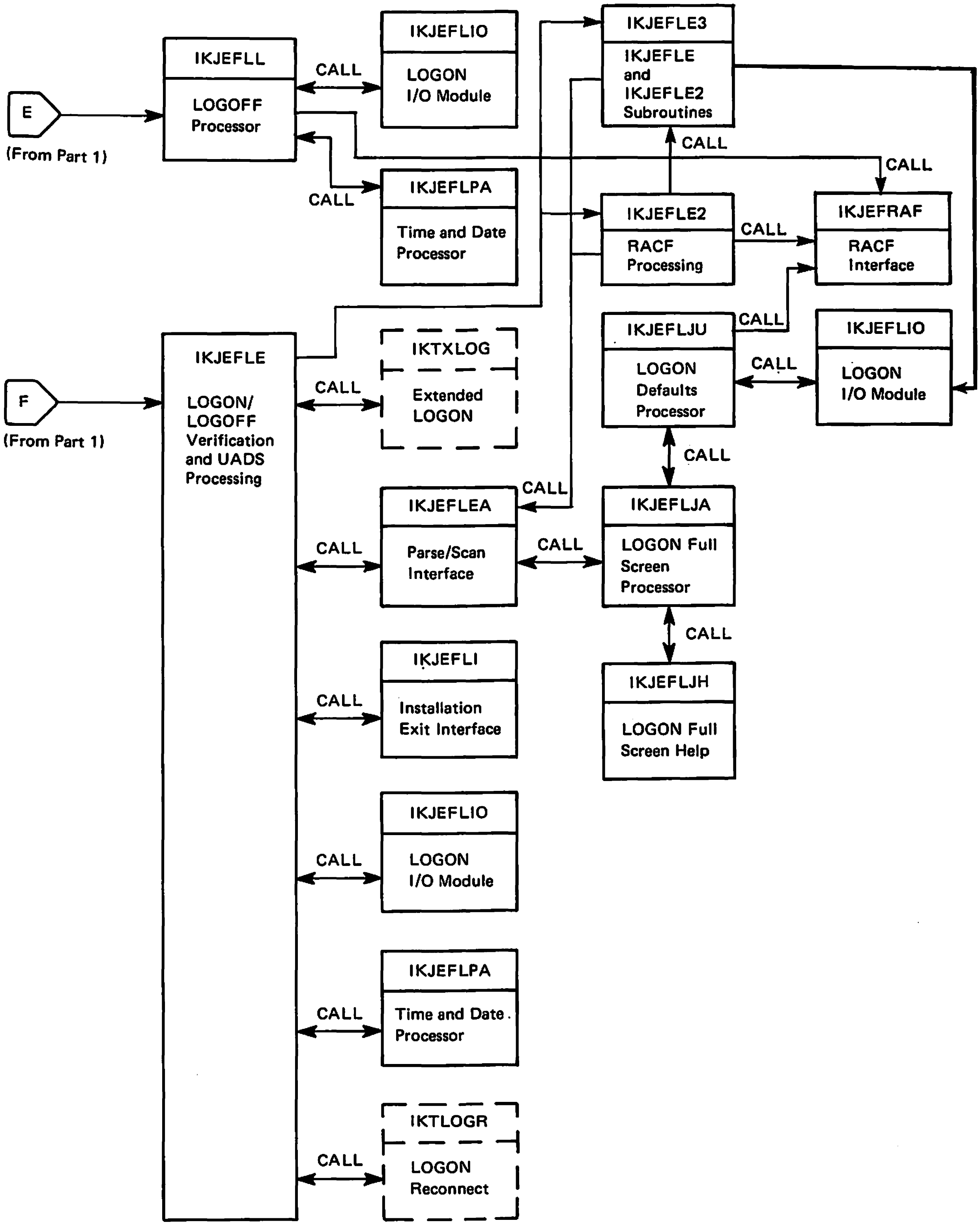


Figure 9-2 (Part 3 of 3). LOGON Schedule Module Flow

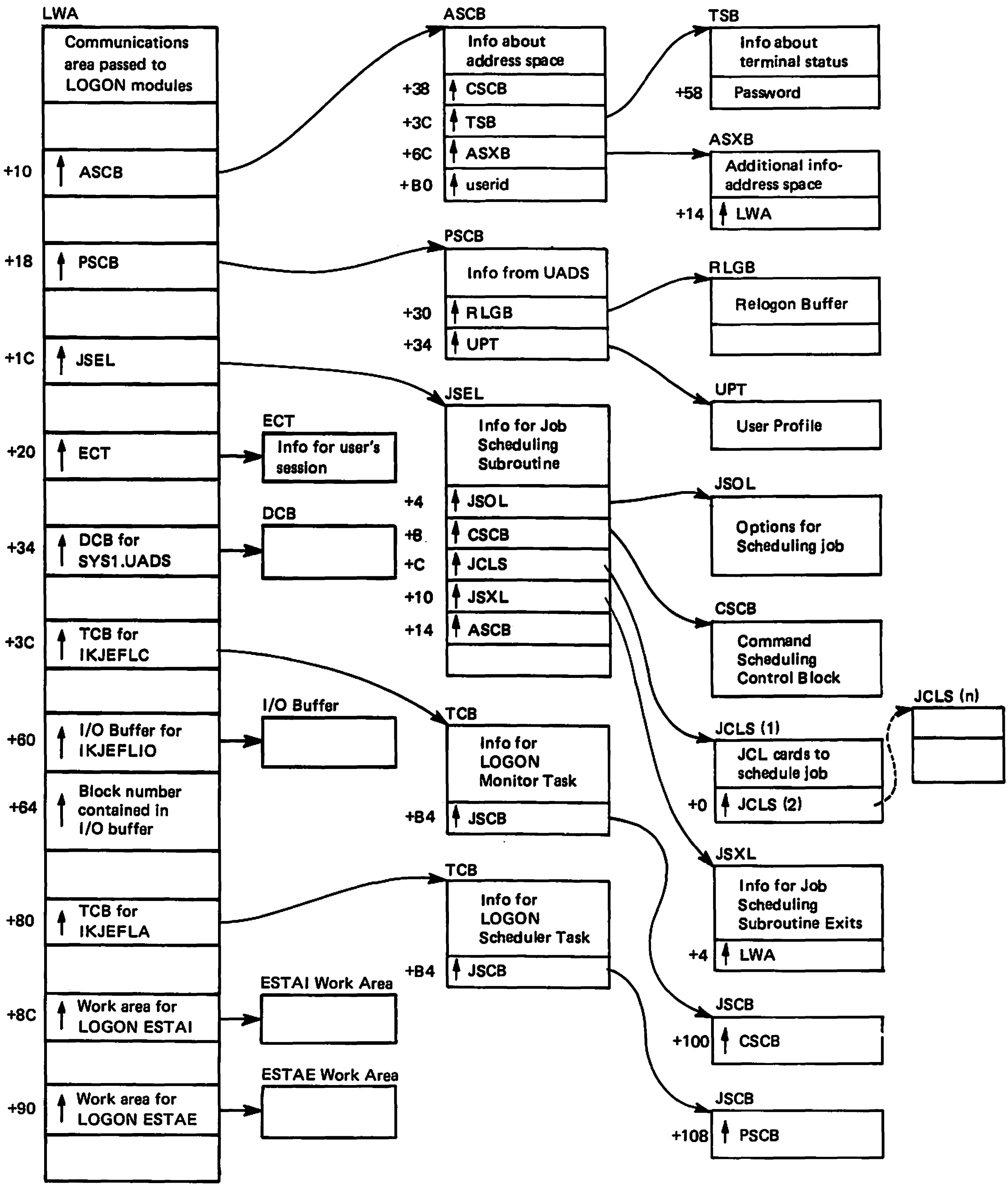
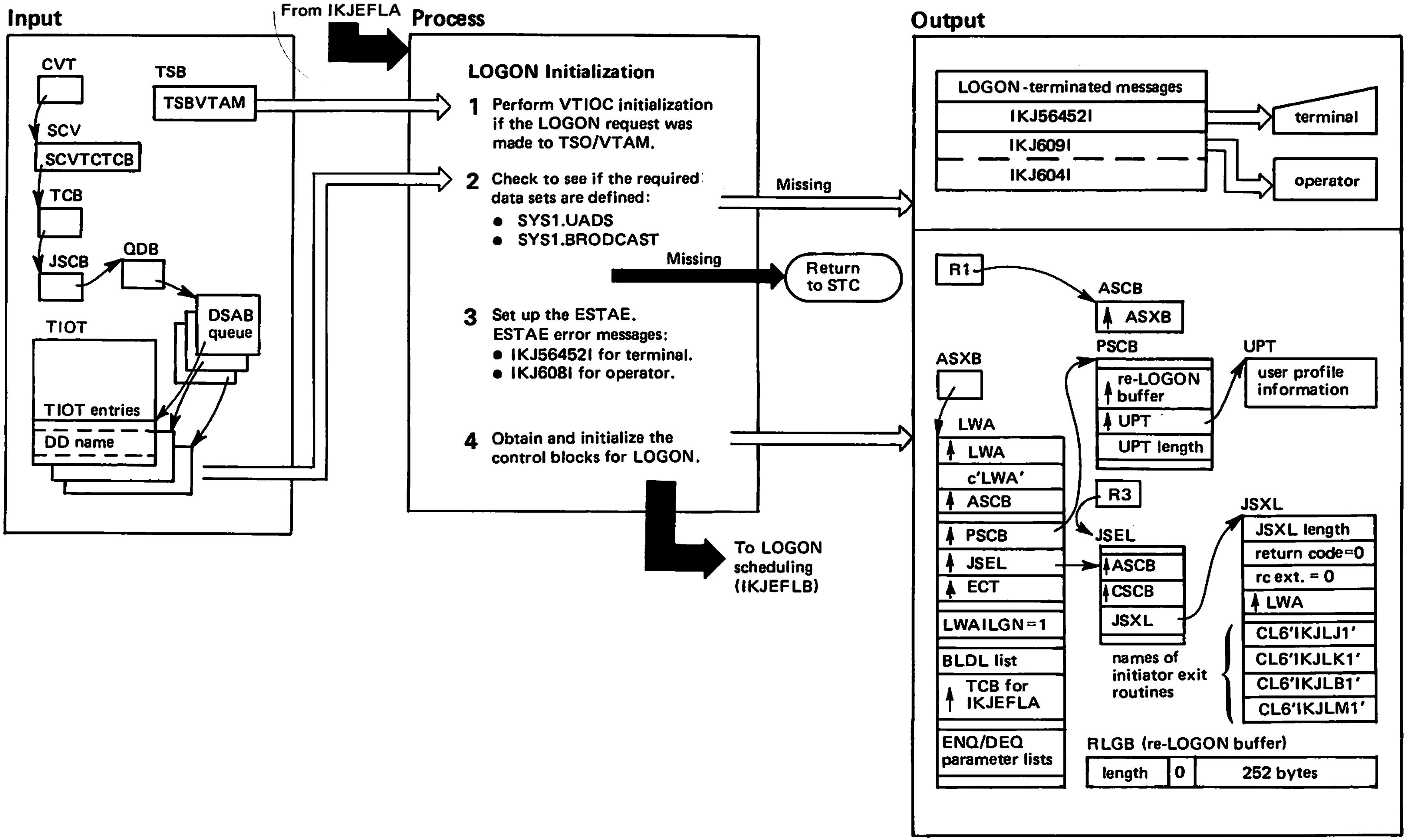


Figure 9-3. LOGON Scheduling Control Block Overview

Diagram 9.1. LOGON Initialization (IKJEFLA1) (Part 1 of 2)



LOGON Initialization

1 Perform VTIOC initialization if the LOGON request was made to TSO/VTAM.

2 Check to see if the required data sets are defined:
- SYS1.UADS
- SYS1.BRODCAST

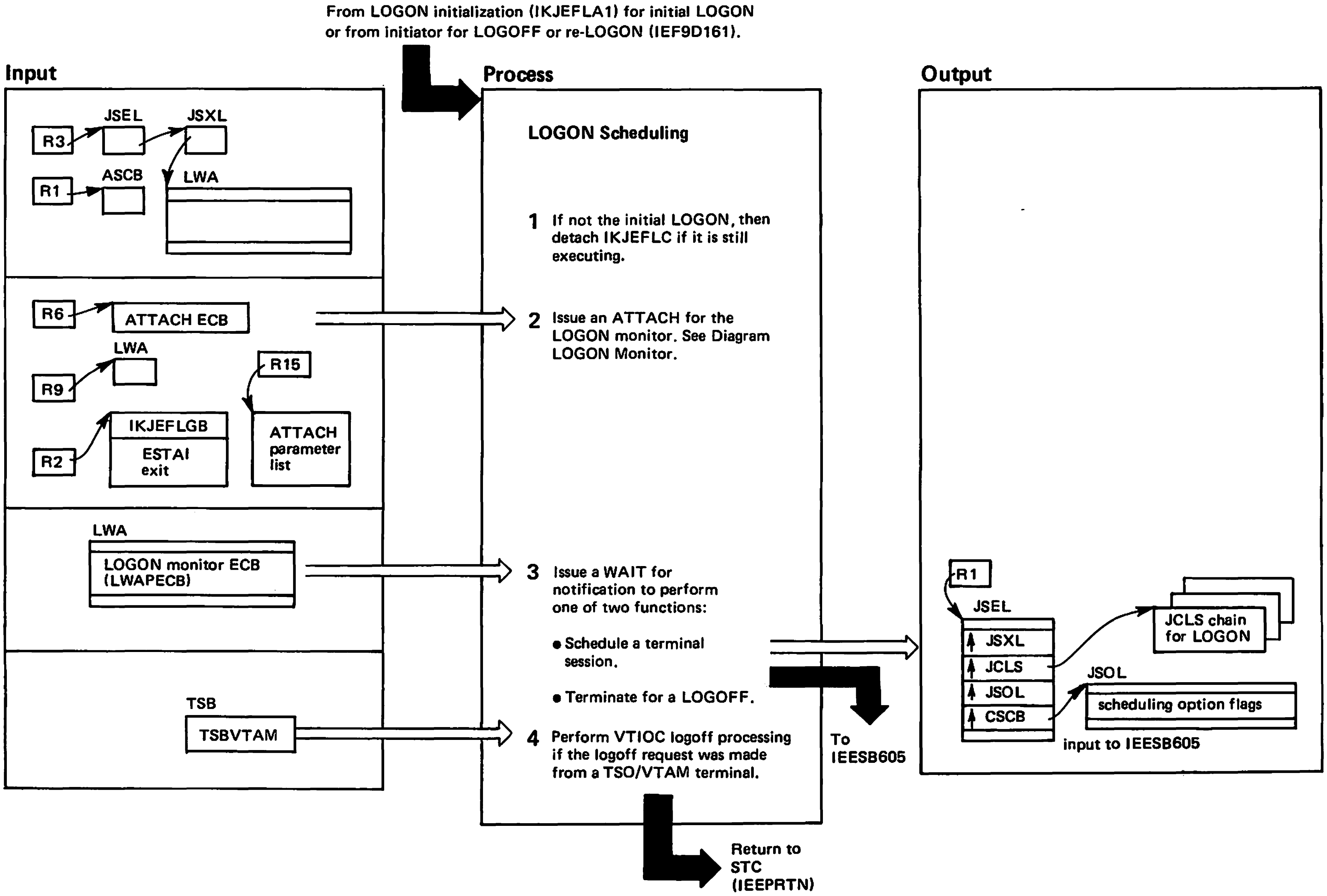
3 Set up the ESTAE.
ESTAE error messages:
- IKJ56452I for terminal.
- IKJ608I for operator.

4 Obtain and initialize the control blocks for LOGON.

Diagram 9.1. LOGON Initialization (IKJEFLA1) (Part 2 of 2)

Extended Description	Module	Label
LOGON initialization receives control from the LOGON pre-initialization routine, IKJEFLA, which received control from started task control (STC), to process an initial LOGON. LOGOFF or re-LOGON bypasses the initialization function.	IKJEFLA1	
1 IKTXINIT initializes VTAM control blocks and the TVWA, and transfers control (OPNDST PASS) of the terminal user's address space. An OPNDST RPL exit is then dispatched by VTAM to verify that the OPNDST was successful.	IKTXINIT	
2 The broadcast data set, SYS1.BRODCAST, must have been defined by master scheduler's JCL (MSTRJCL member of SYS1.LINKLIB). The SYS1.UADS must have been defined by master scheduler's JCL only if all the user IDs have not been converted to the RACF data base. LOGON initialization checks for these data sets by searching the master scheduler's TIOT for the DD names SYSUADS and SYSLBC. An error message is issued if SYSLBC is missing or if SYSUADS is missing and RACF version 1.8 or higher is not installed on the system.	IKJEFLA1	
3 IKJEFLS is used as the ESTAE routine to protect IKJEFLA1 and IKJEFLB.		
4 LOGON initialization creates the control blocks that contain LOGON information needed by the various LOGON routines. LOGON initialization turns on the initial-LOGON bit (LWAILGN) to indicate that this is the first LOGON command to be processed for the current address space.	IKJEFLA1	

Diagram 9.2. LOGON Scheduling (IKJEFLB) (Part 1 of 2)



LOGON Scheduling

1 If not the initial LOGON, then detach IKJEFLC if it is still executing.

2 Issue an ATTACH for the LOGON monitor. See Diagram LOGON Monitor.

3 Issue a WAIT for notification to perform one of two functions:

- Schedule a terminal session.
- Terminate for a LOGOFF.

4 Perform VTIOC logoff processing if the logoff request was made from a TSO/VTAM terminal.

Diagram 9.2. LOGON Scheduling (IKJEFLB) (Part 2 of 2)

Extended Description — **Module** — **Label**

LOGON scheduling receives control from LOGON initialization or from the initiator at the end of the terminal session (for LOGOFF or re-LOGON). The new terminal session that is scheduled following a re-LOGON operates in the same address space as the initial terminal session.

LOGON scheduling invokes the job scheduling subroutine. This subroutine interprets the JCL card images that define the terminal session and attaches the terminal monitor program (TMP), which processes commands from the terminal. The TMP remains active until it intercepts a LOGOFF or a re-LOGON command from the terminal. At that time, the TMP terminates and the initiator passes control back to LOGON scheduling to process the command.

1 Upon receiving control from STC for a LOGOFF or re-LOGON, LOGON scheduling ensures that the LOGON monitor has already terminated. If the monitor is yet active, LOGON scheduling notifies the monitor (ILWASECB-post code 20) to terminate. Once the monitor has terminated (LWAPECB-post code 24), LOGON scheduling detaches it and sets the attach ECB (LWAAECB) to zero. LOGON scheduling then performs the attach of the LOGON monitor (Step 2) as usual.

Module	Label
IKJEFLB	
IKJEFLB	WAITUST
	BEXIT
	LCRESTRT

If the LOGON monitor posts LWAPECB with an invalid post code (other than 16 and 24), LOGON scheduling terminates as follows:

- Detaches the LOGON monitor.
- Cancels the ESTAE environment.
- Places the address of the ASCB in register 1.
- Returns to STC (IEEPRTN) for CSCB clean-up.

But, if the LOGON monitor has caused an ABEND and recovery is to be attempted (LWABEND = 1), LOGON scheduling does not terminate; it reissues the ATTACH of the LOGON monitor (returns to Step 2).

2 LOGON scheduling handles the initial LOGON, a LOGOFF, or a re-LOGON. First, it issues an ATTACH macro instruction to invoke the LOGON monitor (see Diagram "LOGON Monitor"). The monitor routine executes until it requires a function that LOGON scheduling performs. At that time, the monitor notifies LOGON scheduling via the LOGON monitor ECB (LWAPECB).

Module	Label
IKJEFLB	

3 When notified by the LOGON monitor, LOGON scheduling performs one of two functions; the function performed is determined by the post code located in the monitor's ECB (LWAPECB):

Module	Label
IKJEFLB	WAITLIST
IKJEFLB	ENDJOB
IKJEFLB	

post code — **function performed by LOGON scheduling**

16 Schedules a terminal session as follows:

- Notifies the LOGON monitor (LWASECB-post code 16) to invoke the LOGON information routine IKJEFLH.
- Creates the job scheduling option list (JSOL) and chains it to the JSEL. The JSOL contains option flags that affect the scheduling of this terminal session.
- Moves the JCL card image chain (created by either the LOGON monitor or the pre-prompt exit) from subpool 1 to subpool 253.
- Invokes the initiator routine IEESB605 to schedule the terminal session.

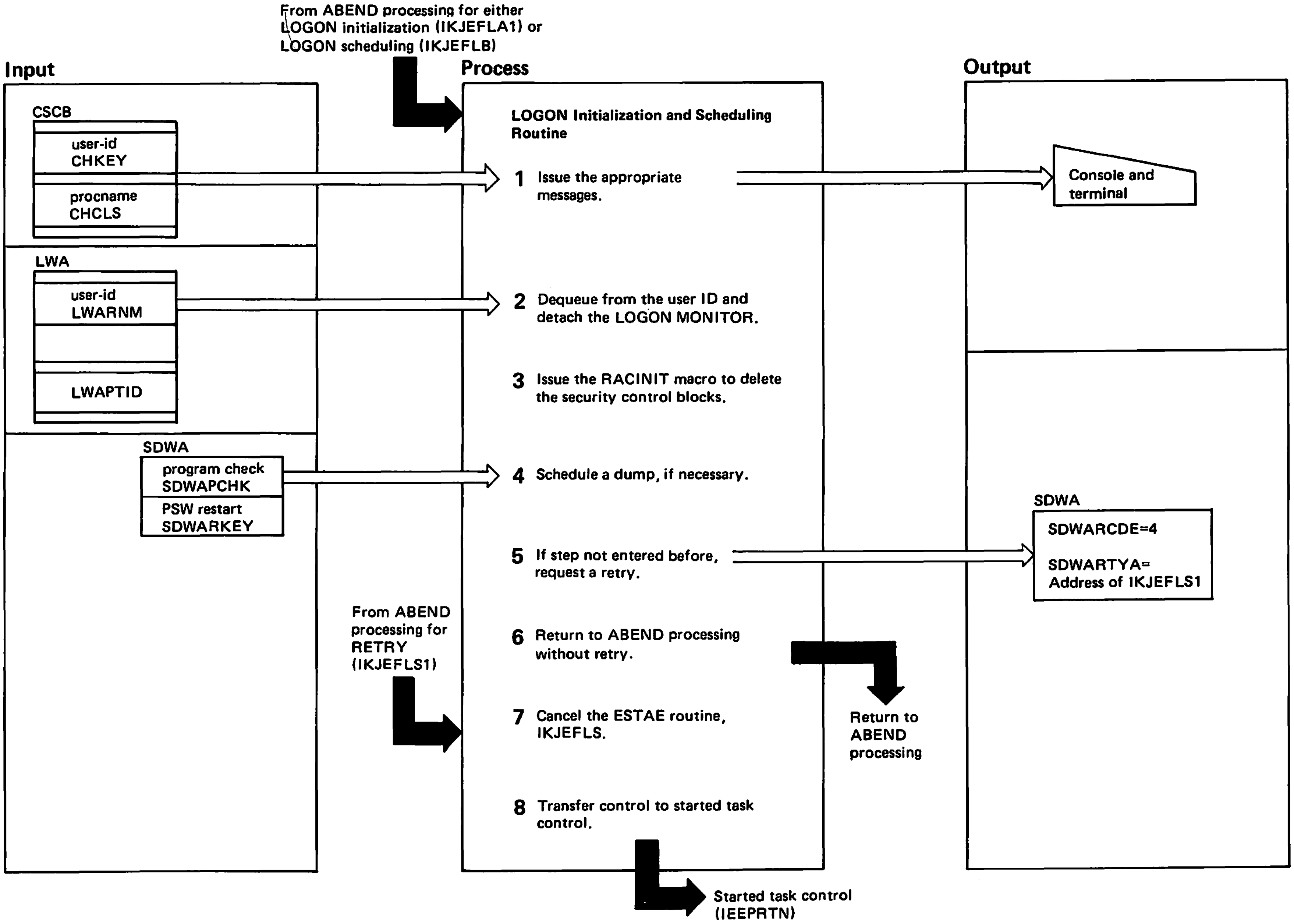
24 Terminates LOGON scheduling as follows (performed following a LOGOFF command):

- Notifies the LOGON monitor to terminate (LWASECB-post code 24).
- Issues a DETACH macro instruction for the LOGON monitor.
- Cancels the ESTAE environment protecting LOGON scheduling.
- Transfers control to STC routine IEEPRTN for CSCB clean-up.

4 VTIOC LOGOFF processing is performed by IKTLOGFF.

Module	Label
IKTLOGFF	

Diagram 9.3. LOGON Initialization and Scheduling Recovery Routine (IKJEFLS) (Part 1 of 2)


From ABEND processing for either LOGON initialization (IKJEFLA1) or LOGON scheduling (IKJEFLB)
Input
CSCB
user-id
CHKEY
procname
CHCLS
LWA
user-id
LWARNM
LWAPTID
SDWA
program check
SDWAPCHK
PSW restart
SDWARKEY
Process
Output
Console and terminal
SDWA
SDWARCDE=4
SDWARTYA=
Address of IKJEFLS1
From ABEND processing for RETRY (IKJEFLS1)
Return to ABEND processing
Started task control (IEEPRTN)

LOGON Initialization and Scheduling Routine

1 Issue the appropriate messages.

2 Dequeue from the user ID and detach the LOGON MONITOR.

3 Issue the RACINIT macro to delete the security control blocks.

4 Schedule a dump, if necessary.

5 If step not entered before, request a retry.

6 Return to ABEND processing without retry.

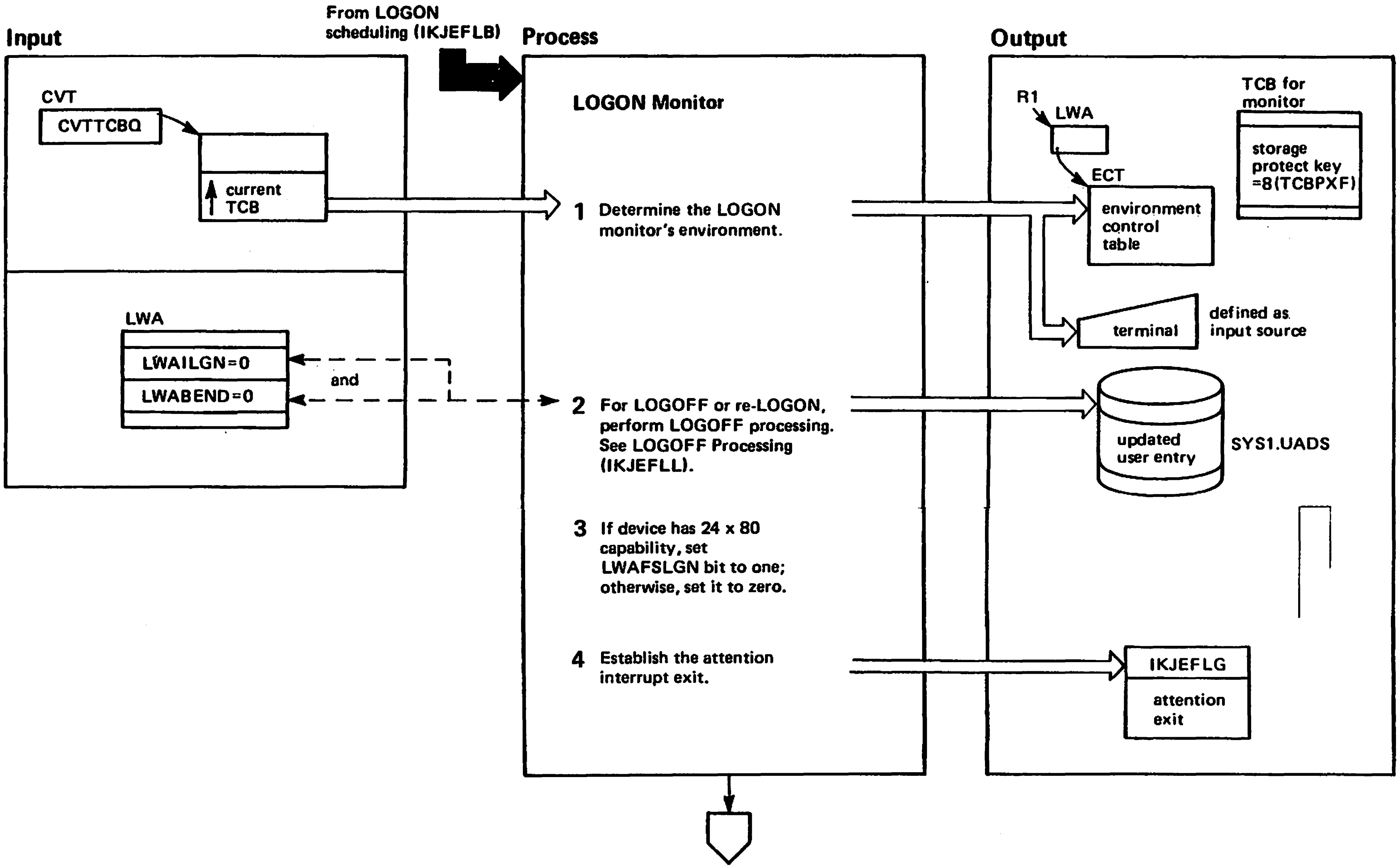
7 Cancel the ESTAE routine, IKJEFLS.

8 Transfer control to started task control.

Diagram 9.3. LOGON Initialization and Scheduling Recovery Routine (IKJEFLS) (Part 2 of 2)

	Extended Description	Module	Label
	LOGON Initialization creates an ESTAE environment that handles abends that can occur during initialization and scheduling.	IKJEFLA1	
1	Message IKJ601I is sent to the operator and message IKJ56452I is sent to the terminal.	IKJEFLS	
2	Dequeue from the user ID and detach the LOGON MONITOR. (The LWAPTID is the LOGON monitor TCB pointer.)		
3	If the user was in the RACF environment, IKJEFLS issues the RACINIT macro to delete the security related control blocks.		
4	Obtain a dump for a program check or PSW restart.		
5	If not a recursive abend, then indicate "RETRY" in the SDWA with the retry routine, IKJEFLS.		
6	Return to ABEND processing (IKJEFLS1) to possibly schedule a retry (see step 4).		
7	Cancel the ESTAE environment.	IKJEFLS1	
8	Transfer control to started task control, IEEPRTN, by using XCTL.		

Diagram 9.4. LOGON Monitor (IKJEFLC) (Part 1 of 4)



1 Determine the LOGON monitor's environment.

2 For LOGOFF or re-LOGON, perform LOGOFF processing. See LOGOFF Processing (IKJEFLL).

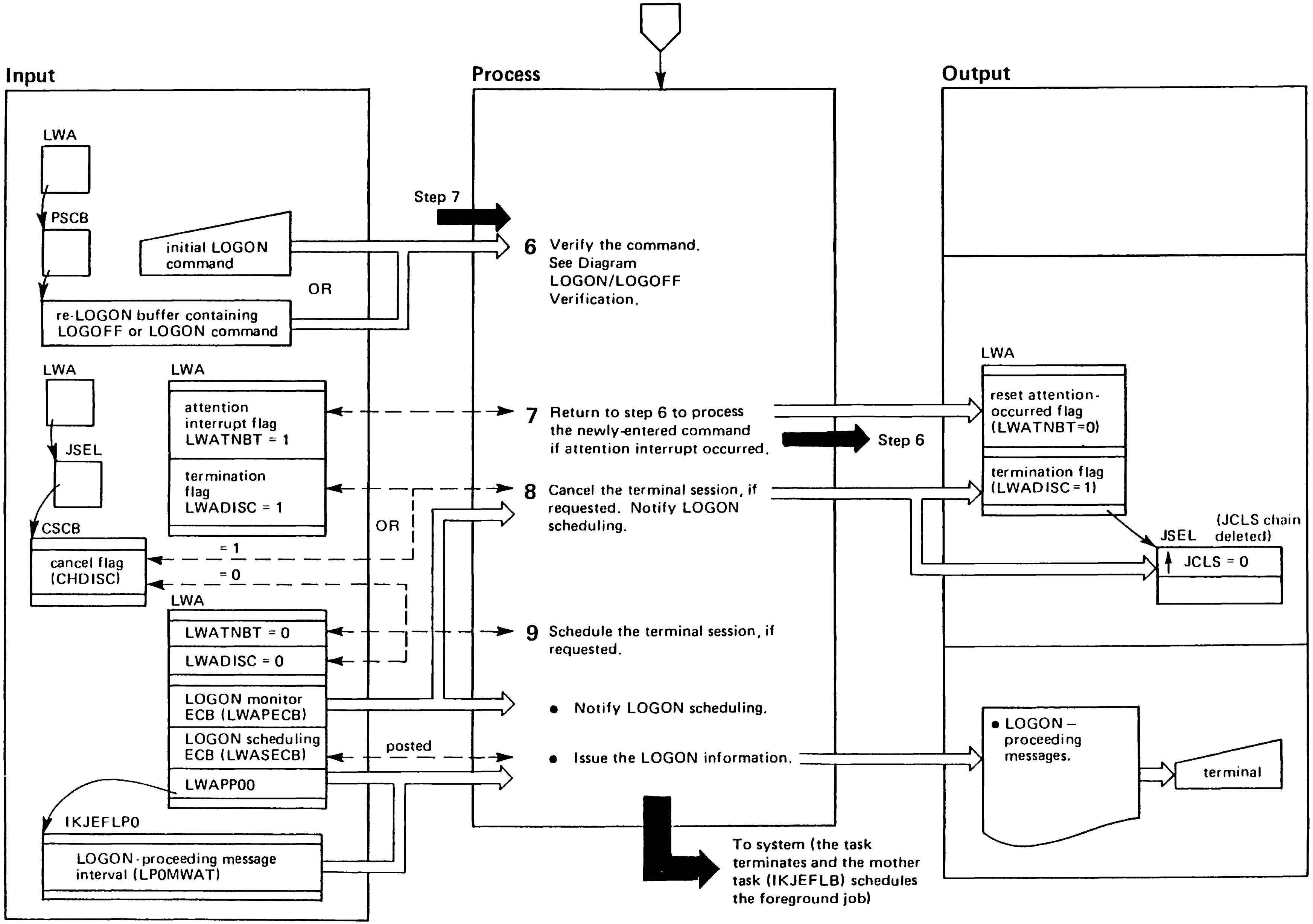
3 If device has 24 x 80 capability, set LWAFSLGN bit to one; otherwise, set it to zero.

4 Establish the attention interrupt exit.

Diagram 9.4. LOGON Monitor (IKJEFLC) (Part 2 of 4)

	Extended Description	Module	Label
	The LOGON monitor controls the processing that verifies the LOGON or LOGOFF command, and the processing that issues informational and prompting messages to the terminal. It notifies LOGON scheduling to schedule a terminal session or, in the case of a LOGOFF, to terminate the LOGON scheduling task. Some of the informational messages (that is, mail, notices, and LOGON-proceeding messages) are issued in parallel with the scheduling of the terminal session. All LOGON monitor messages are issued by the message handler IKJEFLGM.	IKJEFLC	
1	The LOGON monitor creates the environment control table (ECT), which contains information about I/O service routines the monitor will use. Also, the monitor sets its own storage protection key to 8. This allows the storage obtained by the monitor to be referenced by programs not executing in privileged state (for example, LISTBC and the pre-prompt exit). Finally, the monitor issues a STACK macro instruction to define the terminal as the first source of input for time-sharing commands.	IKJEFLC	INITWKAR STACK
2	LOGOFF processing calls IKJEFLL to update either the user's entry in SYS1.UADS or the user's entry in the RACF data base. LOGOFF processing will analyze the return codes from the job scheduling subroutine and from the terminal session. LOGOFF processing is not performed for an initial LOGON (LWAILGN = 1) or for recovery processing (LWABEND = 1). For more detail, refer to the Diagram "LOGOFF Processing."	IKJEFLL	
3	If the device has 24 x 80 capability, it can support full screen LOGON menus; set the LWAFSLGN bit to one. If the device does not have 24 x 80 capability, it cannot support full screen LOGON menus; set the LWAFSLGN bit to zero.		
4	The LOGON monitor issues a STAX macro instruction to establish a full screen or a line mode attention exit (IKJEFLG) that receives control when the terminal user causes an attention interruption by pressing the terminal's attention key. After causing the interruption, the terminal user may enter a question mark (?) to request second-level messages or may enter a new LOGON command to replace the one currently being processed.	IKJEFLC	TERMINAL

Diagram 9.4. LOGON Monitor (IKJEFLC) (Part 3 of 4)


Input
Process
Output
LWA
PSCB
initial LOGON command
OR
re-LOGON buffer containing LOGOFF or LOGON command
LWA
JSEL
CSCB
LWA
attention interrupt flag LWATNBT = 1
termination flag LWADISC = 1
OR
= 1
= 0
cancel flag (CHDISC)
LWA
LWATNBT = 0
LWADISC = 0
LOGON monitor ECB (LWAPECB)
LOGON scheduling ECB (LWASECB)
posted
LWAPP00
IKJEFLPO
LOGON-proceeding message interval (LPOMWAT)
Step 7
Step 6
LWA
reset attention-occurred flag (LWATNBT=0)
termination flag (LWADISC=1)
JSEL
(JCLS chain deleted)
JCLS = 0
• LOGON – proceeding messages.
terminal

6 Verify the command. See Diagram LOGON/LOGOFF Verification.

7 Return to step 6 to process the newly-entered command if attention interrupt occurred.

8 Cancel the terminal session, if requested. Notify LOGON scheduling.

9 Schedule the terminal session, if requested.

- Notify LOGON scheduling.
- Issue the LOGON information.

To system (the task terminates and the mother task (IKJEFLB) schedules the foreground job)

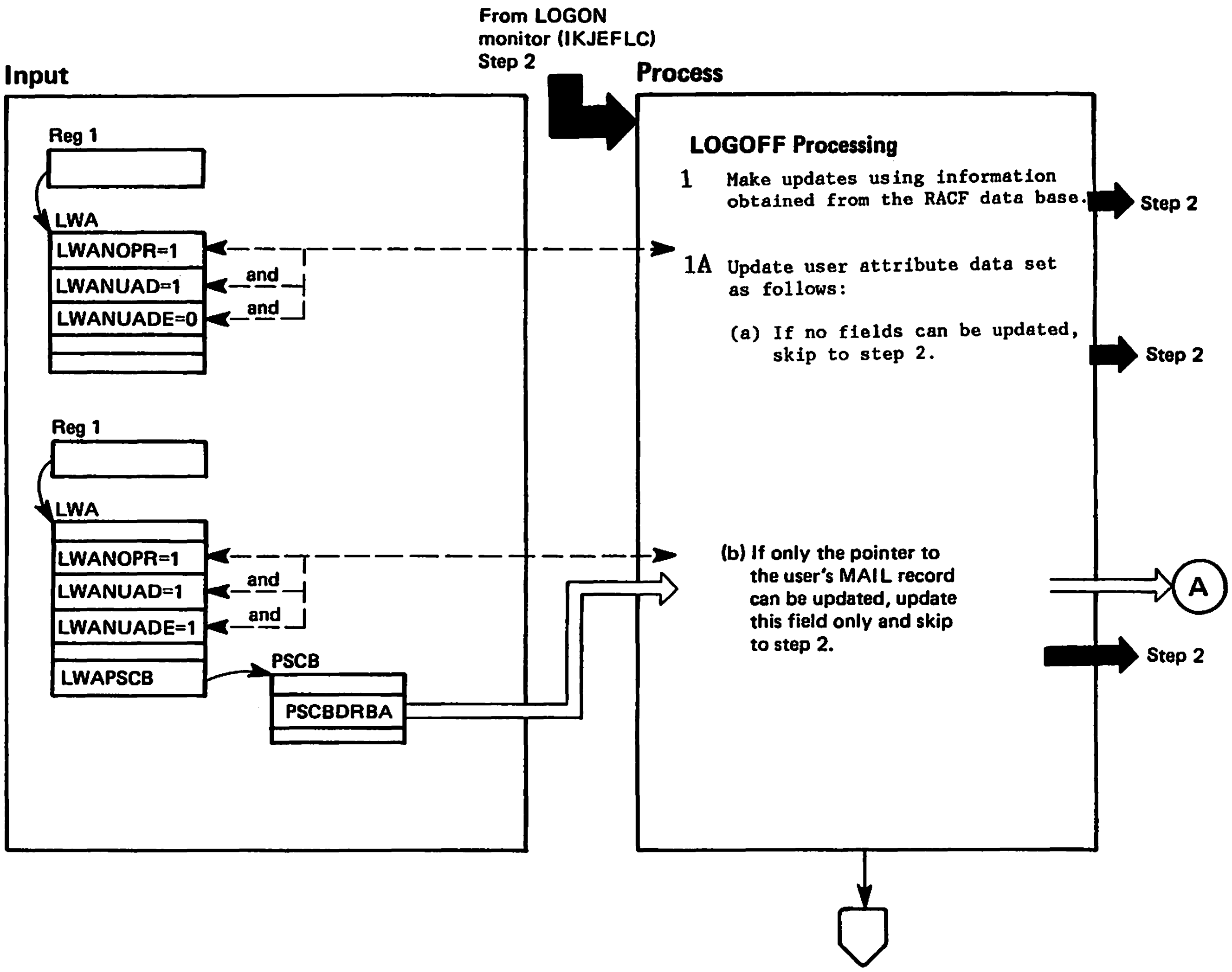
Diagram 9.4. LOGON Monitor (IKJEFLC) (Part 4 of 4)

	Extended Description	Module	Label
6	The LOGON monitor invokes LOGON/LOGOFF verification (IKJEFLE) to scan and parse the LOGON or LOGOFF command. For a LOGOFF or a re-LOGON, the command text is found in the re-LOGON buffer; otherwise, the command is obtained from the terminal. LOGON verification checks the user's authorization and LOGON parameters against either the information in SYS1.UADS (user attribute data set) or in the RACF data base and prompts the user to replace invalid or missing information. See Diagram "LOGON/LOGOFF Verification."	IKJEFLE IKJEFLEA	
7	If the user presses the terminal's attention key during LOGON processing, he may re-enter the LOGON command. In this case, the LOGON monitor re-invokes LOGON verification to analyze the newly-entered command. The attention interrupt flag is reset to zero to indicate that the interrupt has been completely processed.	IKJEFLC	GOTOLE
8	If the system operator cancels the terminal user, if the user has entered a LOGOFF command, or if the user has failed to enter a valid LOGON command, the LOGON monitor ends the terminal session as follows: • Issues an error message (IKJ56453I) to the terminal for an operator cancel. • Issues a null STAX macro instruction to cancel the LOGON attention exit. • Frees the environment control table (ECT). • Notifies LOGON scheduling to terminate (LWAPECB - post code 24). • Waits for notification from LOGON scheduling to terminate (LWASECB - post code 24). • Returns to the operating system via SVC 3.	IKJEFLC IKJEFLGM IKJEFLC	GOTOLE
9	After LOGON verification has processed a valid LOGON command, the LOGON monitor notifies LOGON scheduling to schedule the terminal session (LWAPECB - post code 16). LOGON scheduling invokes the job scheduling subroutine of the initiator, which attaches the terminal monitor program (TMP). When LOGON scheduling is ready to invoke the job scheduling subroutine, it notifies the LOGON monitor to continue its operation. (LWASECB - post code 16). At that time, the LOGON monitor calls the LOGON information routine, allowing it to execute in parallel with the scheduling of the terminal session. Then the routine sets the timer to expire at the interval specified in the module IKJEFLP0. The LOGON-proceeding message is issued repeatedly to the terminal at this timed interval until the initiator is ready to attach the TMP. At that time, the pre-TMP exit (IKJEFLJ) notifies the information routine (LWASECB - post code 20) that the LOGON scheduling process is complete. The routine then cancels the timer. Finally, the LOGON monitor terminates as follows: • Issues a null STAX macro instruction to cancel the LOGON attention exit. (Pressing the terminal attention key no longer has any effect on LOGON processing.) • Deletes the environment control table (ECT). • Returns to the operating system via SVC 3.	IKJEFLC IKJEFLH IKJEFLC	 CLEANUP

Error Processing

LOGON scheduling establishes the LOGON monitor's ESTAI environment via a parameter on the ATTACH macro instruction. If the LOGON monitor task terminates abnormally, the ESTAI routine IKJEFLGB receives control. See Diagram "LOGON Monitor Recovery."	IKJEFLB IKJEFLGB IKJEFLC
The LOGON monitor issues the STACK macro instruction to initialize the terminal as the source of input for commands. If this process encounters any errors, the LOGON monitor invokes the message handler to issue appropriate error messages to the terminal (IKJ56454I) or to the operator (IKJ608I). Also, the monitor turns on the LOGON-termination bit (LWADISC).	IKJEFLGM IKJEFLC IKJEFLGM

Diagram 9.5. LOGOFF Processing (IKJEFLL) (Part 1 of 6)



LOGOFF Processing

1 Make updates using information obtained from the RACF data base.

1A Update user attribute data set as follows:

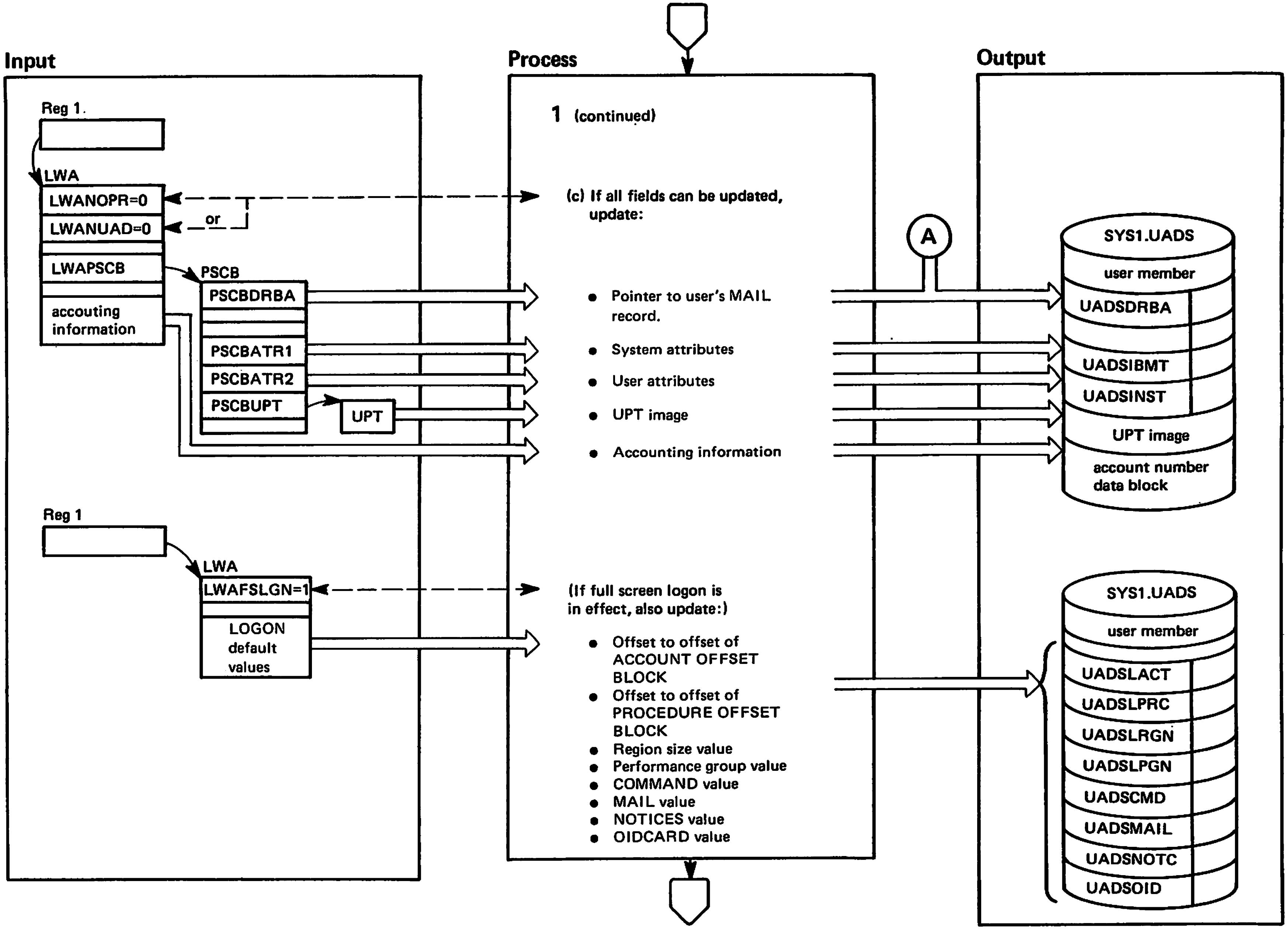
(a) If no fields can be updated, skip to step 2.

(b) If only the pointer to the user's MAIL record can be updated, update this field only and skip to step 2.

Diagram 9.5. LOGOFF Processing (IKJEFLL) (Part 2 of 6)

Extended Description	Module	Label
LOGOFF processing updates the terminal user's entry in either SYS1.UADS or the RACF data base and analyzes the return codes from the job scheduling subroutine (initiator) and from the last step of the terminal session. LOGOFF processing is performed for a LOGOFF command and for a re-LOGON. It is not performed for an initial LOGON (LWAILGN = 1) or for recovery processing (LWABEND = 1).	IKJEFLL	
1 If the user information was obtained from the RACF data base, do the following: a. If no fields can be updated, skip to step 2. b. If only the user's mail pointer is to be updated, update this field and skip to step 2. c. If the UPT still exists, store the UPT and user data. d. If full screen LOGON was used, store the performance group, TSO command, and TSO authority options. e. Call IKJEFRAF to update the information in the RACF data base. f. Skip to step 2.		
1A If the user information was obtained from the user attribute data set, do the following: a. If all of the following conditions are true, do not update any fields in the UADS: • The installation has supplied all of the LOGON information normally supplied by SYS1.UADS (LWANOPR = 1 and LWANUAD = 1). • LOGOFF processing should not update the pointer in the UADS to the user's MAIL directory record (LWANUADE = 0). b. If all of the following conditions are true, update only UADSDRBA by setting it to the value of PSCBDRBA: • The installation has supplied all of the LOGON information normally supplied by SYS1.UADS (LWANOPR = 1 and LWANUAD = 1). • LOGOFF processing should update the pointer in the UADS to the user's MAIL directory record (LWANUADE = 1).	IKJEFLL	UPDTUADS

Diagram 9.5. LOGOFF Processing (IKJEFLL) (Part 3 of 6)



1 (continued)

(c) If all fields can be updated, update:

- Pointer to user's MAIL record.
- System attributes
- User attributes
- UPT image
- Accounting information

(If full screen logon is in effect, also update:)

- Offset to offset of ACCOUNT OFFSET BLOCK
- Offset to offset of PROCEDURE OFFSET BLOCK
- Region size value
- Performance group value
- COMMAND value
- MAIL value
- NOTICES value
- OIDCARD value

Diagram 9.5. LOGOFF Processing (IKJEFLL) (Part 4 of 6)

	Extended Description	Module	Label
1	(continued)		

c. Updates to the pointer to the user's MAIL directory record are maintained in PSCBDRBA. Therefore, update UADSDRBA by setting it to the value of PSCBDRBA.

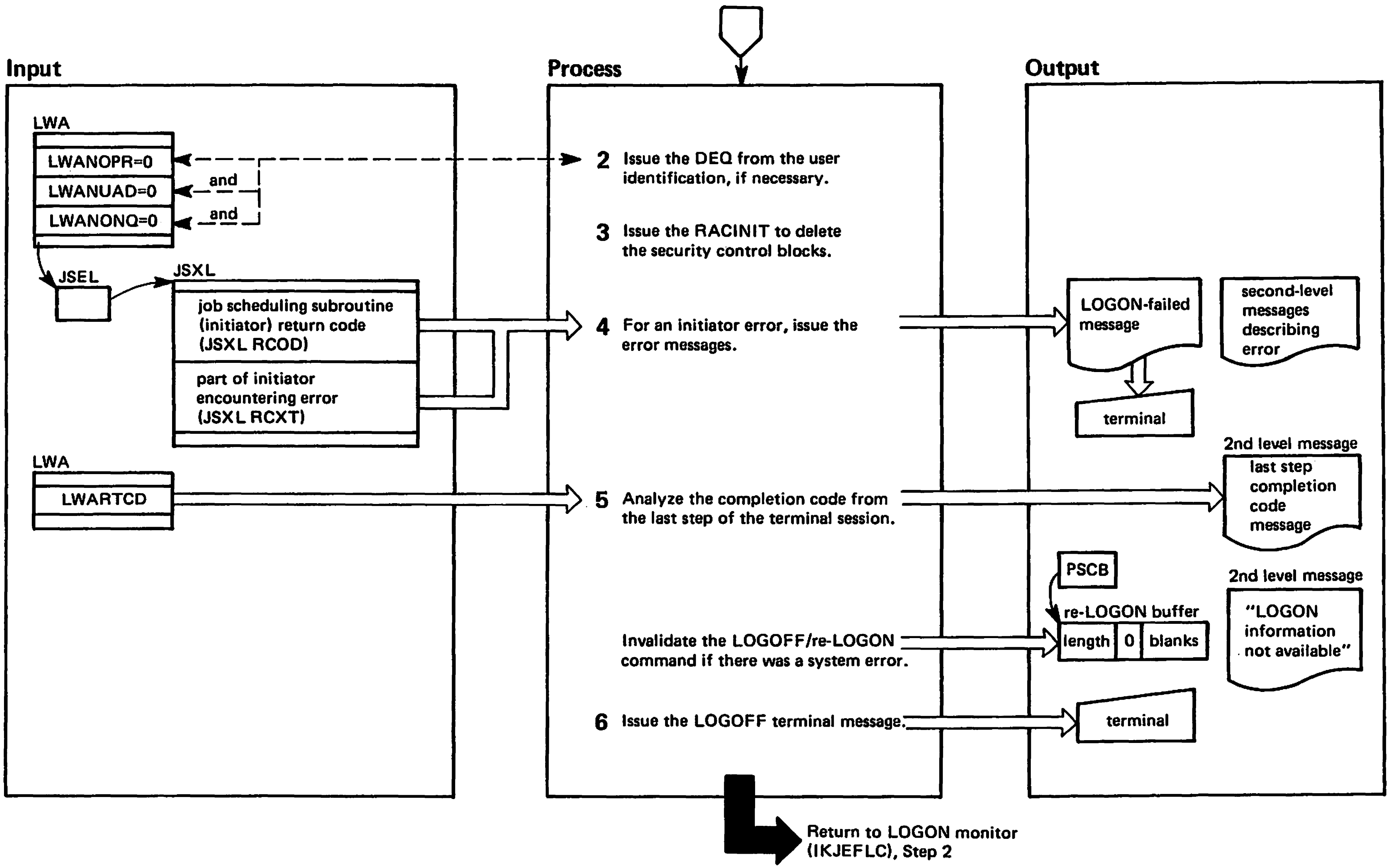
If any of the three bits LWAATR1, LWAATR2, and LWABUPT are off, the corresponding information (system attributes, user attributes, and the user profile, respectively) was not supplied by the installation. The information not supplied by the installation (and, therefore, subject to changes made via the PROFILE command) is updated by LOGOFF processing.

If LWAACCT≠0, the user's accounting information in SYS1.UADS is also updated. Accounting information consists of the following items: the length of the terminal session, the amount of processor time used, and the number of service units used.

Assuming the UADS can be read (LWANUAD=0 or LWANOPR=0), if full screen logon is in effect (LWAFSLGN=1), transfer the appropriate field values in the LWA to the corresponding fields in the UADS. Values transferred are:

- Offset to the offset of the ACCOUNT OFFSET BLOCK
- Offset to the offset of the PROCEDURE OFFSET BLOCK
- Region size
- Performance group value
- Command
- Whether the user elects to:
  - receive mail
  - receive notices
  - enter date via the operator identification card.

Diagram 9.5. LOGOFF Processing (IKJEFLL) (Part 5 of 6)


Input
Process
Output
LWA
LWANOPR=0
and
LWANUAD=0
and
LWANONQ=0
JSEL
JSXL
job scheduling subroutine (initiator) return code (JSXL RCOD)
part of initiator encountering error (JSXL RCXT)
LWA
LWARTCD
LOGON-failed message
second-level messages describing error
terminal
2nd level message
last step completion code message
PSCB
re-LOGON buffer
length
0
blanks
2nd level message
"LOGON information not available"
terminal

2 Issue the DEQ from the user identification, if necessary.

3 Issue the RACINIT to delete the security control blocks.

4 For an initiator error, issue the error messages.

5 Analyze the completion code from the last step of the terminal session.

Invalidate the LOGOFF/re-LOGON command if there was a system error.

6 Issue the LOGOFF terminal message.

Return to LOGON monitor (IKJEFLC), Step 2

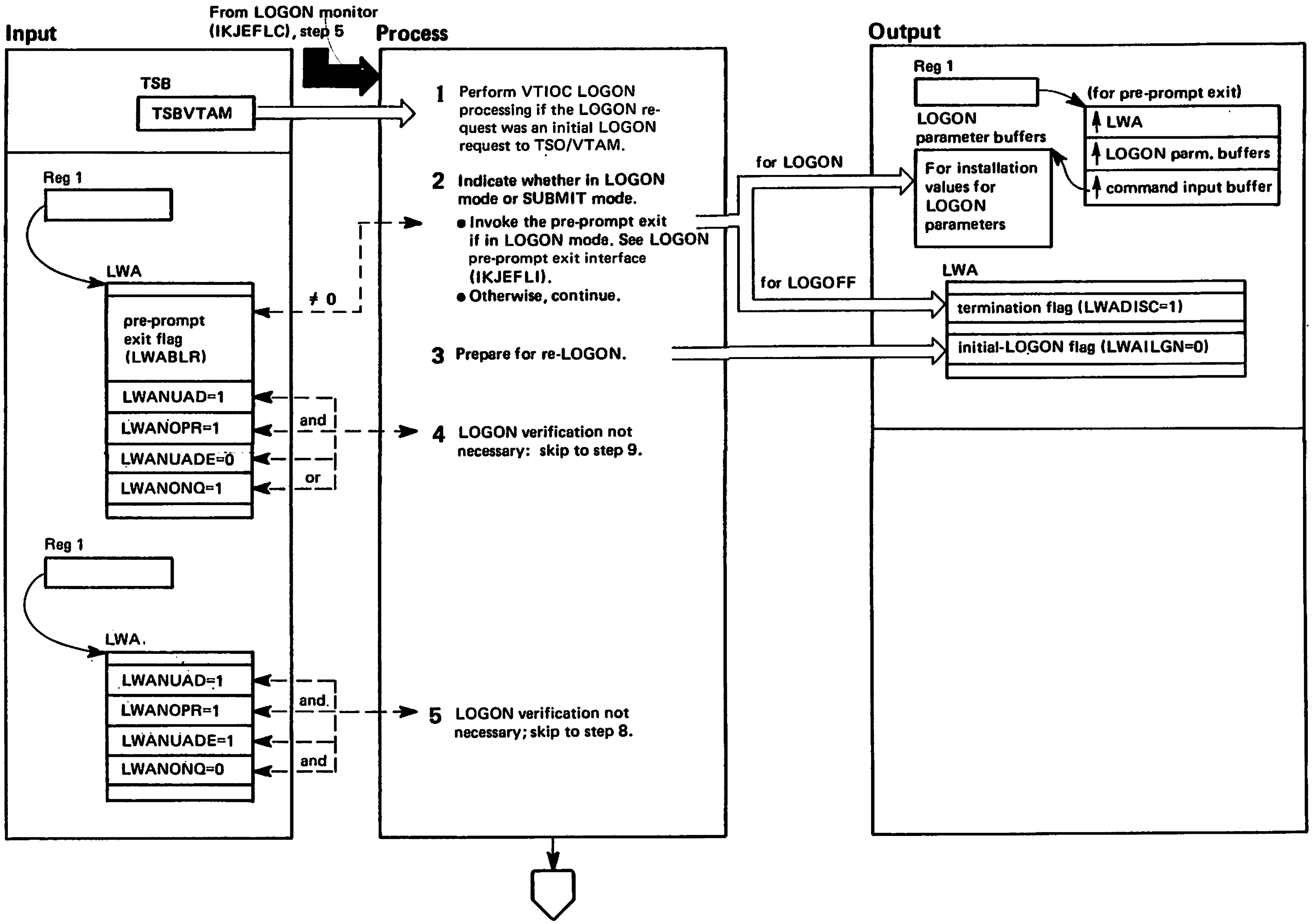
Diagram 9.5. LOGOFF Processing (IKJEFLL) (Part 6 of 6)

	Extended Description	Module	Label
2	LOGOFF processing must release the user identification resource that was obtained during LOGON verification. IKJEFLL calls IKJEFLIO to issue the DEQ macro instruction. If the three bits LWANOPR, LWANUAD, and LWANONQ are turned off, an ENQ was never issued on the user identification. In this case, a DEQ is not necessary.	IKJEFLL	DEQUSER
3	If the user was in the RACF environment, IKJEFLL issues the RACINIT macro to delete the security related control blocks.		
4	If the job scheduling subroutine encountered an error (LWARTCDI0), LOGOFF processing examines the field JSXLRCXT to determine what part of job scheduling failed. Next, it examines the fields JSXLRCOD and LWARCDE to determine the nature of the error. Finally, LOGOFF informs the message handler (IKJEFLGM) to build the appropriate second-level message (IKJ56457I to terminal).	IKJEFLL	
5	LOGOFF analyzes the return code from the last step of the terminal session (LWARTCD) and builds an appropriate second-level message (IKJ56470I to terminal) via the message handler. If the code is a system return code, the re-LOGON buffer is considered to be unusable and is filled with blanks. In this case, LOGON/LOGOFF verification must prompt the user for a LOGON or LOGOFF command. (See Diagram LOGON/LOGOFF Verification.)	IKJEFLL	
6	LOGOFF calls the LOGON time and date processor (IKJEFLPA) to set up the date and time-of-day buffers for the logged-off message. Then LOGOFF invokes the message handler to issue the logged-off message to the terminal (IKJ56470I).	IKJEFLL	LGMSETUP

Error Processing

Extended Description	Module
If, at any time, LOGOFF processing receives a return code from IKJEFLIO indicating an I/O error, an open error, or a service routine error, it issues an error message (IKJ56454I) to the terminal via the message handler and turns on the LOGON termination bit.	IKJEFLL

Diagram 9.6. LOGON/LOGOFF Verification (IKJEFLE and IKJEFLES) (Part 1 of 6)



1 Perform VTIOC LOGON processing if the LOGON request was an initial LOGON request to TSO/VTAM.

2 Indicate whether in LOGON mode or SUBMIT mode.

- Invoke the pre-prompt exit if in LOGON mode. See LOGON pre-prompt exit interface (IKJEFLI).
- Otherwise, continue.

3 Prepare for re-LOGON.

4 LOGON verification not necessary: skip to step 9.

5 LOGON verification not necessary; skip to step 8.

Diagram 9.6. LOGON/LOGOFF Verification (IKJEFLE and IKJEFLES) (Part 2 of 6)

	Extended Description	Module	Label
	LOGON/LOGOFF verification scans the LOGON or LOGOFF command and checks the LOGON parameters against the information in the user's member of the SYS1.UADS data set or the RACF data base. As the verification process is checking LOGON parameters, it records valid LOGON information in various control blocks. An optional installation exit (pre-prompt exit IKJEFLD) can replace any part or all of the verification processing. If the LOGON is valid, JCL card images (JOB and EXEC) that define the terminal session are built. When SUBMIT enters LOGON verification, the LOGON command is parsed and the results are returned to SUBMIT (IKJEFF08). SUBMIT then builds JCL statements to execute commands in the background. The pre-prompt exit interface will not be invoked.	IKJEFLE	
1	VTIOC LOGON processing is done only for an initial LOGON to TSO/VTAM, not for a re-LOGON or a LOGOFF.	IKTXLOG	
2	If the VCON for the installation exit (IKJEFLD) is non-zero (indicating an installation exit is present and link-edited into the LOGON load module), the interface routine IKJEFLI is invoked to initialize a parameter list for the exit. (See Diagram LOGON Pre-Prompt Exit Interface.) The interface does not pass control to pre-prompt exit (IKJEFLD) if the command is a LOGOFF or if LOGON is invoked from SUBMIT.	IKJEFLE	GOTOIER
3	The initial-LOGON flag is turned off following the first GETLINE macro instruction issued by LOGON/LOGOFF verification. Any subsequent LOGON command entered by the terminal user for the current address space is considered to be a re-LOGON.	IKJEFLE	
4	LOGON/LOGOFF verification returns to the LOGON monitor if the termination flag is on (LWADISC). If all of the following conditions are true, then the normal LOGON verification is bypassed; skip to step 9: • The pre-prompt exit has supplied all of the LOGON information (LWANOPR = 1). • The pre-prompt exit indicates that no verification is necessary (LWANUAD = 1). • One of the following has occurred: – SYS1.UADS or the RACF data base cannot be read to access the user's mail directory (RBA) (LWANUADE = 0). – SYS1.UADS or the RACF data base cannot be enqueued on (LWANONQ = 1).	IKJEFLE	
5	If all of the following conditions are true, then the normal LOGON verification is bypassed; skip to step 8 to read SYS1.UADS or the RACF data base for the user's mail directory (RBA): • The pre-prompt exit has supplied all of the LOGON information (LWANOPR = 1). • The pre-prompt exit indicates that no verification is necessary (LWANUAD = 1). • SYS1.UADS or the RACF data base can and should be read for RBA, the pointer to the user's mail directory block (LWANUADE = 1). • SYS1.UADS or the RACF data base can be enqueued on (LWANONQ = 0).	IKJEFLE	

Diagram 9.6. LOGON/LOGOFF Verification (IKJEFLE and IKJEFLES) (Part 3 of 6)

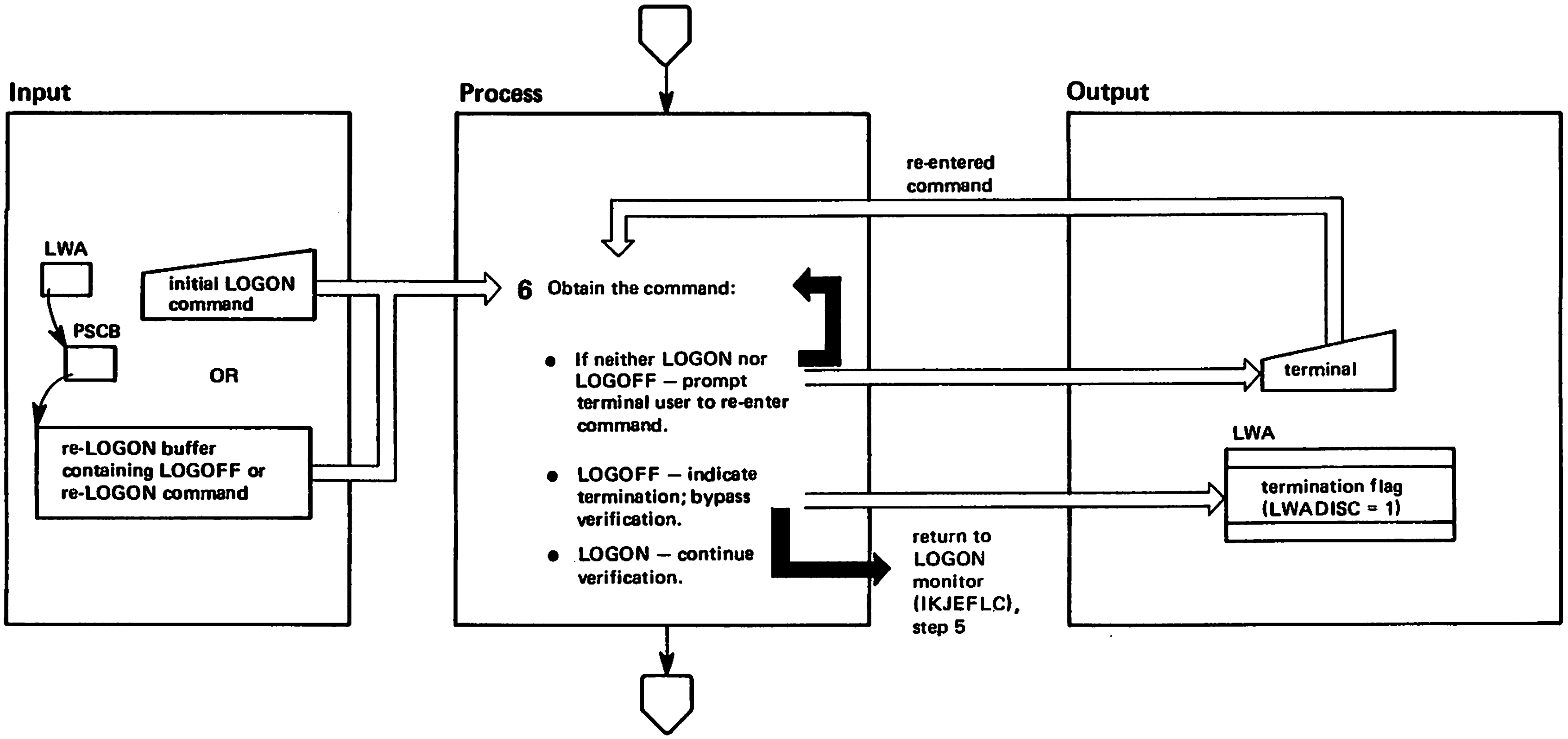
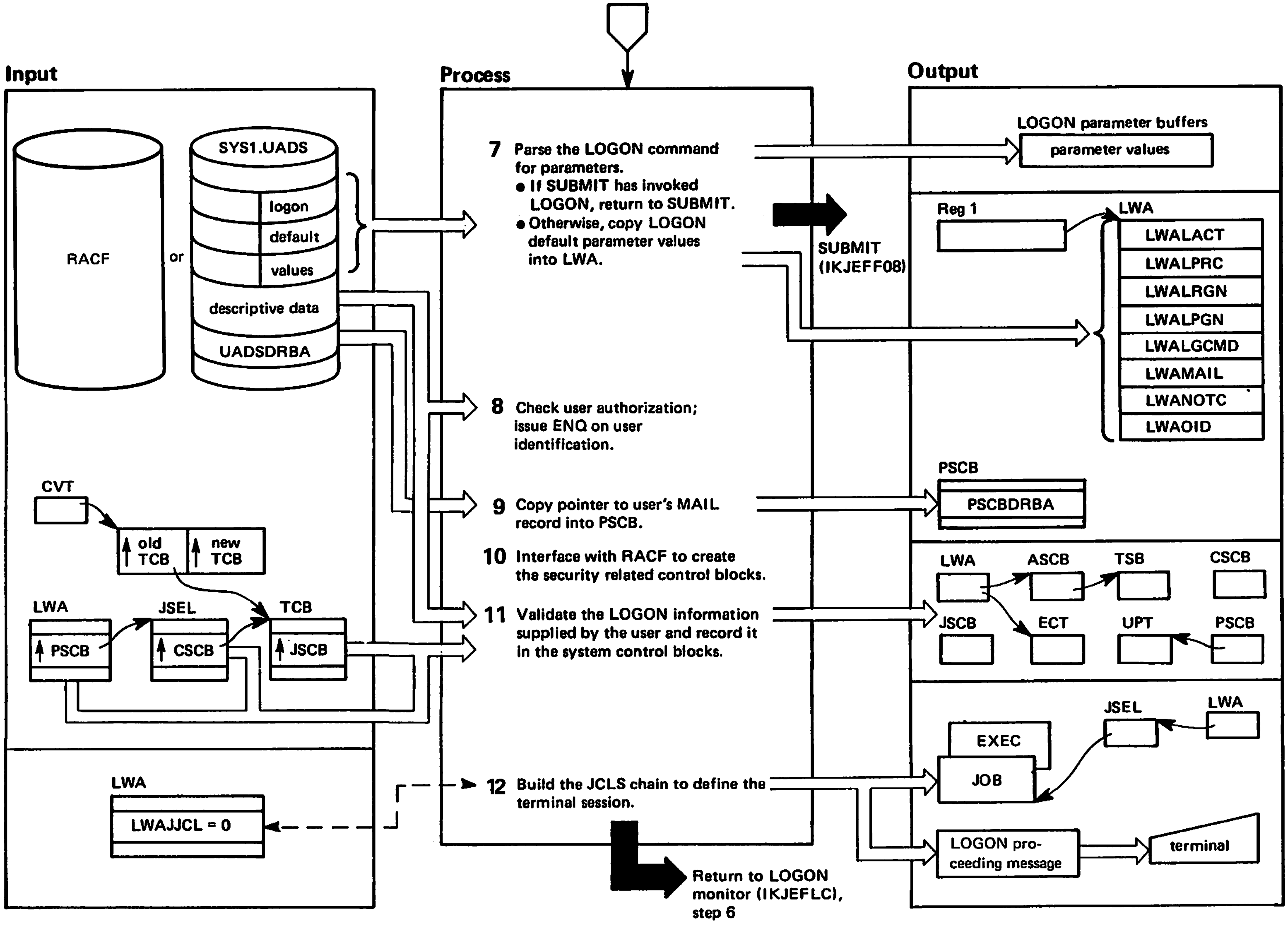

Input
Process
Output
LWA
PSCB
initial LOGON command
OR
re-LOGON buffer containing LOGOFF or re-LOGON command
6 Obtain the command:
● If neither LOGON nor LOGOFF – prompt terminal user to re-enter command.
● LOGOFF – indicate termination; bypass verification.
● LOGON – continue verification.
re-entered command
terminal
LWA
termination flag (LWADISC = 1)
return to LOGON monitor (IKJEFLC), step 5

Diagram 9.6. LOGON/LOGOFF Verification (IKJEFLE and IKJEFLES) (Part 4 of 6)

	Extended Description	Module	Label
6	After the command scan service routine (IKJSCAN) scans the command for LOGON or LOGOFF, the verification process continues as follows: • If neither command was found, the terminal user is prompted to enter LOGON or LOGOFF and the scan is repeated. • If the command was a LOGOFF, the verification process returns control to the caller, the LOGON monitor. For a LOGOFF HOLD (TSBHLDL = 1), terminal input/output control (TIOC) for TSO/TCAM or terminal control address space (TCAS) for TSO/VTAM keeps a line open to the terminal. If at any time a terminal line is accidentally disconnected, TIOC or VTIOC retains, for a time specified in IKJPRM00 of SYS1.PARMLIB, the control blocks and the address space used for the current terminal session. If the terminal user then enters a LOGON RECONNECT command with the same user identification as the retained address space, TIOC or VTIOC reinstates the user in that address space. • If the command was a LOGON, the verification process continues (see	IKJEFLEA	LOGONOFF

Diagram 9.6. LOGON/LOGOFF Verification (IKJEFLE and IKJEFLES) (Part 5 of 6)



7 Parse the LOGON command for parameters.
- If SUBMIT has invoked LOGON, return to SUBMIT.
- Otherwise, copy LOGON default parameter values into LWA.

8 Check user authorization; issue ENQ on user identification.

9 Copy pointer to user's MAIL record into PSCB.

10 Interface with RACF to create the security related control blocks.

11 Validate the LOGON information supplied by the user and record it in the system control blocks.

12 Build the JCLS chain to define the terminal session.

Return to LOGON monitor (IKJEFLC), step 6

Diagram 9.6. LOGON/LOGOFF Verification (IKJEFLE and IKJEFLES) (Part 6 of 6)

	Extended Description	Module	Label
7	The verification process invokes IKJPARSE to check the syntax of the LOGON command. If the command contains the RECONNECT parameter, LOGON determines whether the user identification is already assigned to an address space (one that TIOC or VTIOC retained following a disconnected line). If the user identification has an address space assigned to it, RACF is called to verify user and terminal access security and TIOC or VTIOC reinstates the user in the retained address space. If the user identification has no address space assigned to it, the LOGON RECONNECT is rejected. Then, if SUBMIT invoked LOGON, return to SUBMIT. If SUBMIT did not invoke LOGON, copy the LOGON parameter values obtained from either SYS1.UADS or the RACF data base into the corresponding fields in the LWA.	IKJEFLE	TSBSRCH
8	LOGON calls IKJEFLIO to issue an ENQ on the user identification resource. If the return code from IKJEFLIO indicates that the resource has already been obtained, LOGON verification reinvokes the pre-prompt exit if it exists. The installation can choose to authorize the user or cancel the LOGON process. If the user information is stored in the user attribute data set, LOGON verification calls IKJEFLIO to open SYS1.UADS and copy the member associated with the user identification on the LOGON command into real storage. LOGON verification then ensures that the user identification is authorized. The user identification and its length are stored in the PSCB (protected step control block).	IKJEFLE2 or IKJEFLE	
9	Set PSCBDRBA to the value of the user's mail directory (RBA). LISTBC uses PSCBDRBA to search for the user's mail directory record.		
10	The RACINIT macro is issued by IKJEFLE or IKJEFRAF, causing RACF to create security related control blocks associated with the user identification and password.		
11	LOGON verification compares the LOGON parameter values with the user information contained in SYS1.UADS or the RACF data base to check the validity of the LOGON parameters. If parameters are invalid or missing, LOGON verification prompts the user for correct parameters. The user's reply is re-parsed and verified. Verification checks the user's password, account number, procedure name, region size, and performance group. The system resources manager checks that the performance group is defined to the system and that the group can be used at this time. The job entry subsystem verifies that the destination choice (DEST parameter) defines a valid device for SYSOUT data set.	IKJEFLE or IKJEFLE2	IKJEFL2
	If the user is RACF defined, then password verification with the UADS is bypassed. Both the password and group identification are verified by RACF.	IKJEFL3	IKJEFRAF
	If the UPT is not defined in the TSO segment of the RACF database, an attempt will be made to obtain the UPT from the UADS and store it in the RACF database. If the UPT cannot be obtained, a default UPT is built.	IKJEFLE2	

12	If LWAJJCL = 1, the pre-prompt exit has supplied the JCL card images that define the terminal session. Otherwise, LOGON processing constructs the JCL card images as follows:
	//userid JOB 'account #',REGION = region size
	//procname EXEC procname,PERFORM = performance group
	where the user ID (user identification, account #, region size, and performance group are obtained from the LOGON parameters, from the user's member of SYS1.UADS, the RACF data base, or from the pre-prompt exit.
		IKJEFLEA	BUILDJCL

Error Processing

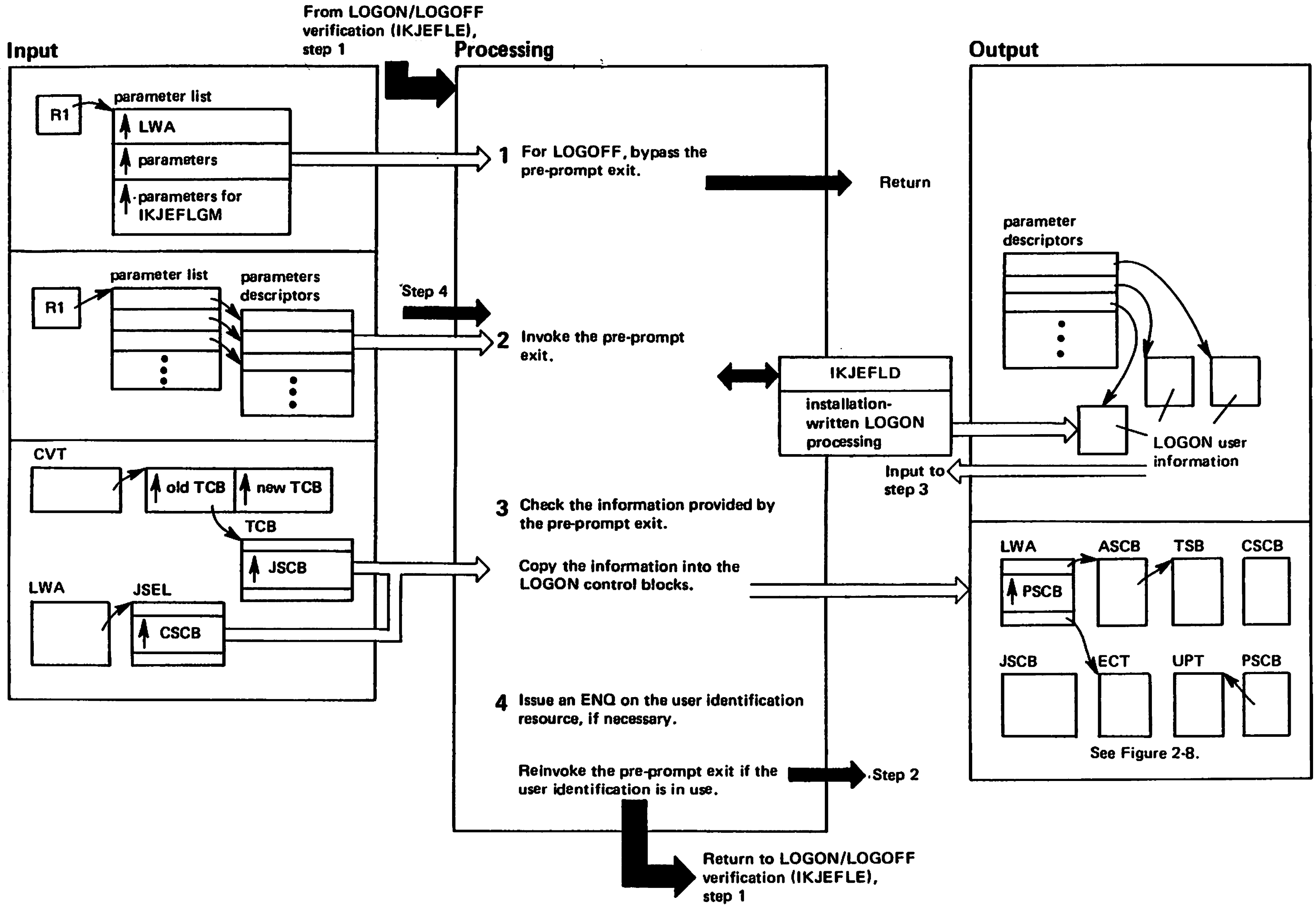
If the LOGON is an initial LOGON (LWAILGN = 1), and the address of the terminal input line is zero, LOGON verification obtains a line from the terminal (issues a GETLINE for the terminal). LOGON verification is part of the LOGON monitor task and, therefore, is protected by the monitor's ESTAI environment in case of an ABEND.	IKJEFLE IKJEFLGB	

The following data areas contain TSO user information supplied by the SYS1.UADS data, the RACF data base, the installation, or by the LOGON parameters:

Data Area Name	Field Name	Contents
ASCB	ASCBJBNS	Address of user identification.
CSCB	CHCLS	Procedure name for this LOGON.
	CHKEY	User identification.
ECT	ECT	Flags that control LISTBC processing.
EXEC card image		Procedure name for this LOGON. Performance group number.
Job card image		Account number. Region size.
JSEL	JSEL	Address of JCL card images.
JSOL	JSOLDEST	Default destination for SYSOUT data sets.
LWA	LWACTLS	Control switches set by the installation exit.
	LWADEST2	Default destination for SYSOUT data sets.
	LWAACCT	Offset of accounting information in SYS1.UADS.
	LWATCPU	Total CPU time used.
	LWATSRU	Total service units used.
	LWATCON	Total time connected to the system.
	LWARTCD	Completion code for the last step of the terminal session.
	LWALACT	Offset to offset of ACCOUNT OFFSET BLOCK in SYS1.UADS.
	LWALPRC	Offset to offset of PROCEDURE OFFSET BLOCK in SYS1.UADS.
	LWALRGN	Region size.
	LWALPGN	Performance group identification.
	LWALGCMD	Full screen LOGON menu command.
	LWAMAIL	MAIL/NOMAIL indicator.
	LWANOTC	NOTICE/NONOTICE indicator.
	LWAOID	Operator Identification Card prompt indicator.
PSCB	PSCBUSER	User identification.
	PSCBUSRL	Length of user identification.
	PSCBATR1	System attributes: switches that control use of OPERATOR, ACCOUNT, and SUBMIT commands, that indicate volume and mount authorization, and that define the attention key as the line-delete key.
	PSCBATR2	User attributes - reserved for installation use.
	PSCBGPNM	Generic unit name.
	PSCBRSZ	Region size.
	RBA	Address of user's mail directory block.
UPT	UPTSWS	Environmental switches.
	UPTNPRM	No-prompting switch.
	UPTMID	Switch that controls printing of message identifiers.
	UPTNCOM	Switch that controls SEND command authorization.
	UPTPAUS	Switch that indicates whether to pause for a "?."
	UPTALD	Switch that defines the attention key as the line-delete key.
	UPTMODE	Switch that controls printing of mode messages.
	UPTWTP	Switch that allows the user to receive WTP messages.
	UPTCDEL	Character-delete character.
	UPTLDEL	Line-delete character.
	UPTPREFX	Data set name prefix.
	UPTPREFL	Length of data set name prefix.

Figure 9-4. Data Areas Containing LOGON User Information

Diagram 9.7. LOGON Pre-prompt Exit Interface (IKJEFLI) (Part 1 of 2)



1 For LOGOFF, bypass the pre-prompt exit.

2 Invoke the pre-prompt exit.

3 Check the information provided by the pre-prompt exit.

Copy the information into the LOGON control blocks.

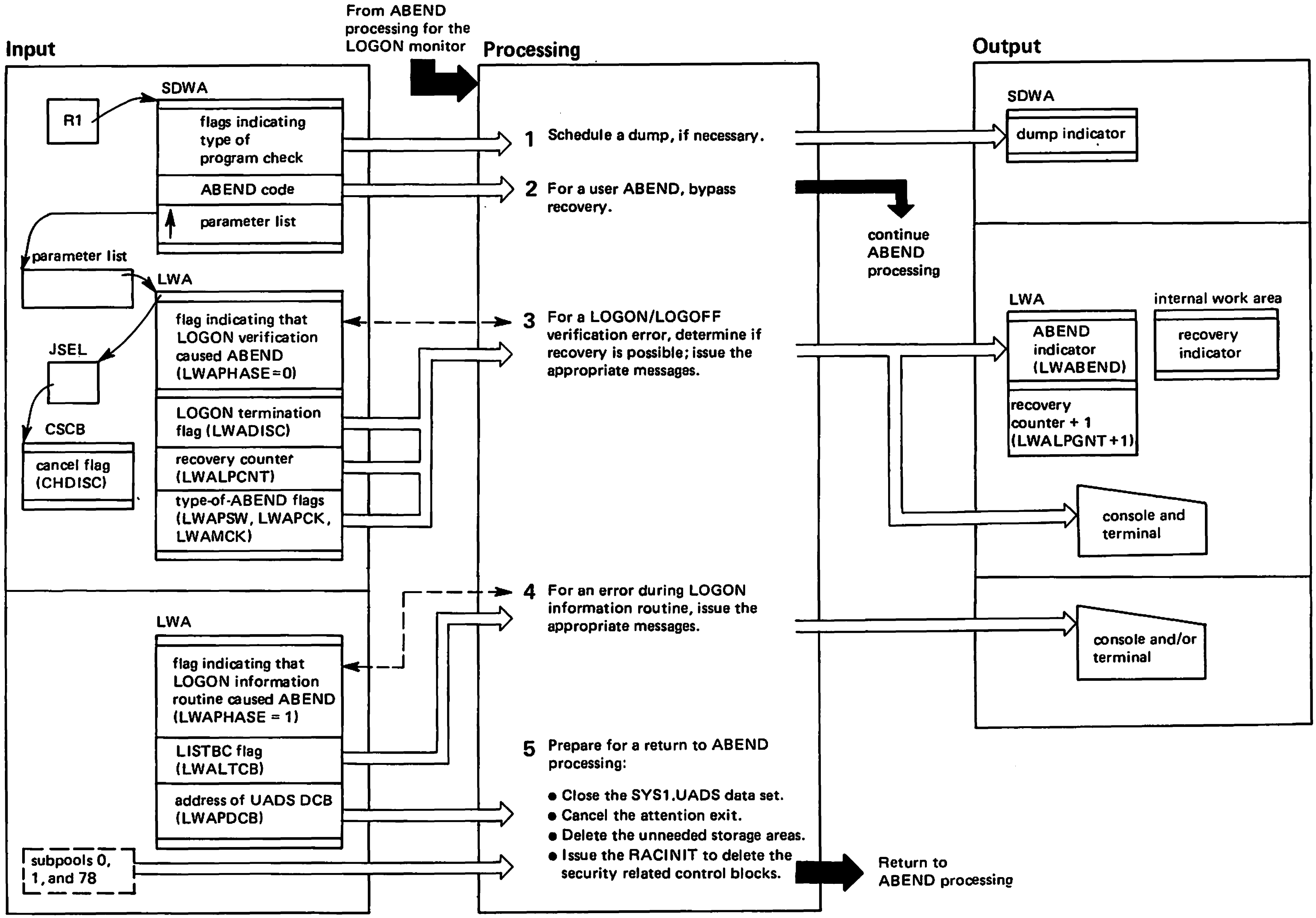
4 Issue an ENQ on the user identification resource, if necessary.

Reinvoke the pre-prompt exit if the user identification is in use.

Diagram 9.7. LOGON Pre-prompt Exit Interface (IKJEFLI) (Part 2 of 2)

Extended Description	Module	Label
The LOGON pre-prompt exit interface invokes the LOGON pre-prompt exit which is a routine written by the installation. The pre-prompt exit can provide LOGON information on behalf of the terminal user, verify the user's LOGON command, and collect accounting information. Any user information provided by the pre-prompt exit overrides the information stored in the user's member of the SYS1.UADS data set or the RACF data base. An installation can, if it wishes, replace all of the normal LOGON verification processing. For directions on writing the exit routine, refer to *TSO Extensions Customization.*	IKJEFL1	
1 The pre-prompt exit interface uses the command scan service routine (IKJSCAN) to determine if the command is a LOGON or LOGOFF. If it is a LOGOFF, the interface does not invoke the pre-prompt exit. Instead, it returns to its caller.	IKJEFLI	
2 The interface builds and passes to the pre-prompt exit a parameter list that defines those parameters the pre-prompt exit needs to verify the LOGON command and to provide LOGON information. Most of the addresses in the parameter list point to two-word descriptors. The first word of the descriptor contains the address of the actual parameter. The second word contains both the maximum length for the parameter and the actual length.		LI0100
3 After invoking the pre-prompt exit, the interface routine checks the parameter list for validity:	IKJEFLI	
• Ensures the parameter list is unchanged. • Ensures the parameter descriptors are unchanged, except for the field containing the actual length of the parameter. • Checks that the actual length of each parameter does not exceed the maximum length for the parameter.		LI800
If errors are discovered, the interface invokes the message handler (IKJEFLGM) to issue error messages and terminates the terminal session (LWADISC = 1). If no errors are found, the interface copies into the appropriate control blocks all user information provided by the pre-prompt exit. A control field in the LOGON work area (LWACTLS) contains bits that indicate what information the installation has provided.		
4 If the pre-prompt exit has specified in the LOGON work area that the terminal user is not to be prompted (LWANOPR = 1), that all LOGON information has been verified (LWANUAD = 1), and that an ENQ is to be issued (LWANONQ = 0), then the interface issues an ENQ on the user identification resource. If the resource is already in use, the pre-prompt exit is re-invoked to determine a course of action. The installation may choose to allow more than one user with the same user identification to be logged-on simultaneously (LWANONQ = 1). In this case, the interface does not issue an ENQ on the user identification resource. Or, the installation may, instead, choose to terminate the session (LWADISC = 1).	IKJEFLI	
Error Processing		
If either the LOGON pre-prompt exit interface (IKJEFLI) or the pre-prompt exit (IKJEFLD) cause an ABEND, the LOGON monitor's ESTAI routine IKJEFLGB is invoked by ABEND processing. In certain cases, the ESTAI routine schedules a re-attach of the LOGON monitor task. See Diagram "LOGON Monitor Recovery".	IKJEFLGB	

Diagram 9.8. LOGON Monitor Recovery (IKJEFLGB) (Part 1 of 2)


From ABEND processing for the LOGON monitor
Input
Processing
Output
R1
SDWA
flags indicating type of program check
ABEND code
parameter list
parameter list
LWA
flag indicating that LOGON verification caused ABEND (LWAPHASE=0)
LOGON termination flag (LWADISC)
recovery counter (LWALPCNT)
type-of-ABEND flags (LWAPSW, LWAPCK, LWAMCK)
JSEL
CSCB
cancel flag (CHDISC)
LWA
flag indicating that LOGON information routine caused ABEND (LWAPHASE = 1)
LISTBC flag (LWALTCB)
address of UADS DCB (LWAPDCB)
subpools 0, 1, and 78
SDWA
dump indicator
continue ABEND processing
LWA
ABEND indicator (LWABEND)
recovery counter + 1 (LWALPGNT+1)
internal work area
recovery indicator
console and terminal
console and/or terminal
Return to ABEND processing

1 Schedule a dump, if necessary.

2 For a user ABEND, bypass recovery.

3 For a LOGON/LOGOFF verification error, determine if recovery is possible; issue the appropriate messages.

4 For an error during LOGON information routine, issue the appropriate messages.

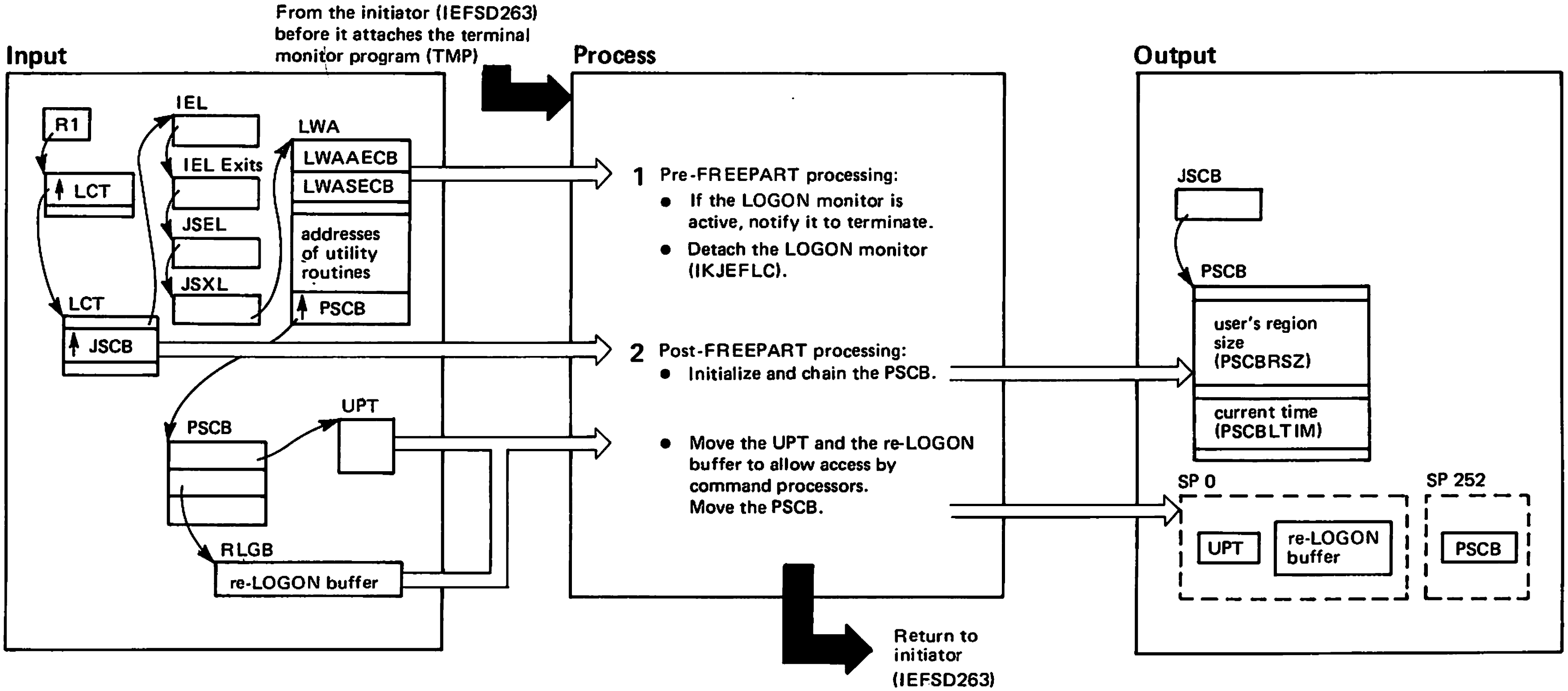
5 Prepare for a return to ABEND processing:

- Close the SYS1.UADS data set.
- Cancel the attention exit.
- Delete the unneeded storage areas.
- Issue the RACINIT to delete the security related control blocks.

Diagram 9.8. LOGON Monitor Recovery (IKJEFLGB) (Part 2 of 2)

	Extended Description	Module	Label
	The LOGON monitor recovery routine receives control from ABEND processing following the abnormal termination of the LOGON monitor task. LOGON monitor recovery is an ESTAI routine that was specified on the ATTACH macro instruction when the LOGON monitor was attached by the LOGON scheduling task. If possible, a retry of the LOGON monitor is attempted by informing the LOGON scheduling task to re-attach the LOGON monitor (LWABEND='1'B).	IKJEFLGB	
1	A dump is scheduled if the abnormal termination was the result of a program check or a PSW restart (an external interrupt from the operator).	IKJEFLGB	
2	If the ABEND code represents a user completion code, then recovery of the LOGON monitor task is not attempted. LOGON monitor recovery issues no error messages and passes control back to ABEND processing to continue the abnormal termination.	IKJEFLGB	
3	If the LOGON monitor abnormally terminated during LOGON/LOGOFF verification, recovery of the LOGON monitor task is scheduled (LWABEND=1). Recovery is not attempted in the following cases: • The system or the operator has canceled the terminal session (CHDISC=1). • The terminal session is scheduled for termination (LWADISC=1). • Four recoveries have already been attempted (LWALPCNT=4). • The current ABEND is the same type as the previous one (determined by checking bit settings in the LOGON work area: fields LWAPSW, LWAPCK, and LWAMCHK). LOGON monitor recovery builds and issues appropriate messages to the terminal and to the system operator. One set (IKJ56451I for the terminal and IKJ603I for the operator) is issued if the LOGON pre-prompt exit terminated abnormally (LWAINX1=1). Another set (IKJ56452I for the terminal and IKJ601I for the operator) is issued if LOGON/LOGOFF verification itself terminated abnormally (LWAINX1=0).	IKJEFLGB	PHASE1 MSGINIT
4	If the ABEND occurred after the user's LOGON information has been processed and the terminal session has been scheduled (that is, LWAPHASE=1), recovery may not be necessary. If LWAPHASE=1, the ABEND occurred either during LISTBC command processing or during the issuing of the LOGON-proceeding messages (issued by LOGON module IKJEFLH). If LISTBC caused the ABEND (LWALTCB=1), LOGON monitor recovery issues an error message to the terminal (IKJ56406I) and the LISTBC task terminates. In this case, the scheduling of the terminal session proceeds normally. If the LOGON module IKJEFLH caused the ABEND, LOGON monitor recovery does not schedule a re-attach of the monitor (LWABEND=0) but does issue error messages to the terminal (IKJ56452) and to the operator (IKJ601).	IKJEFLGB	PHASE2
5	LOGON monitor recovery performs exit processing as follows: • Calls IKJEFLIO to close the SYS1.UADS dataset, using the DCB address in the LOGON work area, and to issue a DEQ on the SYS1.UADS directory resource. If the DCB address is zero, IKJEFLIO does not issue the CLOSE macro instruction. • Issues a null STAX macro instruction to cancel the attention exit. Pressing the terminal attention key no longer has any effect on LOGON processing. • Frees the storage allocated to subpools 0, 1, and 78. • If the user was running in the RACF environment, the RACINIT macro is issued to delete security related control blocks.	IKJEFLGB	CLOSUADS FREECORE

Diagram 9.9. Pre-TMP Exit (IKJEFLJ) (Part 1 of 2)


From the initiator (IEFSD263) before it attaches the terminal monitor program (TMP)
Input
R1
LCT
IEL
IEL Exits
JSEL
JSXL
LCT
JSCB
LWA
LWAAECB
LWASECB
addresses of utility routines
PSCB
PSCB
UPT
RLGB
re-LOGON buffer
Process
Output
JSCB
PSCB
user's region size (PSCBRSZ)
current time (PSCBLTIM)
SP 0
UPT
re-LOGON buffer
SP 252
PSCB
Return to initiator (IEFSD263)

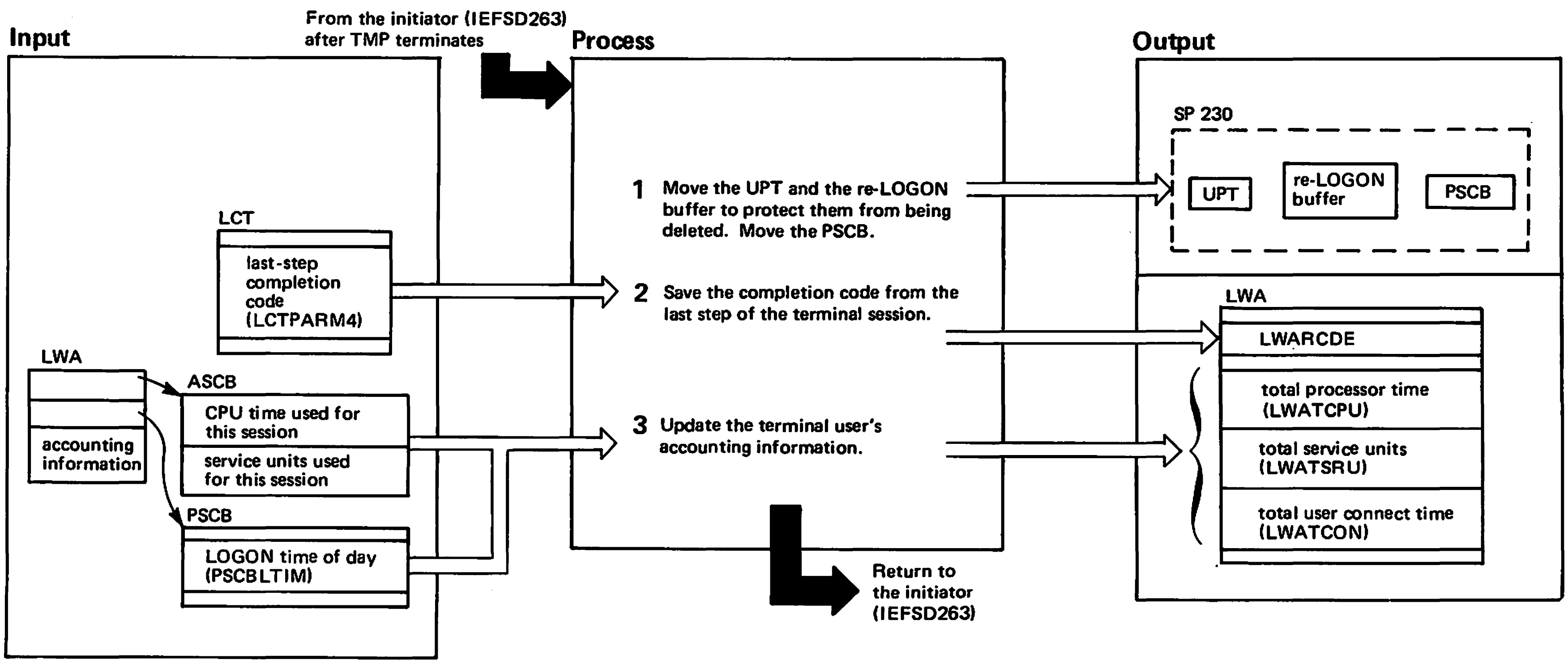
1 Pre-FREEPART processing:
- If the LOGON monitor is active, notify it to terminate.
- Detach the LOGON monitor (IKJEFLC).

2 Post-FREEPART processing:
- Initialize and chain the PSCB.
- Move the UPT and the re-LOGON buffer to allow access by command processors. Move the PSCB.

Diagram 9.9. Pre-TMP Exit (IKJEFLJ) (Part 2 of 2)

Extended Description	Module	Label
The initiator (IEFSD263) invokes the pre-TMP exit before attaching the terminal monitor program (TMP); it invokes the post-TMP exit after the TMP terminates. The pre-TMP exit prepares for the terminal session to begin by notifying the LOGON monitor task to terminate. The pre-TMP exit has two parts; an entry point name is assigned to each part. The first part is invoked before the initiator issues the FREEPART macro instruction (pre-FREEPART processing). The second part is invoked following the FREEPART (post-FREEPART processing).	IKJEFLJ	
1 This step represents pre-FREEPART processing. It is performed before initiator issues the FREEPART macro instruction. Since the LOGON monitor task may still be active, the data areas it uses must not be deleted (by FREEPART) until the task is notified to terminate. • Pre-FREEPART processing notifies the LOGON monitor task to terminate (LWASECB - post code 20). When the monitor task terminates, it notifies pre-FREEPART processing to continue (LWAPECB - post code 20). See LOGON Monitor (IKJEFLC). • The system initiated cancel (SIC) is notified that the TMP was executing when the line dropped or the user canceled. SIC will then notify the post-TMP exit to free other users who are waiting on this memory. For example, SEND W/WAIT option sent to a canceled memory can cause the sender to wait forever unless the post-TMP exit frees the sender.	IKJEFLJ	IKJLM1
2 This step represents post-FREEPART processing. It is performed after the initiator issues the FREEPART macro instruction. Post-FREEPART processing now can move the UPT and the re-LOGON buffer to subpool 0 (which is deleted by the FREEPART). • Post-FREEPART processing invokes the SWA manager to obtain the user's region size from the step control block (SCB). The region size is stored in the protected step control block (PSCB). If the SCT indicates that the terminal session is a job with more than one step, post-FREEPART processing passes a non-zero return code back to the initiator, which then terminates the job. The current time of day is also stored in the PSCB for later use in computing the length of the terminal session. • The UPT and the re-LOGON buffer are moved to subpool 0 (a non-protected subpool) so that the command processors may alter them during the terminal session. The PSCB is moved to subpool 252; the command processors cannot alter data areas in subpool 252.	IKJEFLJ	IKJLJ1

Diagram 9.10. Post-TMP Exit (IKJEFLK) (Part 1 of 2)



1 Move the UPT and the re-LOGON buffer to protect them from being deleted. Move the PSCB.

2 Save the completion code from the last step of the terminal session.

3 Update the terminal user's accounting information.

Diagram 9.10. Post-TMP Exit (IKJEFLK) (Part 2 of 2)

Extended Description		Module	Label
The initiator (IEFSD263) invokes the post-TMP exit after the TMP terminates. The post-TMP exit saves the completion code from the last step of the terminal session and updates the user's accounting information in the LOGON work area. Then, the initiator performs termination processing and passes control back to the LOGON scheduling task.			
1	The post-TMP exit moves the UPT and the re-LOGON buffer from subpool 0 to subpool 230 to prevent job scheduling from deleting them during job termination. The PSCB is also moved to subpool 230.	IKJEFLK	IKJLK1
2	The post-TMP exit saves the completion code from the last step of the terminal session, obtaining it from the linkage control table (LCT). The completion code is later analyzed by LOGOFF processing to determine if the terminal session terminated abnormally. See Diagram "LOGOFF Processing."	IKJEFLK	IKJLK1
3	The post-TMP exit updates the accounting information in the LOGON work area to account for the system resources used during the terminal session that is now terminating.	IKJEFLK	
Error Processing		IKJEFLJ,K	
If either the pre-TMP exit or the post-TMP exit causes an ABEND, LOGON scheduling's ESTAE routine IKJEFLS is invoked by ABEND processing. The function of this ESTAE routine is described under error processing in the diagram "LOGON Initialization and Scheduling".			

Chapter 1. EDIT Command Processing

This chapter describes the internal logic and organization of the EDIT program. It helps the programmer follow the internal operation of a program and locate a malfunction. This chapter gives references to specific functions the programmer can use to find information in program listings without having to scan them for the data he wants.

The EDIT command is described using method of operation diagrams, a program organization chart, a directory, a data area usage chart, and a diagnostic aids section.

The Program Organization section contains a list of the EDIT modules and their functions, and charts showing the flow of control from one module to another.

The Directory contains a module cross reference for the EDIT command and its subcommands. It cross references load module, object module, entry point, and alias name.

Chapter 24, "Data Area Usage," contains a list of data areas used by the EDIT command processor, and a description of the syntax checker control blocks, communications area, and option word. The data area descriptions can be found in *MVS/Extended Architecture Data Areas.*

For a description of how to use the TEST command to diagnose errors in the EDIT program, see*TSO Extensions System Diagnosis: Guide and Index.* There is a diagnostic aid description of the TSO terminal messages in *TSO Messages.*

Overview

The EDIT command processor is a part of TSO/E, the time sharing subsystem of MVS. The EDIT command processor performs the functions of the EDIT command and subcommands.

EDIT resides on SYS1.CMDLIB. When the user enters the EDIT command and operands, a copy of the EDIT program is loaded into the user's address space. The EDIT data set may be new or an existing data set; in either case, the name of the data set is that specified in the EDIT command.

The EDIT command enables the user to create data sets and to modify them by adding, replacing, and deleting records in the data sets. A data set can consist of text or programming language source statements. The user can work on a data set in either input mode or edit mode.

In input mode, the user enters successive lines of data. One line of input becomes one record in the data set. Services for translating tabulation characters to blanks, interpreting character and line delete characters, and translating lowercase characters to uppercase are available in the input mode. Users can request programming language syntax checkers to process source statements as they enter them.

In edit mode, the user enters subcommands to point to particular records, to modify or renumber records, to add or delete records, to control editing of input, or to compile and execute a program. While in the edit mode, EDIT keeps track of the user's position in the data set by means of the current line pointer. EDIT provides subcommands permitting the user to move the current line pointer within the data set. Once the current line pointer is positioned at a particular record, the user can issue the subcommand appropriate for the function needed.

Chapter 10. MVSSERV Command Processing

This chapter describes how to analyze messages, an external trace data set, and dumps to diagnose a possible MVSSERV error. For an overview of the function of the MVSSERV command in IBM System/370 to IBM Personal Computer Enhanced Connectivity Facilities, see *TSO Extensions Command Reference*.

For a complete description of MVSSERV macros, control blocks, and return codes, see *TSO Extensions Programmer's Guide to the Server-Requester Programming Interface for MVS/XA*.

Diagnosing an MVSSERV Error

You can use the following information to diagnose an MVSSERV error:

- Messages
- External trace data set
- Dumps.

This section explains how to use this information to identify a failing component. If you encounter a problem that you cannot fix, refer to *TSO Extensions System Diagnosis: Guide and Index* for a description of how to report the problem to IBM.

Messages

During MVSSERV processing, messages can be sent to the terminal or to a trace data set, or both. You must allocate a trace data set (ddname = CHSTRACE) to receive messages during your MVSSERV session, for example:

```
ALLOCATE F(CHSTRACE)  da('myID.dsname') LRECL(80) RECFM(FB)
```

You can determine if a message is from MVSSERV by checking the message ID. MVSSERV message IDs begin with the letters "CHS." An MVSSERV message may be informational (last character of the message ID is "I") or it may be an error message (last character is "E"). For example, "CHSTCA20E" is an MVSSERV error message.

Error messages require user action because they cause MVSSERV to end abnormally. Refer to *TSO Messages* for information about the action to take for each error message.

You can obtain online help for MVSSERV terminal messages by using the message ID with the HELP command after MVSSERV has ended:

```
HELP MVSSERV MSG(message ID)
```

External Trace Data Set

The external trace data set is a preallocated TSO data set that is used to record informational and error messages issued during an MVSSERV session. The messages, which are produced by various MVSSERV modules, can be used to determine what sequence of events may have led to a failure.

Before using the MVSSERV command, you must allocate the external trace data set to the ddname CHSTRACE with parameters specifying a record length of 80 bytes and a record format that is either fixed or fixed blocked. When you enter the MVSSERV command, you can specify a trace option on the command line with the value NOTRACE (default), TRACE, or IOTRACE. If you specify NOTRACE, MVSSERV does not open the data set and no MVSSERV module attempts to issue any messages to this data set.

If you specify TRACE or IOTRACE, MVSSERV attempts to open the data set allocated to CHSTRACE. If the open fails, internal flags are set to specify NOTRACE and processing continues. If the open is successful, recording to this data set is active for the duration of the MVSSERV session.

NOTE: For any failure, MVSSERV issues diagnostic messages to the external trace data set *only* if the TRACE or IOTRACE command option is in effect.

The external trace data set contains sequential messages issued during the MVSSERV session. These messages are followed by the internal execution path trace table.

Internal Execution Path Trace Table

An entry is made in the internal execution path trace table whenever one MVSSERV module calls another. Thus, the internal execution path trace table provides a history of module calls. Internal execution path trace table entries are of the form "Txxx" or "Txxxx," such as "TIOR" or "TRUTR."

A total of 256 entries is provided in the internal execution path trace table. Though they are sequential, the entries do not necessarily begin at zero and end at 255 because this is a wrap-around data set. That is, after the 256th entry is made, the data set begins overwriting itself starting from zero again.

Message CHSTTP01I always precedes the internal execution path trace table. This message states that the internal trace table follows and provides the number of the last entry in the table.

To read the history of module calls correctly, you begin with the last entry (for example, number 18 in the trace data set illustration in Figure 10-1) and proceed in descending order. When you reach entry 000, you go to entry 255 and continue in descending order until you reach the identified last entry again.

TRACE Option

You use the TRACE option of the MVSSERV command to record the following information in the external trace data set:

- Connectivity Programming Request Block (CPRB) requests and replies.
- VM/PC allocate messages.
- Internal execution path trace table.
- Informational and error messages. These messages provide data about the MVSSERV environment that was created, and a history of service requests for the current session.

Entries are made in the external trace data set for other MVSSERV processing, such as pressing the PF3 key to end an MVSSERV session. A sample trace data set obtained using the MVSSERV TRACE option is shown in Figure 10-1.

Note that message CHSTRR01I (see arrow) does not display the contents of the CPRB because the TRACE option does not record control block data or the actual communication transmission data that is sent and received.

```
      CHSCMI02I The control unit supports Read Partitioned Queries.
      CHSTCA13I DFT access method driver is active.
----> CHSTRR01I CPRB request at 12:37:07 server=SERVER2  function=0001:
      CHSRUTR06I Server request failed; SERVER2 is in an inactive task.
      CHSDCOM09I User pressed the PF3 key, requesting termination.
      CHSCPS08I MVSSERV is ending.
      CHSTTP01I Internal trace table follows.  Last entry is 018:
      CHSTTP02I 000 TIOR    001 TIOR    002 TIOR    003 TIOR
      CHSTTP02I 004 TIOR    005 TIOR    006 TIOR    007 TIOR
      CHSTTP02I 008 TIOR    009 TIOR    010 TIOR    011 TIOR
      CHSTTP02I 012 TIOR    013 TIOR    014 TIOR    015 TIPM
      CHSTTP02I 016 TIOR    017 TIOR    018 TTTP    019 TIOR
      CHSTTP02I 020 TSRV    021 TRUTR   022 TRUTR   023 TRUTR
      CHSTTP02I 024 TRUTR   025 TCMI    026 TLMP    027 TIOR
      CHSTTP02I 028 TDCA    029 HRES    030 TDCOM   031 TCH7
      CHSTTP02I 032 TC7H    033 PACK    034 TINF    035 TTRL
      CHSTTP02I 036 TLMP    037 TIOR    038 HQNL    039 TDCOM
      CHSTTP02I 040 TCH7    041 TC7H    042 PRNL    043 TINF
      CHSTTP02I 044 TTRL    045 TLMP    046 TIOR    047 HQNL
      CHSTTP02I 048 TDCOM   049 TCH7    050 TC7H    051 PRNL
      CHSTTP02I 052 TINF    053 TTRL    054 TLMP    055 TIOR
         .      .           .           .           .
         .      .           .           .           .
         .      .           .           .           .
      CHSTTP02I 252 HQNL    253 TDCOM   254 TCH7    255 TC7H
```

Figure 10-1. Sample Trace Data Set Obtained Using MVSSERV TRACE Option

IOTRACE Option

You use the IOTRACE option of the MVSSERV command to record more comprehensive data to the external trace data set:

- All data provided by the TRACE option.
- The control block associated with each service request (request CPRB).
- The actual data that is transmitted and received in communication transmissions.
- VM/PC communication header data.
- Structured fields associated with DFT communication protocol.

A sample trace data set obtained using the MVSSERV IOTRACE option is shown in Figure 10-2.

Note that message CHSTRR01I (see arrow) is followed by the contents of the CPRB when you use the IOTRACE option. The address under the message ID is the address in a dump where you can locate the same CPRB information. In Figure 10-2, the address of the CPRB is 02825E20.

```
    CHSCMI02I The control unit does not support Read Partitioned Queries.
    CHSTRH01I Sent HRES at 12:38:06 sequence=00000 session=000 length=00010:
    82825E94 0000000E 00000000 2020...  ........ ........ ........ ........ ........

    CHSDCOM01I The following data is about to be sent to VM/PC:
    0004D060 F5C61140 401D6D1D 7C1D7C1D 7D404040 40C36040 40404040 60C84040 401DF1..

    CHSDCOM12I Data received at 12:38:07, length=00000024 RC=00000018:
    0004D830 7D404011 40C11140 C21140C3 1140C440 40404040 E4404040 404060C8 401140D3
    0004D850 401140D5 ........ ........ ........ ........ ........ ........ ........

    CHSTRH02I Received PACK at 12:38:07 sequence=00000 session=000 length=00010:
    8282642E 00000002 40000000 2020.... ........ ........ ........ ........ ........

    CHSTRL01I VM/PC allocate request for CHSMVS  , server ID=0004.
    CHSTRH01I Sent HQNL at 12:38:07 sequence=00001 session=000 length=00040:
    82825E94 00010005 0000001E 001F02FF 04CFD082 8220C3C8 E2D4E5E2 4040E300 010CF0F0
    82825EB4 F0F1F0C3 C80004ED ........ ........ ........ ........ .. ...... ........

--->CHSTRR01I CPRB request at 12:38:18 server=CHSDISK  function=0001:
    02825E20 01000001 C3D7D9C2 00000000 00000000 C3C8E2C4 C9E2D240 00 000001 00000000
    02825E40 00000000 00000000 00000050 02826C58 00000050 02826C58 00 000018 02826C40
    02825E60 00000018 02826C40 00000000 00000000 00000000 00000000 00 000000 00000000
    02825E80 00000000 00000000 00000000 00000000 ........ ........ .. ...... ........

    CHSTRS01I CPRB reply at 12:38:18 RC=00000000 CHSDISK  RC=00000000 .
    CHSTRH01I Sent HRNL at 12:38:18 sequence=00006 session=002 length =00098:
    82825E94 00060209 00000058 20007000 00000000 00000000 00000000 0005BD58 D0BBDA07
    82825EB4 FC000200 01000000 00000200 01010000 00000000 00000000 00000000 00000000
    82825ED4 00000000 18000000 00D4C1D9 E9C9D640 40D9E640 40404040 40F1F9F1 F2F0F040
    82825EF4 40B5.... ........ ........ ........ ........ ........ ........ ........

    CHSDCOM09I User pressed the PF3  key, requesting termination.
    CHSCPS08I MVSSERV is ending.
    CHSTTP01I Internal trace table follows.  Last entry is 127:
    CHSTTP02I 000 TTRT    001 TLMP    002 TIOR    003 TLMP
    CHSTTP02I 004 TIOR    005 PRNL    006 TINF    007 TLG2
    CHSTTP02I 008 TLMP    009 TIOR    010 HRFW    011 TDCOM
    CHSTTP02I 012 TTRH    013 TLMP    014 TIOR    015 TTRT
    CHSTTP02I 016 TLMP    017 TIOR    018 TLMP    019 TIOR
    CHSTTP02I 020 TCH7    021 TLMP    022 TIOR    023 TTRT
    CHSTTP02I 024 TLMP    025 TIOR    026 TLMP    027 TIOR
    CHSTTP02I 028 TLMP    029 TIOR    030 TTRT    031 TLMP
    CHSTTP02I 032 TIOR    033 TLMP    034 TIOR    035 TLMP
    CHSTTP02I 036 TIOR    037 TLMP    038 TIOR    039 TLMP
    CHSTTP02I 040 TIOR    041 TC7H    042 TTRH    043 TLMP
    CHSTTP02I 044 TIOR    045 TTRT    046 TLMP    047 TIOR
        .       .           .           .           .
        .       .           .           .           .
        .       .           .           .           .
    CHSTTP02I 252 PQNL    253 TOCPB   254 TRUT    255 TRUTR
```

Figure 10-2. Sample Trace Data Set Obtained Using MVSSERV IOTRACE Option

Services used by MVSSERV

The MVSSERV command processor uses the following MVS services:

- Parse
- Cell pool services
- Supervisor services:
  - TPUT, TPG, TGET, ENQ, DEQ, ESTAE, ABEND,
  - ATTACH, DETACH, POST, WAIT,
  - GETMAIN, AND FREEMAIN.
- Data management services:
- CATALOG, RENAME, SCRATCH, LOCATE, OBTAIN, BLDL,
- STOW, PUTX, OPEN, CLOSE, GET, PUT, READ, WRITE.

Dumps

Dumps provide useful information for analyzing an MVSSERV error. You may find it useful to have a dump data set and dump suppression data set allocated for your MVSSERV session. If so, allocate them with a record length of 80 and a fixed or fixed blocked format, and the following ddnames:

- Dump data set (ddname = SYSUDUMP, SYSMDUMP, or SYSABEND)
- Dump suppression data set (ddname = CHSABEND).

For example, you can allocate a dump data set and dump suppression data set by entering:

```
ALLOCATE F(sysudump)  da('myID.sysudump') LRECL(80) RECFM(F)
ALLOCATE F(chsabend)  da('myID.chsabend') LRECL(80) RECFM(F)
```

Note: The LRECL for a sysudump is not always 80. Refer to *MVS/Extended Architecture Diagnostic Techniques* for information about which type of dump to use and options you may specify.

MVSSERV may provide a SNAP dump, depending on the scope of the problem. If the dump for a particular ABEND has not been suppressed in the CHSABEND data set, a dump will be provided when a server or MVSSERV module fails. MVSSERV uses the dump suppression data set to determine what diagnostic action is necessary.

MVSSERV provides a dump when:

- An error occurs in MVSSERV and the dump for that ABEND was not suppressed.
- An error occurs in either a server or the initialization/termination program and

  - A dump is not provided in the server's recovery routine.
  - The server has no recovery routine.

The terminal monitor program provides a dump when:

- After issuing the MVSSERV command, you see the following messages and press the ENTER key:

```
IKJ56641I  MVSSERV ended due to an error.
IKJ56640I  abend code ______   reason code ______
```

Using the Dump Suppression Data Set

You use the dump suppression data set to specify which failures may be exempt from additional diagnostics. Each record in the data set contains an ABEND code, Reason code, and a field specifying the action to be taken if the failure matches these codes.

The data set you allocate to CHSABEND must be comprised of 80-byte records in fixed or fixed blocked format as follows:

OFFSET	BYTES	DESCRIPTION
+0	3	EBCDIC ABEND code (hex) (user code is in decimal)
+3	1	Reserved
+4	4	EBCDIC Reason code (hex)
+8	1	Reserved
+9	1	EBCDIC dump action field: 0 = Do not dump 1 = SNAP dump
+10	70	Reserved

For more information about initializing the dump suppression data set, see *TSO Extensions Programmer's Guide to the Server-Requester Programming Interface for MVS/Extended Architecture*.

MVSSERV recovery searches the dump suppression data set to see if there is an entry corresponding to the ABEND and Reason codes for the failure. If there is a match, the recovery manager uses the dump action field to determine what action to take. If there is no match, the recovery manager defaults to providing a SNAP dump. Note that a SNAP dump cannot be issued unless you have pre-defined a SYSUDUMP, SYSMDUMP, or SYSABEND file name.

If the recovery manager receives control with no SDWA available, processing ends and a dump is not issued.

MVSSERV Dump Information

The following is a list of diagnostic information found in a dump, along with the location of each item. Items that are the same for all dumps have the actual dump information next to them instead of a location:

```
Completion code (ABEND code)..............RTM2WA Summary
Abending program name.....................RTM2WA Summary
Abending program address..................RTM2WA Summary
Registers at time of error................RTM2WA Summary
PSW at time of error......................RTM2WA Summary
SDWA address..............................RTM2WA Summary
SDWAVRA address...........................SDWA address + x'194'
```

The **SDWA** contains the following:

```
Name of active load module................MVSSERV
Level of active assembly module (RMID)....JBB2369
Recovery routine assembly module..........CHSTREC
Recovery routine label....................CHSTREC
Component ID..............................28507
Component ID base.........................5665
Name of active assembly module............SDWA address + x'12C'
```

The **SDWAVRA** contains the following:

```
Address of CHSDCOM........................SDWAVRA address + x'E'
Address of CHSDCITT of failing task.......SDWAVRA address + x'1E'
Address of Execution Path Trace Table.....SDWAVRA address + x'36'
Offset into Execution Path Trace Table....SDWAVRA address + x'50'
State of failing task at ABEND............SDWAVRA address + x'56'
Registers at the time of failure..........end of State of failing
                                          task field + x'02'
Address of failing module's
  footprint area, if available............variable

MAIN TASK FAILURE
(if failure occurs in main task)..........variable
  or
Name of initialization/termination
  program for failing task
(if failure occurs in server task)........variable
```

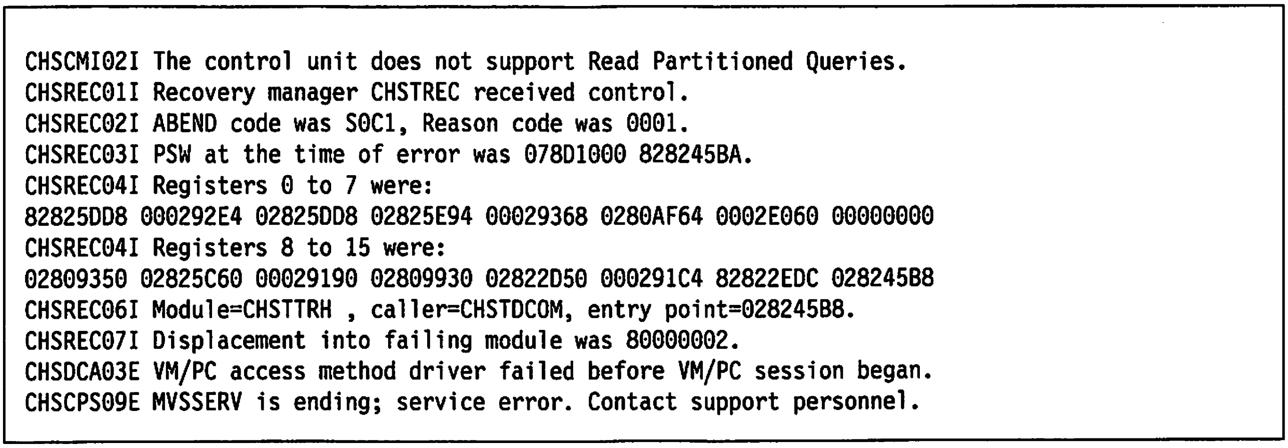
For information about finding and interpreting SDWA and SDWAVRA data in a dump, see *TSO Extensions System Diagnosis: Guide and Index.*

TSO Extensions System Diagnosis: Guide and Index also provides information about reading messages, issuing traces and traps and calling IBM to report a problem you are unable to fix.

Trace Data Set Contents

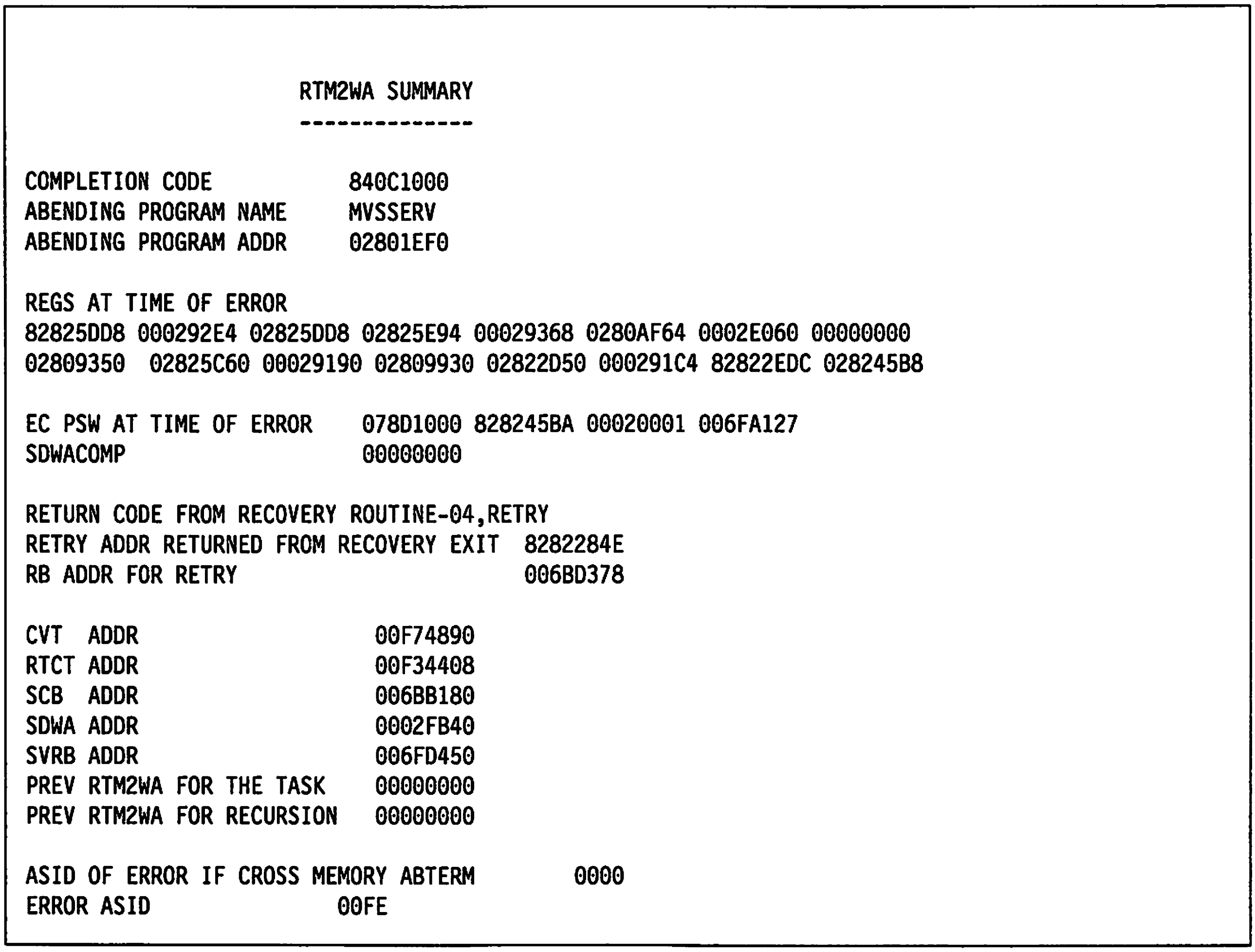
MVSSERV writes messages containing RTM2WA Summary information to the external trace data set during processing. Additionally, the external trace data set displays the contents of the Request CPRB in message CHSTRR01I when you use the IOTRACE option of the MVSSERV command. The address under the message ID is the address in a dump where you can locate the same CPRB information. This information is primarily useful for debugging a server error. (The CPRB is not initialized if a server is not invoked.)

Following are examples of corresponding diagnostic information found in the messages section of a trace data set and the RTM2WA Summary in a dump.



```
CHSCMI02I The control unit does not support Read Partitioned Queries.
CHSREC01I Recovery manager CHSTREC received control.
CHSREC02I ABEND code was S0C1, Reason code was 0001.
CHSREC03I PSW at the time of error was 078D1000 828245BA.
CHSREC04I Registers 0 to 7 were:
82825DD8 000292E4 02825DD8 02825E94 00029368 0280AF64 0002E060 00000000
CHSREC04I Registers 8 to 15 were:
02809350 02825C60 00029190 02809930 02822D50 000291C4 82822EDC 028245B8
CHSREC06I Module=CHSTTRH , caller=CHSTDCOM, entry point=028245B8.
CHSREC07I Displacement into failing module was 80000002.
CHSDCA03E VM/PC access method driver failed before VM/PC session began.
CHSCPS09E MVSSERV is ending; service error. Contact support personnel.
```

Figure 10-3. Diagnostic Messages in a Trace Data Set



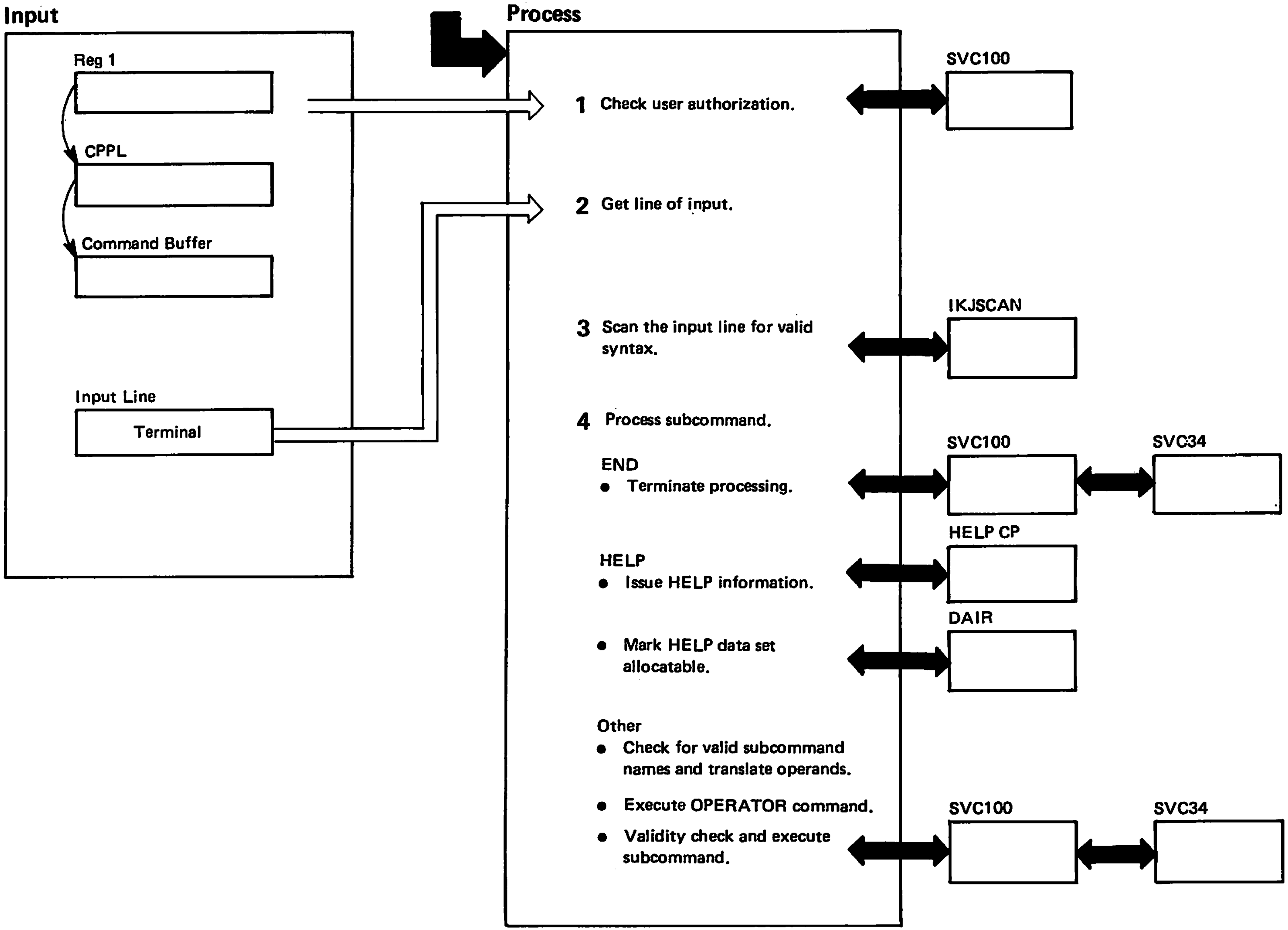
```
                    RTM2WA SUMMARY
                    --------------

COMPLETION CODE          840C1000
ABENDING PROGRAM NAME    MVSSERV
ABENDING PROGRAM ADDR    02801EF0

REGS AT TIME OF ERROR
82825DD8 000292E4 02825DD8 02825E94 00029368 0280AF64 0002E060 00000000
02809350  02825C60 00029190 02809930 02822D50 000291C4 82822EDC 028245B8

EC PSW AT TIME OF ERROR    078D1000 828245BA 00020001 006FA127
SDWACOMP                   00000000

RETURN CODE FROM RECOVERY ROUTINE-04,RETRY
RETRY ADDR RETURNED FROM RECOVERY EXIT  8282284E
RB ADDR FOR RETRY                       006BD378

CVT  ADDR                   00F74890
RTCT ADDR                   00F34408
SCB  ADDR                   006BB180
SDWA ADDR                   0002FB40
SVRB ADDR                   006FD450
PREV RTM2WA FOR THE TASK    00000000
PREV RTM2WA FOR RECURSION   00000000

ASID OF ERROR IF CROSS MEMORY ABTERM       0000
ERROR ASID               00FE
```

Figure 10-4. Diagnostic Information in a Dump

Chapter 11. OPERATOR Command Processing

This section describes the logic of the **OPERATOR** command. It emphasizes the flow of data and control information through buffers and tables, and contains detailed functional descriptions through the use of method of operations diagrams.

Diagram 11.1. OPERATOR Command Processing (Part 1 of 2)



1 Check user authorization.

2 Get line of input.

3 Scan the input line for valid syntax.

4 Process subcommand.

END
- Terminate processing.

HELP
- Issue HELP information.
- Mark HELP data set allocatable.

Other
- Check for valid subcommand names and translate operands.
- Execute OPERATOR command.
- Validity check and execute subcommand.

Diagram 11.1. OPERATOR Command Processing (Part 2 of 2)

Extended Description **Module** **Label**

1 After the ESTAE and ATTEN exits are set up, SVC100 checks the user's authority to enter the OPERATOR command. Information is passed to SVC100 in the FIBPARMS parameter list. If the user is not authorized, OPERATOR will issue an error message and return control to the TMP.

2 Use PUTGET to get a line of input from the terminal and to issue the command mode message if required.

3 Scan the input line with IKJSCAN for valid syntax. If the subcommand syntax was invalid an error message is issued and PUTGET gets another line of input.

4 Process OPERATOR subcommands.

END

- This routine is used to terminate processing due to an error or when an END subcommand is issued by the terminal user to terminate OPERATOR command processing. SVC100 is used to stop active monitors and to issue SVC34 to schedule executions of the subcommand. All buffers are freed and service routines are deleted. Control is returned to the TMP.

HELP

- ATTACH the HELP command processor to send the terminal user the HELP information. If the ATTACH failed control passes to step 4-END.
- When HELP is finished, use DAIR to mark data sets used by HELP as available for allocation. If DAIR fails, control passes to step 4-END.
- Processing continues with step 2.

Other

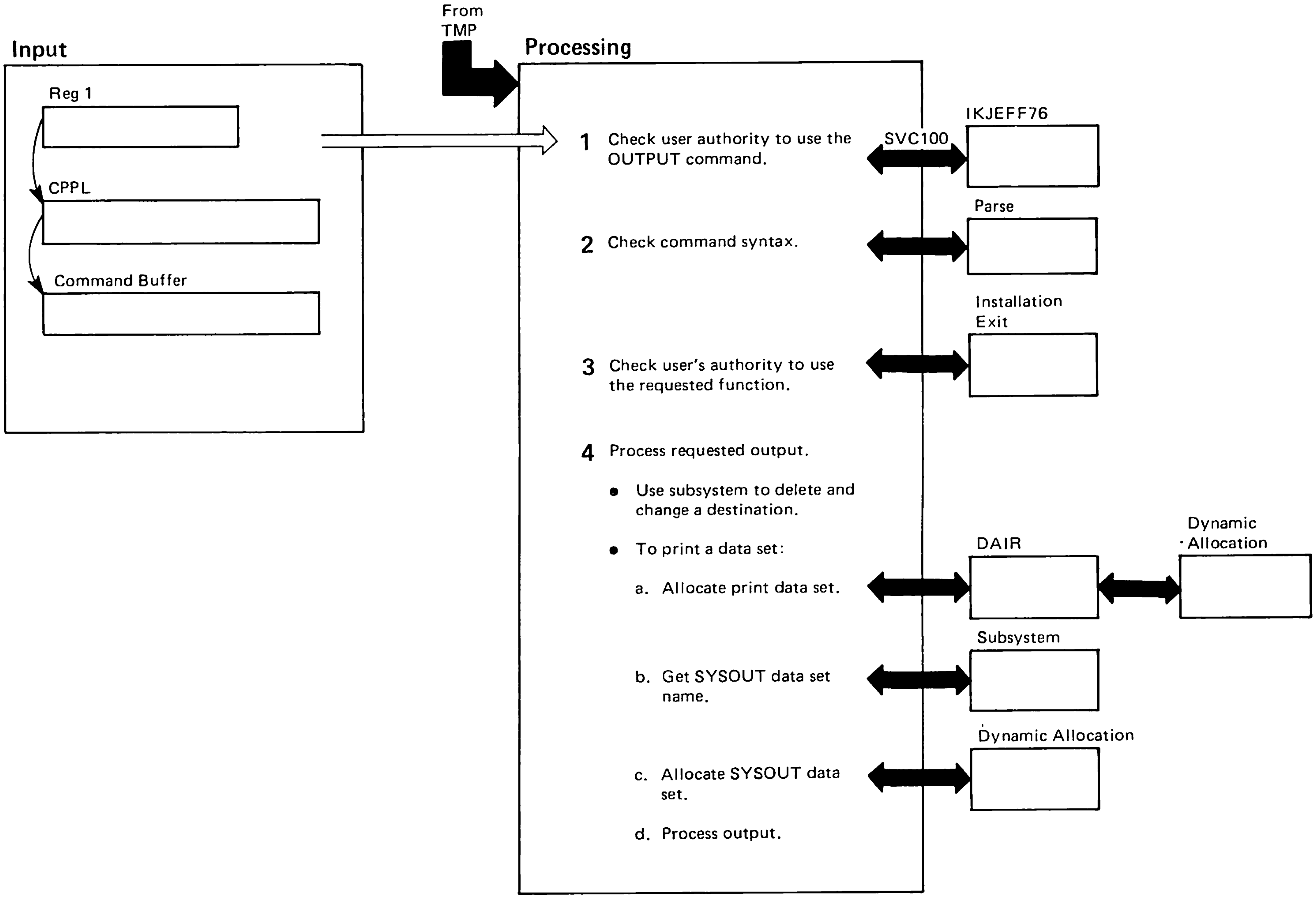
- Check the subcommand name against a list of allowable names (DISPLAY, MONITOR, SEND, CANCEL, STOPMN and SLIP). If parameters were specified on the DISPLAY, MONITOR, CANCEL, STOPMN or SLIP subcommands, translate the operands to upper case for use by SVC100.
- Initialize the FIBPARMS parameter list and use SVC100 to validity check the storage. If the storage is acceptable, SVC100 invokes SVC34. SVC34 executes the OPERATOR command and sends it, along with a timestamp, to hardcopy log. If the validity check fails, an error message is issued to the terminal user. Processing continues with step 3. If there is an error other than validity, processing continues with step 4-END.

Object Modules: IKJEE100, IKJEE1A0, and IKJEE150

Chapter 12. OUTPUT Command Processing

This section describes the logic of the **OUTPUT** command. It emphasizes the flow of data and control information through buffers and tables, and contains detailed functional descriptions through the use of method of operations diagrams.

Diagram 12.1. OUTPUT Processing (Part 1 of 2)



1 Check user authority to use the OUTPUT command.

2 Check command syntax.

3 Check user's authority to use the requested function.

4 Process requested output.

- Use subsystem to delete and change a destination.
- To print a data set:

  a. Allocate print data set.

  b. Get SYSOUT data set name.

  c. Allocate SYSOUT data set.

  d. Process output.

Diagram 12.1. OUTPUT Processing (Part 2 of 2)

Extended Description **Module** **Label**

1 Use SVC100 to post the TMP (IKJEFTSC) requesting IKJEFF76 be attached under a parallel task structure. Information is passed to IKJEFF76 in the FIBPARMS parameter list. IKJEFF76 checks the user's authorization to enter the OUTPUT command. If the user is not authorized to enter foreground initiated background commands, the system issues an error message and returns control to the TMP.

Object Module: IKJCT466

2 Use parse to check the syntax of the command.

Object Module: IKJCT469

3 Use an installation exit to check the user ID for authorization to use the requested function on the job specified. If there is no installation exit the IBM supplied exit IKJEFF53 is used.

Object Module: IKJCT469

4 Determine the operation to be performed: print, delete, or change the destination (station or class) of a data set.

- To delete or change the destination of a data set, set up an interface to the subsystem and request the subsystem to perform the requested operation. Return control to the TMP.

  Object Modules: IKJCT469, IKJCT462

- To print a data set:

  a. Use dynamic allocation to allocate a PRINT data set via the DAIR interface.

  Object Modules: IKJCT469, IKJCT473

  b. Use the job entry subsystem to select all system output data sets for a specific jobname and class.

  Object Module: IKJCT462

  c. Use dynamic allocation to allocate a system output data set by data set name.

  Object Module: IKJCT462

  d. Process the system output data set until an end-of-file condition or an attention.

  For an end-of-file condition, check for more data sets. If there are not more, return control to the TMP.

  For an attention, process the requested subcommand and all remaining data sets and return control to the TMP.

  Object Modules: IKJCT462, IKJCT470, IKJCT471, IKJCT463

Chapter 13. PRINTDS Command Processing

The PRINTDS command is used to print sequential data sets, members of a partitioned data set (PDS), or entire PDSs. The output can be directed to a system printer managed by the Job Entry Subsystem (JES), a JES remote destination (either a printer or another user ID), or a data set.

Overview

The PRINTDS command supports:

- Sequential data sets, members of a partitioned data set, or entire PDSs.
- Output descriptors.
- Formatting of data.
- Information stored in data set by the document composition facility (DCF).

OUTPUT statements in a user's LOGON PROC can define various output characteristics and printer locations by associating them with a common name. This common name can then be specified on the OUTDES operand of the PRINTDS command. Thus, a user does not need to know the actual JES name of a printer or all of the operands needed to use it, only the common name (output descriptor) that references that printer.

PRINTDS has three classes of parameters. Some parameters, like PAGELEN and CLASS, have a fixed default value. Others, like DCF/NODCF or TRC/NOTRC are set dynamically, based on other operands that were specified or based on data set attributes. Still other parameters, such as FLASH or CHARS, have no default values. For a description of PRINTDS parameters and defaults, see *TSO Extensions Customization.*

Diagnosing a PRINTDS Problem

This section describes diagnostic information provided to solve a problem with the PRINTDS command processor. Before going further you may wish to create a search argument and use it to search a problem data base to see if there is already a fix for the problem. For information about searching a problem data base, see *TSO Extensions System Diagnosis: Guide and Index.*

PRINTDS Messages

You can obtain online help for PRINTDS terminal messages by using the message ID with the HELP command after PRINTDS has ended:

```
HELP PRINTDS MSG(message ID)
```

For an explanation for an individual PRINTDS message, look up the message ID in *TSO Messages.*

PRINTDS Return Codes

PRINTDS sets one of the following decimal return codes in register 15 during processing:

- 0 - The data was successfully processed.
- 4 - Processing completed but a warning message has been issued.
- 8 - The input, output, or SYSOUT data set could not be used.
- 12 - An error occurred during the processing of the PRINTDS command.
- 16 - The installation exit requested termination of the PRINTDS command.

PRINTDS Abend Processing

PRINTDS sets the following CLIST variables to zero, unless an abend occurs during processing:

- &SYSPABNCD - The abend code.
- &SYSPABNRC - The abend reason code.

If the PRINTDS command abnormally terminates, RTM issues a dump and the user receives a message indicating that an error occurred. The PRINTDS command recovery routine frees any input and SYSOUT data sets and returns the environment to the state it was in prior to the issuing of the PRINTDS command.

If an error occurs while writing to a TODATASET, the data that has been written to the data set up to the time of the error remains in the data set. This is often useful in determining the cause of the error or the exact point of failure.

Services Used by PRINTDS

The PRINTDS command processor uses the following services:

- TSO services:
  - DAIRFAIL Analysis (IKJEFF18) - to analyze a dynamic allocation error.
  - IKJPARS - TSO parse service.
  - IKJEFF02 - Message issuing service.
  - CLIST variable access routine (IKJCT441) - to set CLIST variables.
- Data management services:
  - FIND, OPEN, CLOSE, PUT, GET, READ, CHECK
  - SYSNADF, SYSNADRLS, LOCATE, OBTAIN.
- System macros:
  - GETMAIN, FREEMAIN, TIME, LINK
  - LOAD, DELETE, ESTAE, SETRP, SNAP
  - SVC 99 - dynamic allocation.

PRINTDS Dump Information

PRINTDS fills in the following information in the system diagnostic work area (SDWA) when it issues a dump:

- SDWACSCT -- active CSECT when the error occurred
- SDWAMLVL -- module level
- SDWAMODN -- load module name (IKJEFY50 or IKJEFY57)

- SDWAREXN -- recovery routine
- SDWARRL -- recovery routine label
- SDWACID -- component ID
- SDWACIDB -- component ID base
- SDWASC -- name of subcomponent (PRINTDS).

PRINTDS fills in the following information in the SDWA variable recording area (SDWAVRA):

- A header indicating that the PRINTDS command processor abended.
- IKJPRCB control block information, including the control block name, address of the PRCB, PRCB version number and length.
- Key VRADAE, to prevent duplicate dumps.
- Command buffer name, address, and length.

For information about finding the SDWA and SDWAVRA in a dump, see *TSO Extensions System Diagnosis: Guide and Index.*

TSO Extensions System Diagnosis: Guide and Index also provides information about reading messages, issuing traces and traps and calling IBM to report a problem you are unable to fix.

Exit Considerations

An optional PRINTDS initialization exit is invoked prior to the PRINTDS command's call to the parse routine to allow an installation to change those operands that have fixed default values. The TITLE/NOTITLE default value may also be changed, but this default may be ignored if it conflicts with the data set attributes. Operands that do not have defaults are ignored by PRINTDS when not specified by the user.

For a description of the PRINTDS default values and the standard format exits provided for the PRINTDS command, see *TSO Extensions Customization.*

Exit Recovery

TSO provides a common recovery routine for the standard format exits. When an abend occurs while attempting to invoke an exit, or during exit processing, the recovery routine furnishes the following information in the SDWA:

- Load module name (IKJRTR01)
- Active module name (IKJRTR01 or the active exit name)
- Level of the active module
- Recovery module name (RTRESTAE)
- Recovery label name (RTRESTAE)
- Component ID (28502)
- Component ID base (5665).

The common recovery routine furnishes the following information in the SDWAVRA when issuing a dump:

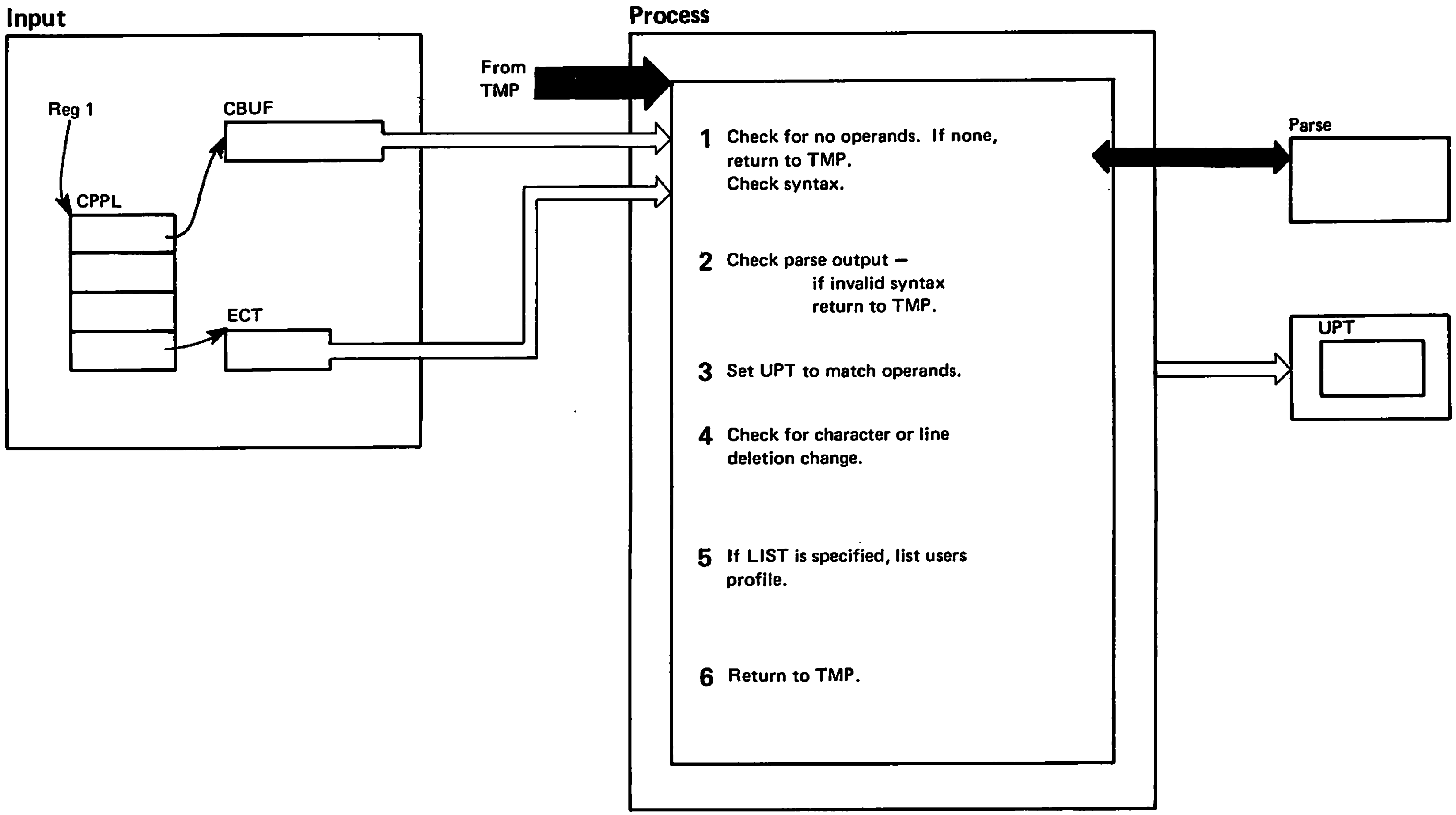
- Name of the exit handler parmlist, followed by its address.
- Name of the abending exit.

For unauthorized exits, a SNAP dump is issued; an SVC dump is issued for authorized exits.

Chapter 14. PROFILE Command Processing

This section describes the logic of the **PROFILE** command. It emphasizes the flow of data and control information through buffers and tables, and contains detailed functional descriptions through the use of method of operations diagrams.

Diagram 14.1. PROFILE Processing (Part 1 of 2)



1 Check for no operands. If none, return to TMP. Check syntax.

2 Check parse output – if invalid syntax return to TMP.

3 Set UPT to match operands.

4 Check for character or line deletion change.

5 If LIST is specified, list users profile.

6 Return to TMP.

Diagram 14.1. PROFILE Processing (Part 2 of 2)

Extended Description **Module** **Label**

1 Check the ECT for the presence of operands in CBUF (the command buffer). If there are none, list the users profile.

Invoke parse to check the syntax of the operands.

2 Check the parse return code.

non-zero - means an operand was not valid and prompting failed. Issue an error message unless the return code indicates the user was in noprompt mode. Return to the caller in any case.

zero - means parse was successful.

3 Set the UPT (the user profile table) to conform to the user options, checked by parse.

4 If a new line or character deletion character was among the operands, issue a STCC macro to change the terminal line or character deletion characters. Check the return code.

- non-zero - means reissue a STCC macro with the former line or character delete characters, and issue an error message.
- zero - means issue SVC100 to update the PSCB with the new line-delete and character-delete change requests.

5 If the operand LIST has been specified, list the users profile.

6 Free storage, set the return code, and return to the TMP.

zero - means successful processing.

non-zero - means unsuccessful processing.

Object Module: IKJEFT82

Chapter 15. PROTECT Command Processing

This section describes the logic of the **PROTECT** command. It emphasizes the flow of data and control information through buffers and tables, and contains detailed functional descriptions through the use of method of operations diagrams.

Diagram 15.1. PROTECT Command Processing (Part 1 of 2)

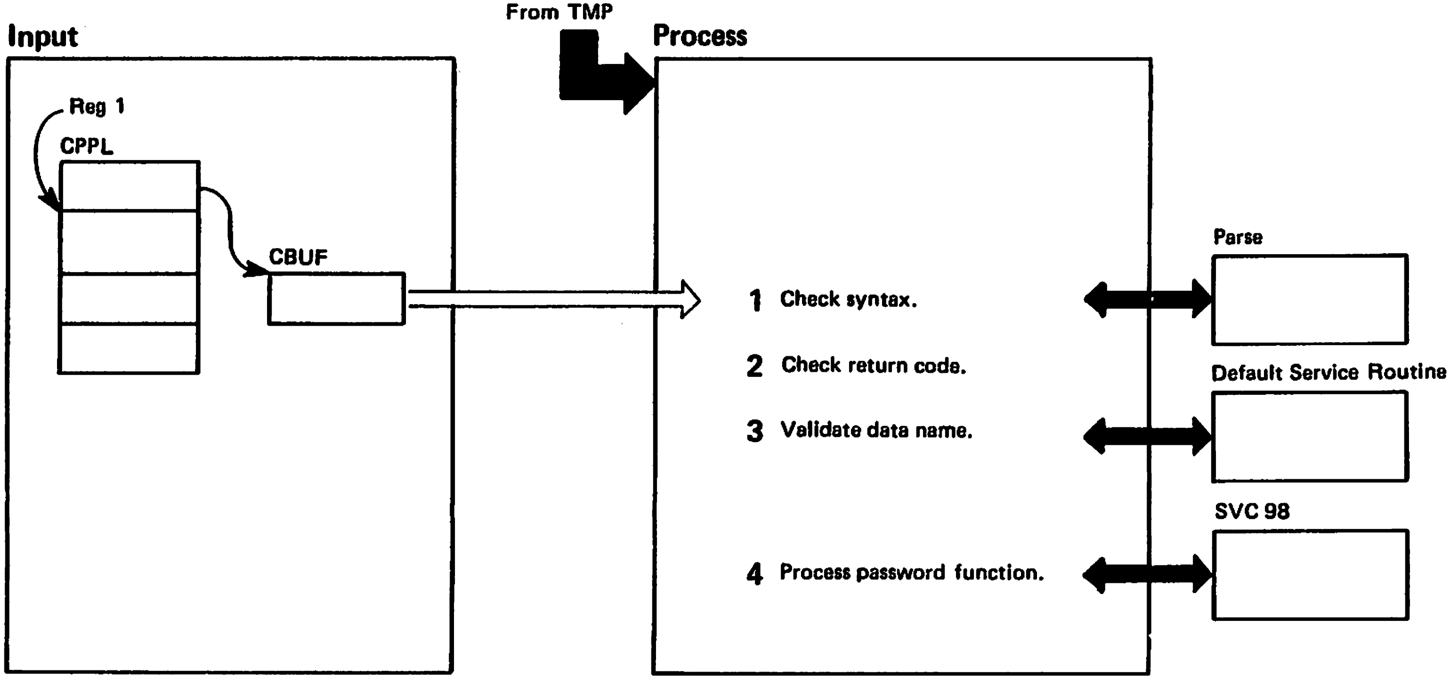


Diagram 15.1. PROTECT Command Processing (Part 2 of 2)

Extended Description **Module** **Label**

1 Use parse to scan and check the command for proper syntax.

Possible messages: IKJ58102I, IKJ58112I

2 After check the parse return code, move the control password, if one was specified, to the SVC 98 buffer.

Possible message: IKJ58108I

3 If the data set name was not fully qualified, it is fully qualified using IKJEHDEF (the default service routine).

Possible messages: IKJ58103I, IKJ58111I, IKJ58112I

4 Check the function to be performed, and fill in the parameter list (SVCPARMS) for SVC 98 accordingly. The first byte of the parameter list contains a hexadecimal value indicating the function, as follows.

X'01' ADD an entry to the password data set.

X'02' REPLACE an entry in the password data set.

X'03' DELETE an entry from the password set.

X'04' LIST protection, security counter, and optional data information of a protected data set. (The last 80 bytes of the password data set entry for this data set password is placed in the 80 byte buffer pointed to by the SVC parameter list.)

Issue SVC 98 return control to the TMP issuing error messages, depending on the return code provided by SVC 98.

Possible messages: IKJ58101I, IKJ58104I, IKJ58105I, IKJ58106I, IKJ58107I, IKJ58110I, IKJ58112I

Object Module: IKJEHPRO

Chapter 16. RACONVRT Command Processing

This chapter describes the operation of the RACONVRT command processor and its diagnostic aids.

Overview

The RACONVRT command is an aid in converting information from the SYS1.UADS data set to the RACF data base. RACONVRT must be invoked authorized and you must have "RACF Special" authority to issue the command. When RACONVRT is invoked, it creates a CLIST data set containing the necessary commands to convert the TSO user information in the SYS1.UADS data set to a TSO segment within the RACF data base and to RACF resources. The RACONVRT utility:

- Reads SYS1.UADS information and gathers the data pertinent to each user ID.
- Creates RACF commands that create or update the RACF user profile to contain a TSO segment.
- Defines resources to RACF and allows the user to access those resources.

RACONVRT performs the following operations:

- Obtains the necessary conversion information from SYS1.UADS.
- Creates the RACF commands that will perform the conversion.

A CLIST data set, 'user-prefix.IKJ.RACONVRT.CLIST', is created by the RACONVRT command. The members of this data set contain the commands created during execution of the command. After execution, these members can be edited by the installation to customize the conversion.

The conversion is completed by executing the RUN member of the CLIST data set. This member contains commands to execute all of the data set members created by the latest RACONVRT command in the proper order to allow a successful conversion. The members that are created are:

- **ADDUSER** - contains commands necessary to define a user to RACF and to define a TSO segment within the newly-created profile.

- **ALTUSER** - contains commands necessary to define the TSO segment within existing RACF user profiles.

- **DEFAUTH** - contains commands necessary to define each of the TSO authorities (OPERATOR, ACCOUNT, JCL, MOUNT, RECOVER) as RACF resources and the commands necessary to permit users to access each of the TSO authorities.

- **DEFPROC** - contains commands necessary to define the TSO LOGON procedures as RACF resources and the commands necessary to permit users to access the procedures.

- **DEFACCT** - contains commands necessary to define the TSO LOGON account numbers as RACF resources and permit users to access the account numbers.
- **DEFPERF** - contains commands necessary to define the TSO LOGON performance groups as RACF resources and the commands necessary to permit users to access the performance groups.
- **RUN** - contains commands necessary to invoke the other members, created by the latest RACONVRT command, in the proper order to complete the conversion to the RACF data base successfully.

Diagnosing a RACONVRT Problem

This section describes diagnostic information provided to solve a problem with the RACONVRT command processor. Before going further you may wish to create a search argument and use it to search a problem data base to see if there is already a fix for the problem. For information about searching a problem data base, see "Creating a Search Argument" in *TSO Extensions System Diagnosis: Guide and Index.*

RACONVRT ABEND Codes

The RACONVRT command issues a system x'01B' abend code. The following reason codes are associated with the abend code:

- 4 - an unexpected return code was received from the RACROUTE macro.
- 8 - an invalid data management request was passed to the RACONVRT I/O routine.

RACONVRT does not request a dump for the following system abend codes:

- '913'
- '13E'
- 'x22'
- 'x37'.

RACONVRT Return Codes

Following is a description of the return codes and reason codes that result from RACONVRT processing. The reason codes are associated with return code 64.

```
EXIT NORMAL:

Register 15 contains one of the following decimal return codes:

0 - Processing completed successfully.

4 - Processing completed unsuccessfully, the recovery
    environment could not be established.

8 - Processing completed unsuccessfully, the command was
    not invoked in an authorized state.

12 - Processing completed unsuccessfully, the invoker has
     insufficient authority to issue the command.

16 - Processing completed unsuccessfully, the command
     operand could not be parsed.

20 - Processing completed unsuccessfully, IKJRUR04 and
     IKJEFA51 could not be loaded.

24 - Processing completed unsuccessfully, storage for the
     necessary tables could not be obtained.

28 - Processing completed unsuccessfully, the SYS1.UADS
     data set could not be opened.

32 - Processing completed unsuccessfully, an I/O error
     occurred while reading the SYS1.UADS data set.

36 - Processing completed unsuccessfully, RACF is not active.

40 - Processing completed unsuccessfully, an error occurred
     during RACF processing.

44 - Processing completed unsuccessfully, PUTGET failed when
     prompting the user with message IKJ56781A.

48 - Processing completed unsuccessfully, the user has
     terminated RACONVRT.

52 - Processing completed unsuccessfully, the CLIST data set
     is allocated with invalid attributes.

56 - Processing completed unsuccessfully, the CLIST data set
     could not be allocated.

60 - Processing completed unsuccessfully, the CLIST data set
     could not be opened.

64 - Processing completed unsuccessfully, an I/O error occurred
     writing to the CLIST data set.
```

RACONVRT Reason Codes

The following decimal reason codes are associated with return code 64:

4 - WRITE to the ADDUSER member failed.

8 - STOW to create the ADDUSER member failed

12 - WRITE to the ALTUSER member failed.

16 - STOW to create the ALTUSER member failed.

20 - WRITE to the DEFAUTH member failed.

24 - STOW to create the DEFAUTH member failed.

28 - WRITE to the DEFPROC member failed.

32 - STOW to create the DEFPROC member failed.

36 - WRITE to the DEFACCT member failed.

40 - STOW to create the DEFACCT member failed.

44 - WRITE to the DEFPERF member failed.

48 - STOW to create the DEFPERF member failed.

52 - WRITE to the RUN member failed.

56 - STOW to create the RUN member failed.

Services Used by RACONVRT

The RACONVRT command uses the following services during processing:

- MVS system services:
  - ESTAE, GETMAIN, FREEMAIN, ABEND
  - TESTAUTH, LOAD, DELETE, ENQ, DEQ
  - SETRP, VRADATA, SDUMP, DYNALLOC (SVC 99).
- TSO/E services:
  - Parse, PUTLINE, PUTGET.
- Data management services:
  - OPEN, READ, WRITE, CHECK
  - STOW, CAMLST, CLOSE, LOCATE, OBTAIN.
- RACF services:
  - RACXTRT.

RACONVRT Dump Information

RACONVRT issues an ESTAE macro to invoke a recovery routine. If an abend occurs, control passes to the recovery routine, which furnishes the following diagnostic information in the system diagnostic work area (SDWA):

- Load module name (RACONVRT)
- Active module name
- Level of the active module
- Recovery module name (IKJRUR03)
- Recovery label name (IKJRUR03)
- Component ID (28502)
- Component ID base (5665)
- A functional description of the RACONVRT utility.

The recovery routine furnishes the following information in the SDWA variable recording area (SDWAVRA):

- Name of the abending module
- Name and address of the common work area (IKJRUCWA)
- Name and address of the recovery work area (IKJRURWA)
- Name and address of the I/O work area (IKJRUIO).

For information about finding the SDWA and SDWAVRA in a dump, see *TSO Extensions System Diagnosis: Guide and Index.*

TSO Extensions System Diagnosis: Guide and Index also provides information about reading messages, issuing traces and traps and calling IBM to report a problem you are unable to fix.

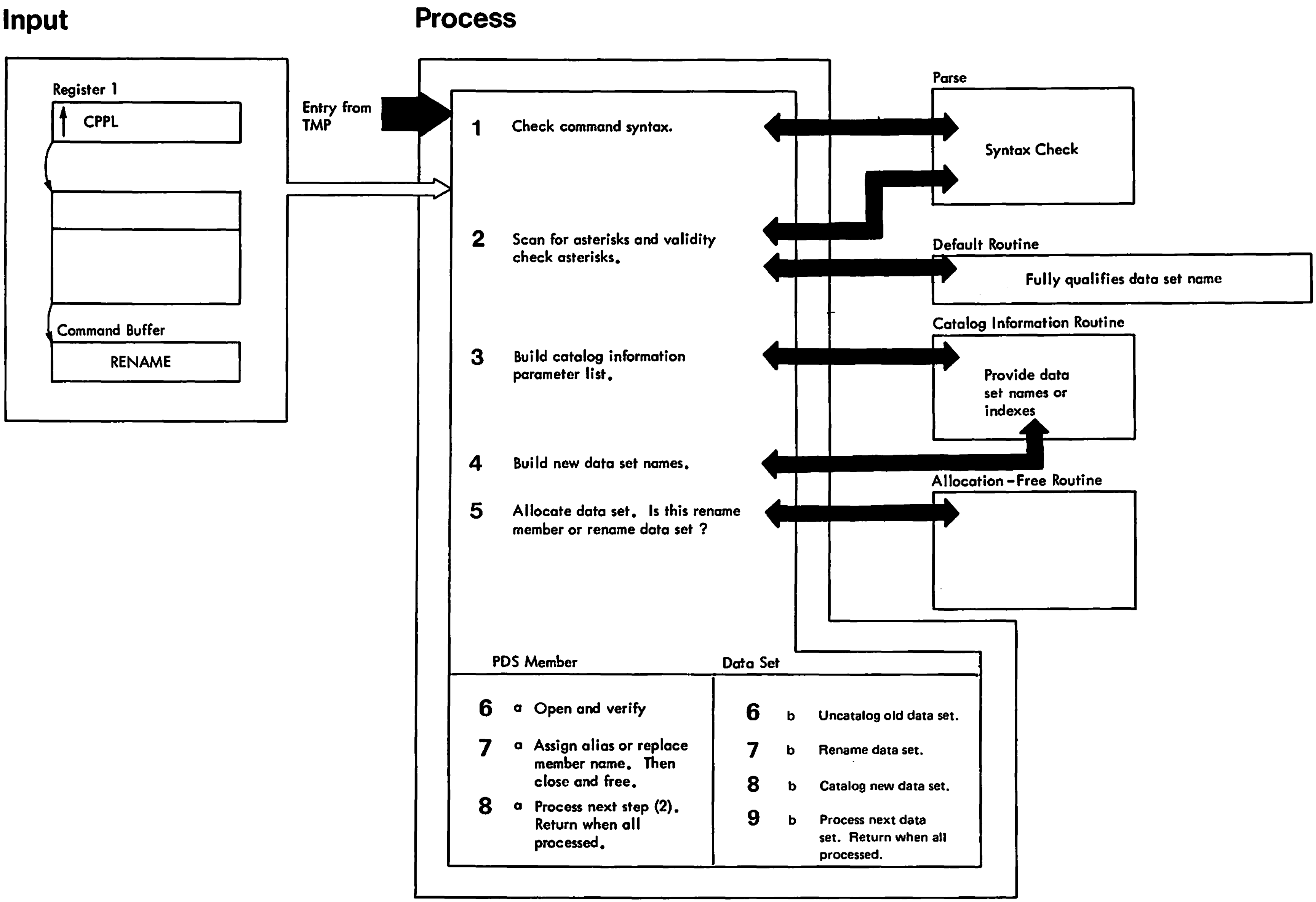
Chapter 17. RECEIVE Command Processing

For a complete description of diagnosing a problem with the RECEIVE command, see the TRANSMIT command processor description in *TSO Extensions System Diagnosis: Command Processors, T-Z.*

Chapter 18. RENAME Command Processing

This section describes the logic of the **RENAME** command. It emphasizes the flow of data and control information through buffers and tables, and contains detailed functional descriptions through the use of method of operations diagrams.

Diagram 18.1. RENAME Command Processing (Part 1 of 2)



1 Check command syntax.

2 Scan for asterisks and validity check asterisks.

3 Build catalog information parameter list.

4 Build new data set names.

5 Allocate data set. Is this rename member or rename data set ?

PDS Member

6 a Open and verify

7 a Assign alias or replace member name. Then close and free.

8 a Process next step (2). Return when all processed.

Data Set

6 b Uncatalog old data set.

7 b Rename data set.

8 b Catalog new data set.

9 b Process next data set. Return when all processed.

Diagram 18.1. RENAME Command Processing (Part 2 of 2)

Extended Description **Module** **Label**

1 The parse subroutine syntax checks the command.

Possible messages: IKJ58202I, IKJ58223I

Storage is obtained for work areas.

2 A user ID is prefixed if necessary. The data set name is scanned for asterisks. If none, prompting is done for any necessary qualification of data set names by the default routine. Then operation continues from step 4. If asterisks are found, they are checked to ensure that they occur in the same relative position within the fully qualified data set names.

Possible messages: IKJ58206I, IKJ58208I, IKJ58209I, IKJ58218I, IKJ58225I, IKJ58227I,

3 If asterisks were found, the catalog information routine is used to look up candidates for renaming.

Possible messages: IKJ58201I, IKJ58219I

4 The new data set names are built in preparation for the renaming operation.

Possible messages: IKJ58205I, IKJ58208I

5 Allocation is done to make use of the system enqueueing facility which ensures that the data set is not renamed while some other user is using it. (Also, this enables the OPEN and CLOSE operation for partitioned members.)

Possible messages: IKJ58201I, IKJ58202I, IKJ58211I, IKJ58212I, IKJ58213I, IKJ58214I, IKJ58215I, IKJ58227I, IKJ58225I, IKJ58229I

6a OPEN and BLDL are used to open the PDS.

Possible messages: IKJ58203I, IKJ58204I, IKJ58207I, IKJ58217I

7a STOW is used to assign the alias or new member name.

Possible messages: IKJ58207I, IKJ58217I, IKJ58223I, IKJ58226I

8a The data set is closed and unallocated.

Possible messages: IKJ58201I, IKJ58207I, IKJ58216I, IKJ58222I, IKJ58224I

6b CATALOG is used to uncatalog an old data set.

Possible messages: IKJ58210I

7b RENAME is used to rename the data set.

Possible messages: IKJ58207I, IKJ58230I

8b CATALOG is used to catalog a new data set.

Possible messages: IKJ58227I, IKJ58228I, IKJ58230I

9b Repeat from step 4, if applicable.

Object Module: IKJEHREN

Chapter 19. RUN Command Processing

This section describes the logic of the **RUN** command. It emphasizes the flow of data and control information through buffers and tables, and contains detailed functional descriptions through the use of method of operations diagrams. Figure 19-1 is the visual table of contents for the RUN command.

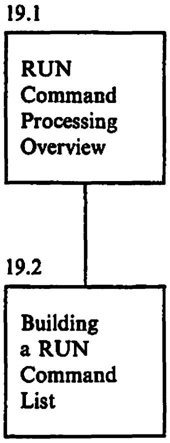
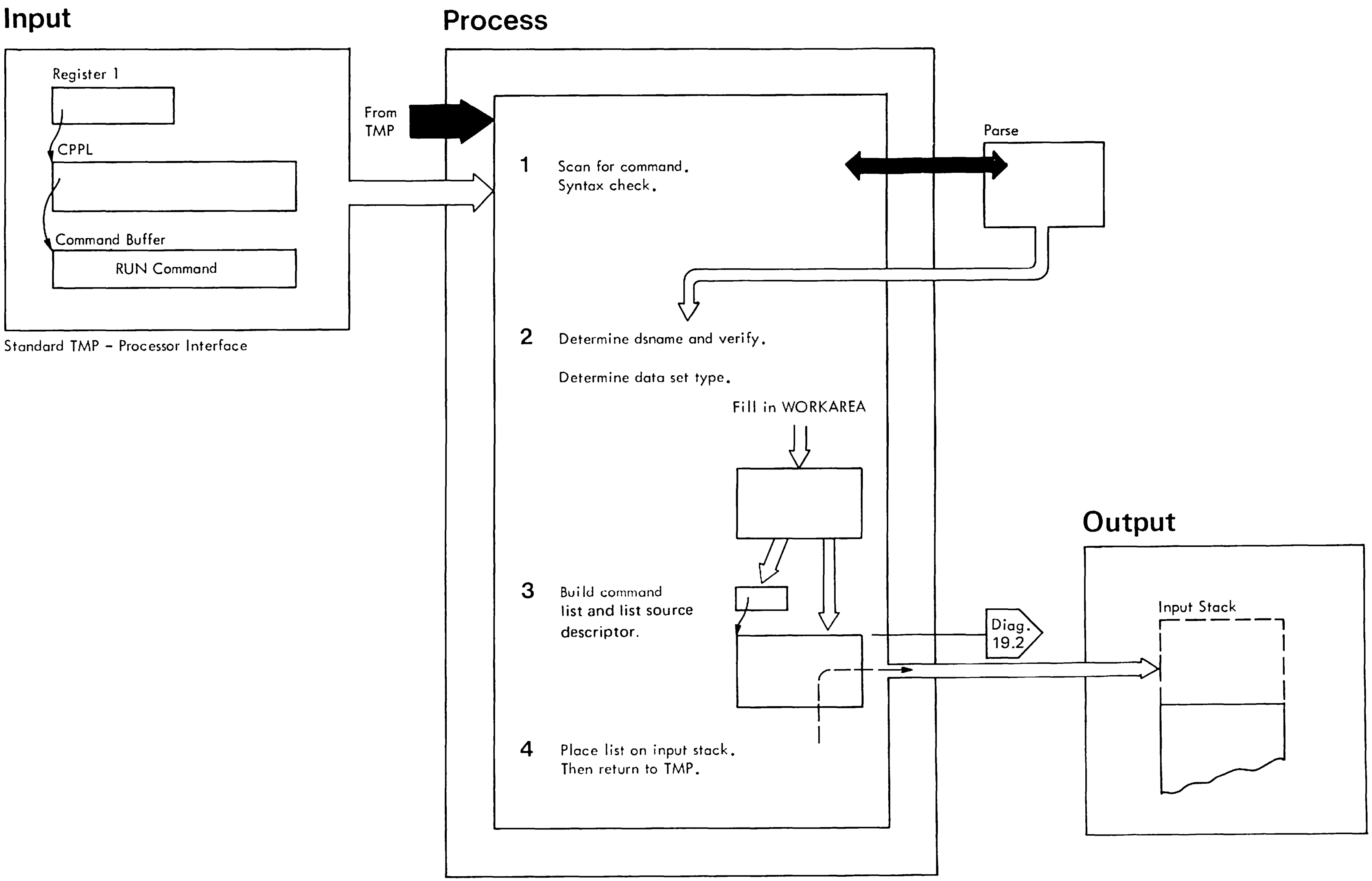


Figure 19-1. RUN Command Processing Visual Table of Contents

Diagram 19.1. RUN Command Prncessing Overview (Part 1 of 2)



1 Scan for command. Syntax check.

2 Determine dsname and verify.

Determine data set type.

3 Build command list and list source descriptor.

4 Place list on input stack. Then return to TMP.

Diagram 19.1. RUN Command Processing Overview (Part 2 of 2)

Extended Description **Module** **Label**

1 IKJEFR00 uses parse to scan and syntax check the RUN command. Prompting occurs if required parameters are missing or if syntactically incorrect parameters are present.

2 Upon return from parse, the return code is checked. If an error was encountered, a message is issued to the user; otherwise, processing continues.

Control passes to a routine that examines the specified data set or member name and places applicable information into a buffer in WORKAREA. If the data set is fully qualified, an indicator is set. If a password is specified, the password and length are placed in WORKAREA.

Then the data set type (ASM, etc) is determined and placed in the data set type buffer of WORKAREA. Parse is again used, if necessary, to prompt for the data set type.

DAIR is then given control to search for a data set having the specified name. First the set of currently allocated data sets is searched; then if necessary, the system catalog. If the data set is found, processing continues; otherwise, the user is prompted for a respecification, and another search is made.

3 When a data set is verified as existing, storage is obtained in shared subpool 78 for an in-storage command list and a table (the list source descriptor) describing the list. See Diagram 19.2 for details of this operation.

4 After the in-storage command list and list source descriptor are built, the address of the list source descriptor is placed in the STACK parameter list and control is passed to stack. This routine places the command list on the input stack.

Then control returns to the TMP. The TMP will select the next command from the top of the input stack.

Object Module: IKJEFR00

Diagram 19.2. Building a RUN Command List (Part 1 of 2)

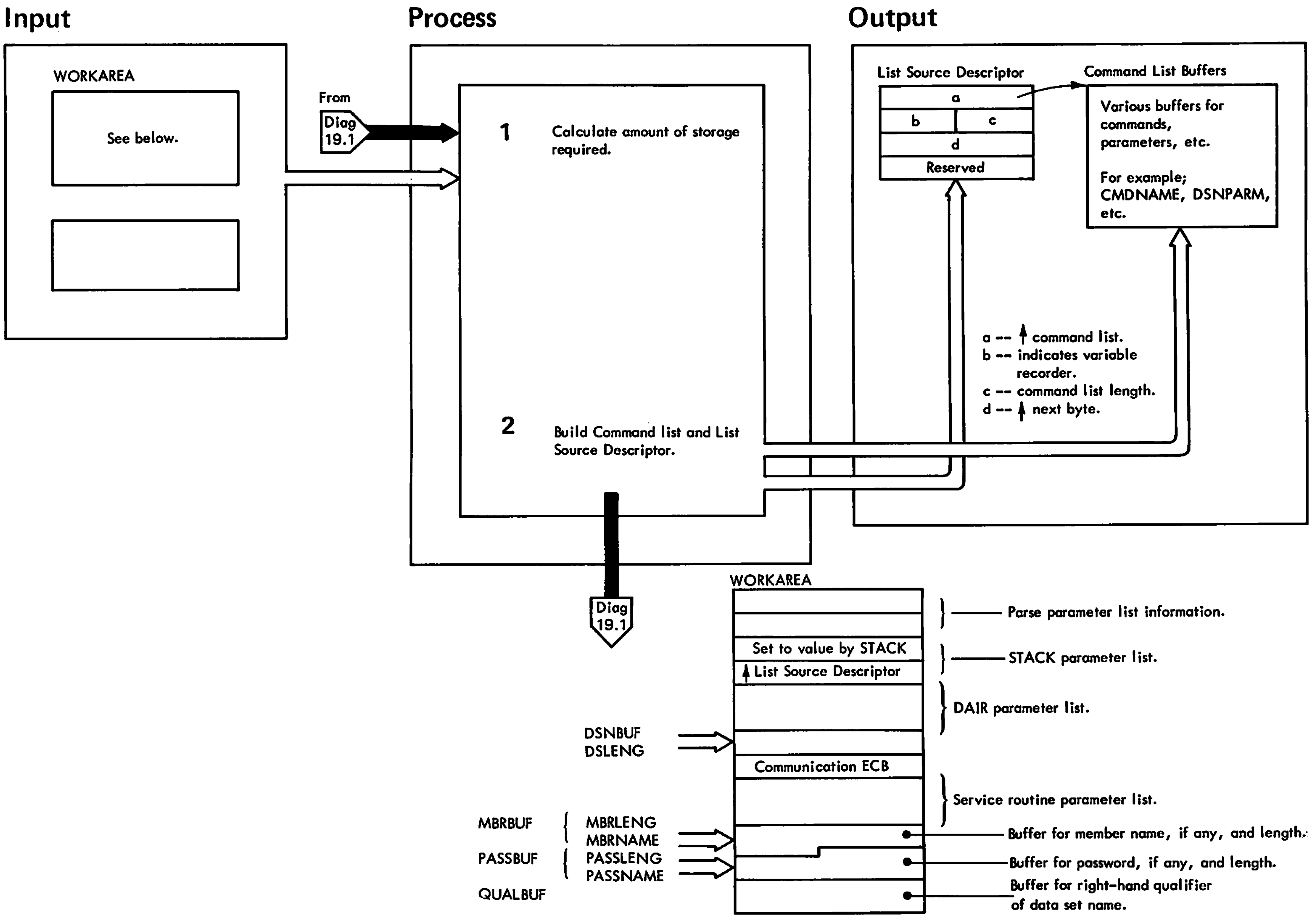

Input
Process
Output
WORKAREA
See below.
From
Diag 19.1
1
Calculate amount of storage required.
2
Build Command list and List Source Descriptor.
List Source Descriptor
a
b
c
d
Reserved
Command List Buffers
Various buffers for commands, parameters, etc.
For example; CMDNAME, DSNPARM, etc.
a -- ↑ command list.
b -- indicates variable recorder.
c -- command list length.
d -- ↑ next byte.
Diag 19.1
WORKAREA
Parse parameter list information.
Set to value by STACK
↑ List Source Descriptor
STACK parameter list.
DAIR parameter list.
DSNBUF
DSLENG
Communication ECB
Service routine parameter list.
MBRBUF
MBRLENG
MBRNAME
PASSBUF
PASSLENG
PASSNAME
QUALBUF
Buffer for member name, if any, and length.
Buffer for password, if any, and length.
Buffer for right-hand qualifier of data set name.

Diagram 19.2. Building a RUN Command List (Part 2 of 2)

Extended Description **Module** **Label**

1 WORKAREA fields and parse information previously located through the PDL are examined to calculate the amount of storage required for the command list. Included in the calculation are:

- The length of the list source descriptor (16 bytes).
- The size of the compiler command.
- The length of the data set name.
- Compiler parameters, if any.
- LOADGO command size (for ASM, FORT, PLI with OPT operand, or COBOL). This size includes control information length; LOADGO length; LOADGO data set name length; the length of the WHEN/END command, which is used to prevent execution of the program in the event the compiler does not complete successfully; parameter information, if any; COBLIB, PLIBASE, and FORTLIB length (for COBOL, PLI with OPT operand, and FORT data sets, LIB operand length, including the length of the data set list contained within parentheses).

2 The parameters are checked for validity with compiler types. Issue a message if they are invalid.

The command list and list descriptor are built. The list source descriptor is filled in as the command list is constructed.

First, the compiler command (type) is built. Control information consists of a two-byte length field followed by two bytes containing 0. The compiler command is moved to the appropriate buffer (CMDNAME).

Then the data set name is moved to the command list buffer (DSNPARM).

If a compiler parameter is specified (for BASIC, IPLI, or GOFORT), it is placed in the buffer, along with the parameter length.

If the compiler is ASM, FORT, PLI with OPT operand, or COBOL, the WHEN command is built and placed in the list. Then the LOADGO command is created. This consists of placing in the buffer the proper data set name, applicable parameters, and for the FORT, PLIBASE, and COBOL data sets, FORTLIB or COBLIB, respectively. The length of the LOADGO command is placed in the control field.

After the command list is complete, it is placed on the input stack (see step 4 of Diagram 19.1).

Object Module: IKJEFR00

Chapter 20. SEND Command Processing

SEND provides the facility to send short messages from a TSO user, background job, or operator to a TSO user. The message is displayed on the terminal if that user is logged on. The sender may request that a message sent to a user who is not logged on be stored for the receiver. Such messages can be retrieved using the LISTBC command. The sender may request that a message be stored even if the receiver is logged on.

Overview

The SEND (and LISTBC) function allows messages to be stored in a separate log for each user. PARMLIB settings allow the installation to supply a data set name for this log. Exits are provided to allow the installation to modify the data set name based on the parameters passed to the SEND command. This data set name may refer to a member of an existing PDS of each TSO user. If the data set name, as defined in PARMLIB and modified by the pre-save, pre-display, or initialization exit, is not found in the catalog, it is created for the user during LISTBC processing.

Instead of individual logs, the installation may choose to continue the use of SYS1.BRODCAST as a message repository. This may be done by setting the LOGNAME field in PARMLIB to 'SYS1.BRODCAST'. If user logs are not being used, only the initialization and termination exits are invoked.

LISTBC displays the notices found in SYS1.BRODCAST. LISTBC displays messages that have been stored for the user in the user's private log, if one has been defined. At the option of the installation, it also lists any messages for the user found in SYS1.BRODCAST. For SEND, as for LISTBC, PARMLIB and exits are used to determine the data set name of the user's private log. An exit allows the installation to process the message text before display. After reading and processing the messages, LISTBC removes the messages from the user's log and/or SYS1.BRODCAST.

Log Storage Implications

User logs impact direct storage access device utilization. If each user has a new file dedicated to the SEND log, a minimum of one track will be used for each user. The installation may choose to set LOGNAME to a member of a file most users already have or exits can be used to implement other access methods that require less than one track per user.

Diagnosing a SEND Problem

This section describes diagnostic information provided to solve a problem with the SEND command processor. Before going further you may wish to create a search argument and use it to search a problem data base to see if there is already a fix for the problem. For information about searching a problem data base, see "Creating a Search Argument" in *TSO Extensions System Diagnosis: Guide and Index.*

SEND Return Codes

SEND provides the following decimal return codes if user logs are being used:

0 -- Message was successfully sent for display; all users received
4 -- Message successfully stored; user not logged on
8 -- Message successfully stored; forced save
12 -- Message was not displayed; user not logged on
16 -- Message was not displayed; user's terminal is busy
20 -- Message was not displayed; user not accepting messages
24 -- Message was not stored; saving is not allowed
28 -- Message was not stored, user log not available
32 -- Message was not sent; installation exit denied access
36 -- Message was not sent; SEND is not active
40 -- Message was not sent; no such user
44 -- Message was not sent; command not invoked authorized
92 -- Message could not be sent; system error.

If user logs are not being used, SEND issues the following decimal return codes:

- 0 -- SEND processing was successful.
- 12 -- SEND processing was not successful.

Note: Because SEND propagates reason codes issued during processing of standard format exits, you may receive return codes other than those listed. For a description of standard format exit reason and return codes, see *TSO Extensions Customization.*

SEND Abend Codes

SEND does not issue any abend codes. The SEND command processor issues an SVC dump when an error occurs. It does not request a dump for the following system abend codes:

- '913'
- '13E'
- 'x22'
- 'x37'.

Services Used by SEND

The SEND command, as issued by a user and an operator, uses the following services:

- Data management services:
  - OPEN, READ, WRITE, CHECK
  - CLOSE, BLDL, STOW.

- MVS system services:
  - GETMAIN, FREEMAIN, LOAD, DELETE
  - ENQ, DEQ, WTO, TPUT, DYNALLOC (SVC 99)
  - MGCR (operator SEND only).

Only user SEND uses the following services:

- TSO services:
  - Parse, TSO Service Facility
  - PUTLINE, STACK, TCLEARQ, DAIR.
- MVS system services:
  - MODESET, TESTAUTH, SETRP
  - VRADATA, SDUMP, LINK.

SEND Dump Information

When an abend occurs, SEND furnishes the following information in the system diagnostic work area (SDWA):

- Load module name (SEND)
- Active module name
- Level of the active module
- Recovery module name (IKJEES01)
- Recovery label name (IKJEES01)
- Component ID (28502)
- Component ID base (5665)
- A functional description of the SEND command.

SEND furnishes the following information in the SDWA variable recording area (SDWAVRA) when issuing a dump:

- Name of the abending module
- Name of the recovery work area, followed by its address.

For information about finding the SDWA and SDWAVRA in a dump, see *TSO Extensions System Diagnosis: Guide and Index.*

TSO Extensions System Diagnosis: Guide and Index also provides information about reading messages, issuing traces and traps and calling IBM to report a problem you are unable to fix.

Exit Considerations

The SEND command processor invokes standard format exits that can change the processing of the command. For a description these exits, see *TSO Extensions Customization.*

TSO provides a common recovery routine for the standard format exits. When an abend occurs while attempting to invoke an exit, or during exit processing, the recovery routine furnishes the following information in the SDWA:

- Load module name (IKJRTR01)
- Active module name (IKJRTR01 or the active exit name)
- Level of the active module
- Recovery module name (RTRESTAE)
- Recovery label name (RTRESTAE)
- Component ID (28502)
- Component ID base (5665)

The common recovery routine furnishes the following information in the SDWAVRA when issuing a dump:

- Name of the exit handler parmlist, followed by its address.
- Name of the abending exit.

For unauthorized exits, a SNAP dump is issued; an SVC dump is issued for authorized exits.

Chapter 21. SUBMIT Command Processing

This section describes the logic of the **SUBMIT** command. It emphasizes the flow of data and control information through buffers and tables, and contains detailed functional descriptions through the use of method of operations diagrams. Figure 21-1 is the visual table of contents for the SUBMIT command.

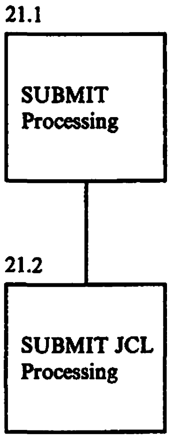
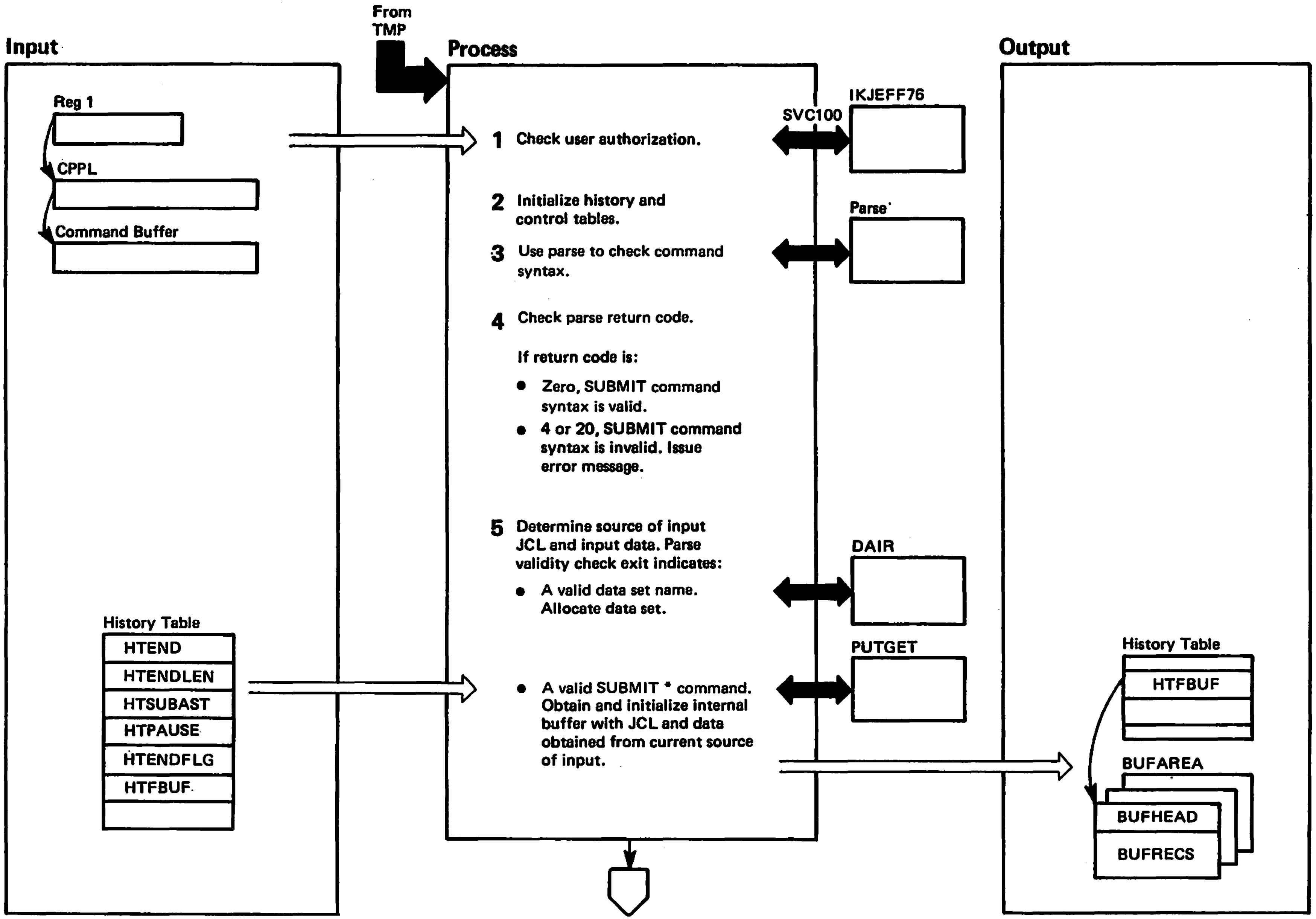


Figure 21-1. SUBMIT Command Processing Visual Table of Contents

Diagram 21.1. SUBMIT Processing (Part 1 of 4)



1 Check user authorization.

2 Initialize history and control tables.

3 Use parse to check command syntax.

4 Check parse return code.

If return code is:

- Zero, SUBMIT command syntax is valid.
- 4 or 20, SUBMIT command syntax is invalid. Issue error message.

5 Determine source of input JCL and input data. Parse validity check exit indicates:

- A valid data set name. Allocate data set.
- A valid SUBMIT * command. Obtain and initialize internal buffer with JCL and data obtained from current source of input.

Diagram 21.1. SUBMIT Processing (Part 2 of 4)

Extended Description **Module** **Label**

1 Use SVC100 to post the TMP (IKJEFTSC) requesting IKJEFF76 be attached under a parallel task structure. Information is passed to IKJEFF76 in the FIBPARMS parameter list. IKJEFF76 checks the user's authorization to enter the SUBMIT command. If the user is not authorized to enter foreground initiated background commands, the system issues an error message and returns control to the TMP.

Object Module: IKJEFF01

2 Initialize history and control tables for SUBMIT processing.

Object Module: IKJEFF04

3 Use parse to check the command syntax. Invoke parse validity check exit IKJEFF16 to get the fully qualified data set name from the default service routine and to check for unusual syntax errors. If the END keyword was specified, set the END flag on. If the PAUSE keyword was specified, set the PAUSE flag on.

Object Module: IKJEFF04

4 Check the parse return code.

- If the return code is zero, the SUBMIT command syntax is valid.
- If the return code is 4 or 20, the SUBMIT command syntax is invalid. If parse has not already issued an error message, issue an error message and, if necessary, prompt for valid data.

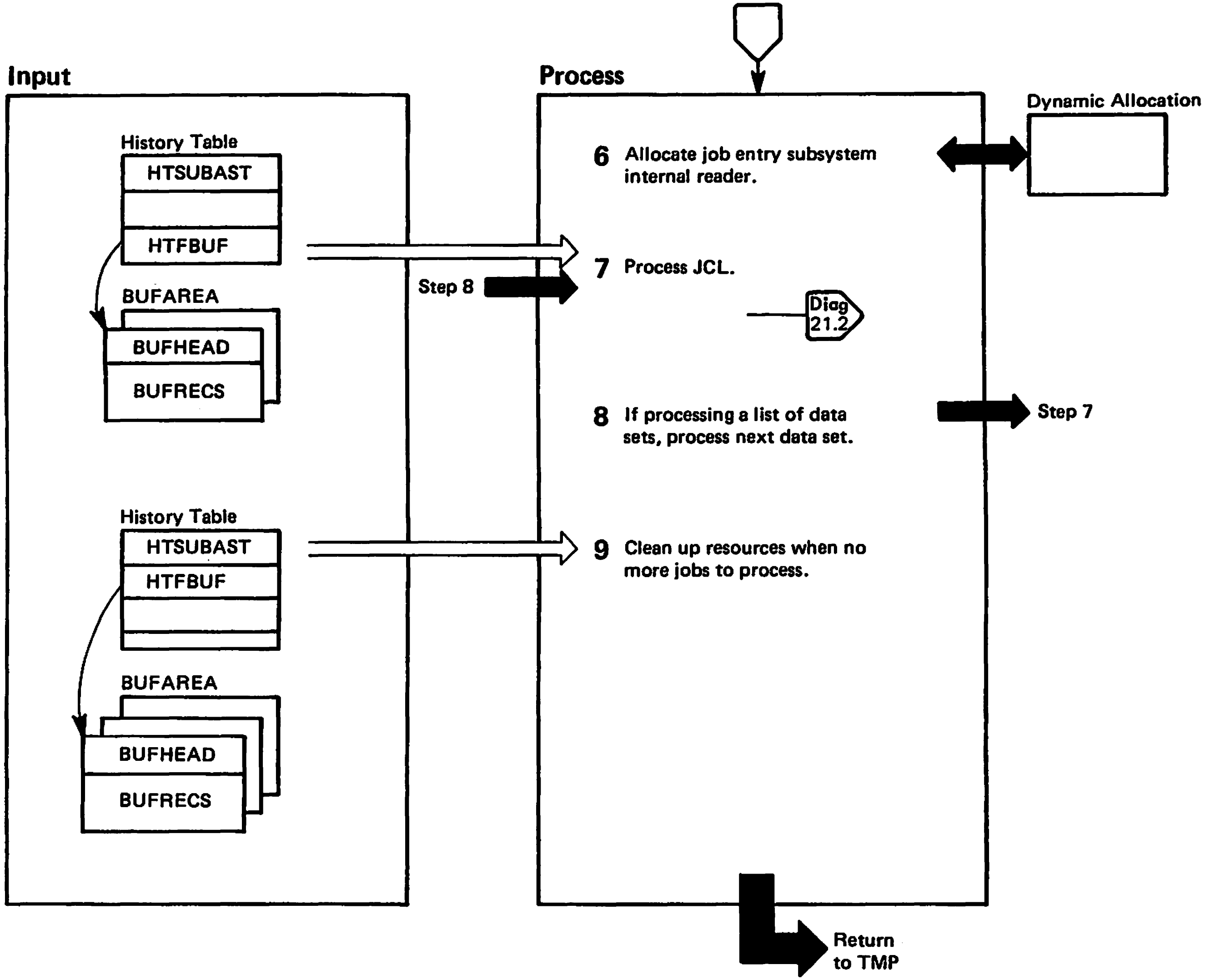
Object Module: IKJEFF04

5 If the parse return code is zero, determine the source of input JCL and input data:

- If the user entered a data set name other than *, check for PAUSE or END. If PAUSE and END are not present, use DAIR to allocate the input data set(s) containing the JCL and data. If either PAUSE or END is present, invoke IKJEFF02 to issue a message informing the user that the keyword has been ignored. Then, using DAIR, allocate the input data set(s).
- If the data set name is *, then:
  - Record the END characters and also whether PAUSE was specified.
  - Obtain an internal buffer via a GETMAIN. If the attempt is unsuccessful, invoke message issuer routine IKJEFF02 to issue an error message.
  - If the attempt is successful, prompt the user to enter the job stream. Use PUTGET to obtain the card images from the current source of input, e.g. terminal, CLIST.
  - If a buffer is returned and the card image information does not match the END characters, check the length of the card image. If the length is greater than 80 characters, send the user an error message, prompting him to reenter the input data. If the length is valid (80 characters or less), save the card image in the internal job stream buffer. Update the counter to indicate the number of card images in the job stream. Continue reading card images.
  - If a buffer is returned and the card image information matches the END characters, do not save the card image in the internal buffer. Instead, check if PAUSE was specified. If PAUSE was specified, pass control to the user, prompting to indicate whether the job stream should be submitted. If user indicates YES, proceed to step 6. If NO, proceed to step 9. If PAUSE was not specified, proceed to step 6.

Object Module: IKJEFF04

Diagram 21.1. SUBMIT Processing (Part 3 of 4)



6 Allocate job entry subsystem internal reader.

7 Process JCL.

8 If processing a list of data sets, process next data set.

9 Clean up resources when no more jobs to process.

Diagram 21.1. SUBMIT Processing (Part 4 of 4)

Extended Description **Module** **Label**

6 Use dynamic allocation to allocate a job entry subsystem internal reader and open the internal reader.

Object Module: IKJEFF15

For an attention interruption or ABEND, the internal reader will be closed and the last job submitted will be flushed.

Object Modules: IKJEFF20, IKJEFF15

7 Read and process JCL statements:

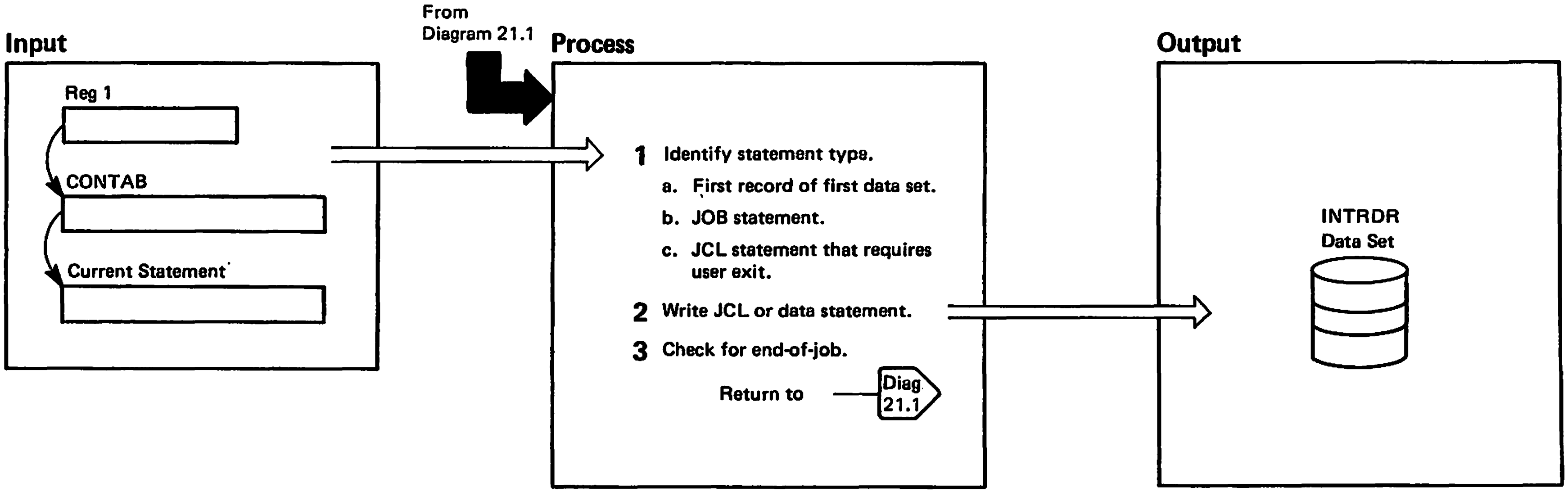
- If the user entered a data set name other than *, the JCL statements and data are located in the input data set. Read and process the input data set.
- If the data set name is *, the JCL statements and data are represented as card images in the internal buffer. Invoke the control routine, IKJEFF06, to process each image.

Object Module: IKJEFF05 (See Diagram 21.2)

8 If processing a list of data sets, return to step 7 to process the next data set in the list. If there are no more data sets to process, proceed to step 9.

9 Clean up resources when there are no more files to process or when the user does not want his job stream submitted. If SUBMIT * was not specified, free the input data set(s). If SUBMIT * was specified, free the input buffer(s). In both cases, return control to the TMP after the resources have been cleaned up.

Diagram 21.2. SUBMIT JCL Processing (Part 1 of 2)



1 Identify statement type.

a. First record of first data set.

b. JOB statement.

c. JCL statement that requires user exit.

2 Write JCL or data statement.

3 Check for end-of-job.

Diagram 21.2. SUBMIT JCL Processing (Part 2 of 2)

	Extended Description	Module	Label
1	Identify the statement as data or type of JCL (scans for JOB, EXEC, DD *, DD DATA, //XMIT, or /*XMIT.		

Object Module: IKJEFF07

- First record that is not a subsystem control card for first data set and not a JOB statement, create a JOB statement.

  Object Module: IKJEFF08

- JOB statement. Verify that the job name is not equal to the user ID. If it is equal, the user is prompted for an identifying character.

  Object Module: IKJEFF13

- JCL statement that requires user exit (IKJEFF10) for installation required processing.

  Object Modules: IKJEFF09, IKJEFF10

2 Write JCL or data statement to job entry subsystem internal reader data set.

3 Check for end-of-job. If an end-of-job, get a job ID from the job entry subsystem for the 'job submitted' message. If not end-of-job get the next statement. (See Diagram 21.1)

Object Module: IKJEFF05

Chapter 22. SYNC Command Processing

The SYNC command processor formats broadcast data set and synchronizes the user IDs in that data set with the User Attribute Data Set (UADS) and/or the TSO segment of the RACF data base. Its use is necessary when the UADS is created; its use thereafter causes all existing mail messages in the broadcast data set to be deleted.

Overview

SYNC formats the notices section of the broadcast data set to reserve room for the maximum number of broadcast messages. This maximum is set via the IKJBCAST macro. For information about issuing this macro instruction, see the broadcast data set specification section in *TSO Extensions Customization.*

SYNC formats the mail section of the broadcast data set with entries from the UADS and/or user IDs from the TSO segment of the RACF data base. This is done by reading directory entries for each user ID in the UADS, and by extracting user IDs from the TSO segment of the RACF data base.

A user must have TSO ACCOUNT authority in order to issue the SYNC command. If the command is invoked with the BOTH or RACF operands, the command must also be invoked authorized.

The RACF option has a dependency of RACF version 1.8 or higher being installed on the system. The BOTH option synchronizes the broadcast data set with the UADs data set if RACF version 1.8 or higher is not installed.

RACF special authority is not needed to issue the SYNC command. The SYNC command should be used in off hours due because it needs exclusive access to frequently used data sets.

SYNC Processing Summary

To synchronize the user IDs in the UADS and/or the TSO segment of the RACF data base with the broadcast data set, the SYNC command processor formats both the notices and the mail sections of the broadcast data set. It reserves enough room in the notices section for the maximum number of broadcast messages specified by the IKJBCAST macro. In the mail section, SYNC formats one directory entry for each user ID it reads from the UADS and/or the TSO segment of the RACF data base. For a description of the broadcast data set, see the "Broadcast Data Set Interface" section of the ACCOUNT command processor description in *TSO Extensions System Diagnosis: Command Processors, A-D.*

Diagnosing a SYNC Problem

This section describes diagnostic information provided to solve a problem with the SYNC command processor. Before going further you may wish to create a search argument and use it to search a problem data base to see if a there is already a fix for this problem. For information about searching a problem data base, see "Creating a Search Argument" in *TSO Extensions System Diagnosis: Guide and Index.*

SYNC Return Codes

The SYNC command provides a return code of zero upon successful synchronization of the broadcast data set. In all other cases, the SYNC command returns a decimal twelve return code.

SYNC Abend Codes

The SYNC command issues a system x'01C' abend code.

Zero is the only reason code; it denotes that an unexpected return code was received from the RACROUTE macro.

The SYNC command processor does not request a dump for the following system abend codes:

- 'x22'
- 'x37'
- '913'.

SYNC requests an SDUMP for all other abend codes if the command was invoked authorized. Otherwise, it requests a SNAP dump.

Services used by SYNC

The SYNC command uses the following services during processing:

- RACF services:
  - RACROUT interface to RACXTRT.
- TSO services:
  - Parse
  - Message services (IKJEFF02)
  - Broadcast data set I/O routine (IEEVSDIO)
  - UADS read routine (IKJEFA51).
- MVS services:
  - TESTAUTH, ENQ, DEQ, SDUMP
  - Attention processing services (STAX)
  - Recovery services
  - Data management services.

SYNC Dump Information

The SYNC command processor provides the following information in the system diagnostic work area (SDWA):

- Failing load module
- Failing CSECT
- Failing module level
- Recovery CSECT
- Recovery name
- Component ID
- Component ID base
- Functional description of the command that failed.

The SYNC command processor provides the following information in the SDWA variable recording area (SDWAVRA):

- Name and address of the recovery work area (IKJRBBWA).

For information about finding the SDWA and SDWAVRA in a dump, see *TSO Extensions System Diagnosis: Guide and Index.*

TSO Extensions System Diagnosis: Guide and Index also provides information about reading messages, issuing traces and traps and calling IBM to report a problem you are unable to fix.

Chapter 23. Directory

This section presents a module directory for the EDIT command, followed by a directory for the rest of the commands described in this book. For a complete listing of all TSO data areas, see *MVS/XA Data Areas (micrfiche).*

Directory for the EDIT Command

Name	Type	Load	Object	Entry	Alias	M.O. Diag.
E	Load	IKJEBEIN	IKJEBEIN	IKJEBEIN		1.1
EDIT	Alias	IKJEBEIN	IKJEBEIN	IKJEBEIN		
IKJEBEAA	Load		IKJEBEAA	IKJEBEAD		
				IKJEBEAS		
				IKJEBEDL		
				IKJEBEDR		
				IKJEBEDS		
				IKJEBEDU		
				IKJEBELO		
				IKJEBEMV		
				IKJEBERB		
				IKJEBERR		
				IKJEBEWA		
				IKJEBEWB		
				IKJEBEWP		
				IKJEBEWR		
				IKJEBEWS		
				IKJEBESY		
	Object	IKJEBEAA		IKJEBEAD		1.4, 1.5,
				IKJEBEAS		1.7, 1.8
				IKJEBEDL		
				IKJEBEDR		
				IKJEBEDS		
				IKJEBEDU		
				IKJEBELO		
				IKJEBEMV		
				IKJEBERB		
				IKJEBERR		
				IKJEBEWA		
				IKJEBEWB		
				IKJEBEWP		
				IKJEBEWR		
				IKJEBEWS		
				IKJEBESY		
IKJEBEAD	Entry	IKJEBEAA	IKJEBEAA			
IKJEBEAE	Object	IKJEBEMA		IKJEBEAE		1.1,1.2
	Entry	IKJEBEMA	IKJEBEAE			
IKJEBEAS	Entry	IKJEBEAA	IKJEBEAA			
IKJEBEAT	Object	IKJEBEMA		IKJEBEAT		1.1,1.3
	Entry	IKJEBEMA	IKJEBEAT			
IKJEBEBO	Load		IKJEBEBO	IKJEBEBO		
	Object	IKJEBEBO		IKJEBEBO		1.9
	Entry	IKJEBEBO	IKJEBEBO			

Name	Type	Load	Object	Entry	Alias	M.O. Diag.
IKJEBECH	Load		IKJEBECH	IKJEBECH		
			IKJEBESE	IKJEBESE		
	Object	IKJEBECH		IKJEBECH		1.10
	Entry	IKJEBECH	IKJEBECH			
IKJEBECI	Load		IKJEBECI	STAIEXIT		
				STAERTRY		
				STAEEXIT		
				IKJEBECI		
	Object	IKJEBECI		STAIEXIT		1.16,1.21,1.22,
				STAERTRY		1.25,1.28
				STAEEXIT		
				IKJEBECI		
	Entry	IKJEBECI	IKJEBECI			
IKJEBECO	Load		IKJEBECO	IKJEBECO		
	Object	IKJEBECO		IKJEBECO		1.21
	Entry	IKJEBECO	IKJEBECO			
IKJEBEDA	Load	IKJEBEDA	IKJEBEDA	IKJEBEDA		
	Object	IKJEBEDA		IKJEBEDA		1.16,1.21,
	Entry	IKJEBEDA	IKJEBEDA			1.25,1.28
IKJEBEDC	Alias	IKJEBEDR	IKJEBEDC	IKJEBEDC		
IKJEBEDC	Object	IKJEBERE		IKJEBEDC		1.24,1.25
		IKJEBERU		IKJEBEDC		
	Entry	IKJEBERE	IKJEBEDC			
	Entry	IKJEBERU	IKJEBEDC			
IKJEBEDE	Load		IKJEBEDE	IKJEBEDE		
	Object	IKJEBEDE		IKJEBEDE		1.12
	Entry	IKJEBEDE	IKJEBEDE			
IKJEBEDL	Entry	IKJEBEAA	IKJEBEAA			
IKJEBEDO	Load		IKJEBEDO	IKJEBEDO		
	Object	IKJEBEDO		IKJEBEDO		1.13
	Entry	IKJEBEDO	IKJEBEDO			
IKJEBEDR	Entry	IKJEBEAA	IKJEBEAA			
IKJEBEDS	Entry	IKJEBEAA	IKJEBEAA			
IKJEBEDU	Entry	IKJEBEAA	IKJEBEAA			
IKJEBEDX	Alias	IKJEBERE	IKJEBEDX			
	Object	IKJEBERE		IKJEBEDX		1.24
IKJEBEEN	Load		IKJEBEEN	IKJEBEXT		
				IKJEBEEN		
	Object	IKJEBEEN		IKJEBEXT		1.1,1.14,1.28
				IKJEBEEN		
	Entry	IKJEBEEN	IKJEBEEN			
IKJEBEEX	Alias	IKJEBEEN	IKJEBEEX	IKJEBEEX		
	Object	IKJEBEEN		IKJEBEEX		1.4,1.14,1.22
	Entry	IKJEBEEN	IKJEBEEX			
IKJEBEFC	Load		IKJEBEFC	IKJEBEFC		
	Object	IKJEBEFC		IKJEBEFC		1.16,1.22,1.26,
	Entry	IKJEBEFC	IKJEBEFC			1.27,1.29
IKJEBEFI	Load		IKJEBEFI	IKJEBEFI		
			IKJEBESE	IKJEBESE		
	Object	IKJEBEFI		IKJEBEFI		1.15
	Entry	IKJEBEFI	IKJEBEFI			
IKJEBEFO	Load		IKJEBEFO	IKJEBEFO		
	Object	IKJEBEFO		IKJEBEFO		1.16
	Entry	IKJEBEFO	IKJEBEFO			

Name	Type	Load	Object	Entry	Alias	M.O. Diag.
IKJEBEHE	Load		IKJEBEHE	IKJEBEHE		
	Object	IKJEBEHE		IKJEBEHE		1.23
	Entry	IKJEBEHE	IKJEBEHE			
IKJEBEIA	Load		IKJEBEIA	IKJEBEIA		
	Object	IKJEBEIA		IKJEBEIA		
	Entry	IKJEBEIA	IKJEBEIA			
IKJEBEIN	Alias	E	IKJEBEIN	IKJEBEIN		
				IKJEBEIN	EDIT	
	Object	E		IKJEBEIN	E	1.1,1.18,1.22
				IKJEBEIN	EDIT	
	Entry	E	IKJEBEIN		EDIT	
IKJEBEIP	Load		IKJEBEIP	IKJEBEIP		
	Object	IKJEBEIP		IKJEBEIP		1.18
	Entry	IKJEBEIP	IKJEBEIP			
IKJEBEIS	Load		IKJEBEIS	IKJEBEIS		
	Object	IKJEBEIS		IKJEBEIS		1.18,1.19
	Entry	IKJEBEIS	IKJEBEIS			
IKJEBELE	Load		IKJEBELE	IKJEBELE		
	Object	IKJEBELE		IKJEBELE		1.10,1.15,
	Entry	IKJEBELE	IKJEBELE			1.18,1.20
IKJEBELI	Load		IKJEBELI	IKJEBELI		
	Object	IKJEBELI		IKJEBELI		1.20
	Entry	IKJEBELI	IKJEBELI			
IKJEBELO	Entry	IKJEBEAA	IKJEBEAA			
IKJEBELT	Load		IKJEBELT	IKJEBELT		
	Object	IKJEBELT		IKJEBELT		1.21
	Entry	IKJEBELT	IKJEBELT			
IKJEBEMA	Load		IKJEBEAE	IKJEBEAE	IKJEBENT	
			IKJEBEAT	IKJEBEAT		
			IKJEBEMA	MAAERTRY		
				IKJEBMA2		
				IKJEBEMA		
			IKJEBEUT	IKJEBEUT		
	Object	IKJEBEMA		MAAERTRY		1.1,1.2,
				IKJEBMA2		1.18,1.28
				IKJEBEMA		
	Entry	IKJEBEMA	IKJEBEMA			
IKJEBEML	Load		IKJEBEMC	IKJEBEMC	IKJEBEMM	
IKJEBEME	Load		IKJEBEME	IKJEBEME		
	Object	IKJEBEME		IKJEBEME		1.22
	Entry	IKJEBEME	IKJEBEME			
IKJEBEMM	Alias	IKJEBEMC	IKJEBEMC			
IKJEBEMR	Load		IKJEBEMR	IKJEBEMR		
	Object	IKJEBEMR		IKJEBEMR		1.22,1.25
	Entry	IKJEBEMR	IKJEBEMR			
IKJEBEMS	Load		IKJEBEMS	IKJEBEMS		
	Object	IKJEBEMS		IKJEBEMS		1.1
	Entry	IKJEBEMS	IKJEBEMS			
IKJEBEMV	Entry	IKJEBEAA	IKJEBEAA			
	Entry					
IKJEBEM1	Load		IKJEBEM1			
	Object	IKJEBEM1				
	Entry					

Name	Type	Load	Object	Entry	Alias	M.O. Diag.
IKJEBEM2	Load		IKJEBEM2			
	Object	IKJEBEM2				
	Entry					
IKJEBEM3	Load		IKJEBEM3			
	Object	IKJEBEM3				
	Entry					
IKJEBEM4	Load		IKJEBEM4			
	Object	IKJEBEM4				
	Entry					
IKJEBEM5	Load		IKJEBEM5			
	Object	IKJEBEM5				
	Entry					
IKJEBEM6	Load		IKJEBEM6			
	Object	IKJEBEM6				
	Entry					
IKJEBEM7	Load		IKJEBEM7			
	Object	IKJEBEM7				
IKJEBEPD	Entry	IKJEBEPS	IKJEBEPD			
IKJEBEPS	Load		IKJEBEPS	IKJEBEPS		
			IKJEBEPD	IKJEBEPD		
	Object	IKJEBEPS		IKJEBEPS		1.1
				IKJEBEPD		
	Entry	IKJEBEPS	IKJEBEPS			
IKJEBERB	Entry	IKJEBEAA	IKJEBEAA			
IKJEBERE	Load		IKJEBEDC	IKJEBEDC	IKJEBEDX	
					IKJEBRE5	
			IKJEBEDX	IKJEBEDX		
			IKJEBERE	IKJEBERE		
	Object	IKJEBERE		IKJEBERE		1.25
	Entry	IKJEBERE	IKJEBERE			
IKJEBERN	Load		IKJEBERN	IKJEBERN		
	Object	IKJEBERN		IKJEBERN		1.25
	Entry	IKJEBERN	IKJEBERN			
IKJEBERR	Entry	IKJEBEAA	IKJEBEAA			
IKJEBERU	Load		IKJEBEDC	IKJEBEDC	IKJEBEDC	
			IKJEBERU	IKJEBERU		
	Object	IKJEBERU		IKJEBERU		1.26
	Entry	IKJEBERU	IKJEBERU			
IKJEBESA	Load		IKJEBESA	IKJEBESA		
	Object	IKJEBESA		IKJEBESA		1.14,1.27
	Entry	IKJEBESA	IKJEBESA			
IKJEBESC	Load		IKJEBESC	IKJEBESC		
	Object	IKJEBESC		IKJEBESC		1.1,1.14,1.28
	Entry	IKJEBESC	IKJEBESC			
IKJEBESE	Alias	IKJEBECH	IKJEBESE			
IKJEBESE	Object	IKJEBECG		IKJEBESE		1.10,1.15
		IKJEBEFI		IKJEBESE		
	Entry	IKJEBECG	IKJEBESE			
		IKJEBEFI	IKJEBESE			
IKJEBESU	Load		IKJEBESU	IKJEBESU		
	Object	IKJEBESU		IKJEBESU		1.29
	Entry	IKJEBESU	IKJEBESU			
IKJEBETA	Load		IKJEBETA	IKJEBETA		
	Object	IKJEBETA		IKJEBETA		1.30
	Entry	IKJEBETA	IKJEBETA			

Name	Type	Load	Object	Entry	Alias	M.O. Diag.
IKJEBETO	Load		IKJEBETO	IKJEBETO		
	Object	IKJEBETO		IKJEBETO		1.31
	Entry	IKJEBETO	IKJEBETO			
IKJEBEUI	Load		IKJEBEUI	IKJEBEUI		
	Object	IKJEBEUI		IKJEBEUI		1.1,1.4
	Entry	IKJEBEUI	IKJEBEUI			
IKJEBEUN	Load		IKJEBEUN	IKJEBEUN		
	Object	IKJEBEUN		IKJEBEUN		
	Entry	IKJEBEUN	IKJEBEUN			
IKJEBEUP	Load		IKJEBEUP	IKJEBEUP		
	Object	IKJEBEUP		IKJEBEUP		1.33
	Entry	IKJEBEUP	IKJEBEUP			
IKJEBEUT	Alias		IKJEBEUT	IKJEBEUT		
	Object	IKJEBEMA		IKJEBEUT		1.4
		IKJEBEUT		IKJEBEUT		
	Entry	IKJEBEMA	IKJEBEUT			
		IKJEBEUT	IKJEBEUT			
IKJEBEVE	Load		IKJEBEVE	IKJEBEVE		
	Object	IKJEBEVE		IKJEBEVE		1.34
	Entry	IKJEBEVE	IKJEBEVE			
IKJEBEWA	Entry	IKJEBEAA	IKJEBEAA			
IKJEBEWB	Entry	IKJEBEAA	IKJEBEAA			
IKJEBEWP	Entry	IKJEBEAA	IKJEBEAA			
IKJEBEWR	Entry	IKJEBEAA	IKJEBEAA			
IKJEBEWS	Entry	IKJEBEAA	IKJEBEAA			
IKJEBESY	Entry	IKJEBEAA	IKJEBEAA			
IKJEBEXT	Entry	IKJEBEEN	IKJEBEEN			
IKJEBMA2	Entry	IKJEBEMA	IKJEBEMA			
IKJEBRE5	Alias	IKJEBERE				
MAAERTRY	Entry	IKJEBEMA	IKJEBEMA			
STAEEXIT	Entry	IKJEBECI	IKJEBECI			
STAERTRY	Entry	IKJEBECI	IKJEBECI			
STAIEXIT	Entry	IKJEBECI	IKJEBECI			

The following module information relates to the command processors documented in this book (except EDIT).

Name	Type	Load	Object	Entry	Alias	M.O. Diag.
AKJLKL01	Object	AKJLKL01		AKJLKL01		LINK/LOADGO
	Entry	AKJLKL01	AKJLKL01			LINK/LOADGO
AKJLKL02	Object	AKJLKL02		AKJLKL02		LINK/LOADGO
	Entry	AKJLKL02	AKJLKL02			LINK/LOADGO
AKJLKMSG	Object	AKJLKL01				LINK/LOADGO
EX	Alias	EXEC	IKJCT430	IKJCT430		EXEC
EXEC	Load		IKJCT430	IKJCT430	EX	EXEC
			IKJCT431			EXEC
			IKJCT432			EXEC
			IKJCT43A			EXEC
FREE	Load		IKJEFD20	IKJEFD20		FREE
H	Alias	HELP	IKHEFH01	IKJEFH01		HELP
HELP	Load		IKJEFH00			HELP
			IKJEFH01	IKJEFH01	H	HELP
			IKJEFH02			HELP
			IKJEFH03			HELP
	Object		IKJEFH04			HELP
IEEVSDIO	Object	SEND				SEND
IGC0010{ ({=x'C0')	Load		IKJEFF00	IKJEFF00		See Note
			IKJEFF20			See Note
IKJCT430	Object	EXEC		IKJCT430		EXEC
	Entry	EXEC	IKJCT430			EXEC
IKJCT431	Object	EXEC				EXEC
IKJCT432	Object	EXEC				EXEC
IKJCT43A	Object	EXEC				EXEC
IKJCT460	Object	IKJCT469				OUTPUT
IKJCT462	Object	IKJCT469				OUTPUT
IKJCT463	Object	IKJCT469				OUTPUT
IKJCT464	Object	IKJCT469				OUTPUT
IKJCT466	Object	OUTPUT		IKJCT466	OUT	OUTPUT
	Entry	OUTPUT	IKJCT466		OUT	OUTPUT
IKJCT467	Object	IKJCT469			IKJCT467	
	Alias	IKJCT469	IKJCT467			
IKJCT469	Load		IKJCT460			OUTPUT
			IKJCT462			OUTPUT
			IKJCT463			OUTPUT
			IKJCT464			OUTPUT
			IKJCT469	IKJCT469		OUTPUT
			IKJCT470			OUTPUT
			IKJCT471			OUTPUT
			IKJCT472			OUTPUT
			IKJCT473			OUTPUT
	Object	IKJCT469		IKJCT469		OUTPUT
	Entry	IKJCT469	IKJCT469			OUTPUT
IKJCT470	Object	IKJCT469				OUTPUT
IKJCT471	Object	IKJCT469				OUTPUT

Note: IGC0010{ is invoked during CANCEL,OPERATOR, OUTPUT,PROFILE,STATUS, and SUBMIT processing.

Name	Type	Load	Object	Entry	Alias	M.O. Diag.
IKJCT472	Object	IKJCT469				OUTPUT
IKJCT473	Object	IKJCT469				OUTPUT
IKJEES10	Object	SEND		IKJEES10	SE	SEND
	Entry	SEND	IKJEES10		SE	SEND
IKJEES11	Object	SEND				SEND
IKJEES20	Object	SEND				SEND
IKJEES70	Object	IKJEES73		IKJEES73		LISTBC
		LISTBC		IKJEES70	LISTB	LISTBC
	Entry	LISTBC	IKJEES70		LISTB	LISTBC
IKJEES73	Load		IKJEES70	IKJEES73		LISTBC
			IKJEES74			LISTBC
			IKJEES75			LISTBC
	Entry	IKJEES73	IKJEES70			LISTBC
IKJEES74	Object	IKJEES73				LISTBC
		LISTBC				LISTBC
IKJEES75	Object	IKJEES73				LISTBC
		LISTBC				LISTBC
IKJEE1A0	Object	OPERATOR				OPERATOR
IKJEE100	Object	OPERATOR		IKJEE101		OPERATOR
				IKJEE100	OPER	OPERATOR
	Entry	OPERATOR	IKJEE100		OPER	OPERATOR
IKJEE101	Entry	OPERATOR	IKJEE100			OPERATOR
IKJEE150	Object	OPERATOR				OPERATOR
IKJEFD20	Object	FREE		IKJEFD20		FREE
	Entry	FREE	IKJEFD20			FREE
IKJEFF00	Object	IGC0010{ ({=X'C0')		IKJEFF00		See Note
	Entry	IGC0010{ ({=X'C0')	IKJEFF00			See Note
IKJEFF01	Object	SUBMIT		IKJEFF01	SUB	SUBMIT
	Entry	SUBMIT	IKJEFF01		SUB	SUBMIT
IKJEFF02	Alias	IKJTSLAR	IKJTSLAR	IKJEFF02		SUBMIT
IKJEF02R	Load		IKJEF02R	IKJEF02R		SUBMIT
	Object	IKJEF02R		IKJEF02R		SUBMIT
	Entry	IKJEF02R	IKJEF02R			SUBMIT
IKJEFF03	Object	IKJEFF04			IKJEFF03	SUBMIT
	Alias	IKJEFF04	IKJEFF03			SUBMIT
IKJEFF04	Load		IKJEFF02			SUBMIT
			IKJEFF03		IKJEFF03	SUBMIT
			IKJEFF04	IKJEFF04		SUBMIT
			IKJEFF05			SUBMIT
			IKJEFF07			SUBMIT
			IKJEFF08			SUBMIT
			IKJEFF09			SUBMIT
			IKJEFF13			SUBMIT
			IKJEFF15			SUBMIT
			IKJEFF16			SUBMIT
	Object	IKJEFF04		IKJEFF04		SUBMIT
	Entry	IKJEFF04	IKJEFF04			SUBMIT
IKJEFF05	Object	IKJEFF04				SUBMIT
IKJEFF07	Object	IKJEFF04				SUBMIT

Note: IGC0010{ is invoked during CANCEL,OPERATOR,
OUTPUT,PROFILE,STATUS, and SUBMIT processing.

Name	Type	Load	Object	Entry	Alias	M.O. Diag.
IKJEFF08	Object	IKJEFF04				SUBMIT
IKJEFF09	Object	IKJEFF04				SUBMIT
IKJEFF10	Load		IKJEFF10	IKJEFF10		SUBMIT
	Object	IKJEFF10		IKJEFF10		SUBMIT
	Entry	IKJEFF10	IKJEFF10			SUBMIT
IKJEFF13	Object	IKJEFF04				SUBMIT
IKJEFF15	Object	IKJEFF04				SUBMIT
IKJEFF16	Object	IKJEFF04				SUBMIT
IKJEFF19	Load		IKJEFF19	IKJEFF19		SUBMIT
	Object	IKJEFF19		IKJEFF19		SUBMIT
	Entry	IKJEFF19	IKJEFF19			SUBMIT
IKJEFF20	Object	IKJEFF76				See Note
IKJEFH00	Object	HELP				HELP
IKJEFH01	Object	HELP		IKJEFH01	H	HELP
	Entry	HELP	IKJEFH01		H	HELP
IKJEFH02	Object	HELP				HELP
IKJEFH03	Object	HELP				HELP
IKJEFH04	Object	HELP				HELP
IKJEFLA1	Load		IKJEFLA1	IKJEFLA1	IKJEFLES	LOGON
			IKJEFLB		IKJLB1	LOGON
			IKJEFLC		IKJEFLC	LOGON
			IKJEFLCM			LOGON
			IKJEFLE			LOGON
			IKJEFLEA			LOGON
			IKJEFLG			LOGON
			IKJEFLGB			LOGON
			IKJEFLGH			LOGON
			IKJEFLGM			LOGON
			IKJEFLGN			LOGON
			IKJEFLH			LOGON
			IKJEFLI			LOGON
			IKJEFLJ		IKJLJ1	LOGON
			IKJEFLJA			LOGON
			IKJEFLJH			LOGON
			IKJEFLJU			LOGON
			IKJEFLK		IKJLK1	LOGON
			IKJEFLL			LOGON
			IKJEFLM		IKJLM1	LOGON
			IKJEFLPA			LOGON
			IKJEFLS			LOGON
	Object	IKJEFLA1				
IKJEFLA	Load		IKJEFLA	IKJEFLA		LOGON
			IKJEFLIO			LOGON
			IKJEFTBL			LOGON
	Object	IKJEFLA				LOGON
IKJEFLB	Object	IKJEFLA1				LOGON
IKJEFLC	Object	IKJEFLA1				LOGON
IKJEFLCM	Object	IKJEFLA1				LOGON
IKJEFLE	Object	IKJEFLA1				LOGON
IKJEFLEA	Object	IKJEFLA1				LOGON
IKJEFLG	Object	IKJEFLA1				LOGON
IKJEFLGB	Object	IKJEFLA1				LOGON

Note: IGC0010{ is invoked during CANCEL,OPERATOR,
OUTPUT,PROFILE,STATUS, and SUBMIT processing.

Name	Type	Load	Object	Entry	Alias	M.O. Diag.
IKJEFLGH	Object	IKJEFLA1				LOGON
IKJEFLGM	Object	IKJEFLA1				LOGON
IKJEFLGN	Object	IKJEFLA1				LOGON
IKJEFLI	Object	IKJEFLA1				LOGON
IKJEFLJ	Object	IKJEFLA1				LOGON
IKJEFLJA	Object	IKJEFLA1				LOGON
IKJEFLJH	Object	IKJEFLA1				LOGON
IKJEFLJU	Object	IKJEFLA1				LOGON
IKJEFLK	Object	IKJEFLA1				LOGON
IKJEFLL	Object	IKJEFLA1				LOGON
IKJEFLM	Object	IKJEFLA1				LOGON
IKJEFLPA	Object	IKJEFLA1				LOGON
IKJEFLS	Object	IKJEFLA1				LOGON
IKJEFR00	Load		IKJEFR00	IKJEFR00	RUN	RUN
					R	RUN
	Object	IKJEFR00		IKJEFR00	RUN	RUN
					R	RUN
	Entry	IKJEFR00	IKJEFR00		RUN	RUN
IKJEFT80	Object	TERMINAL		IKJEFT80	TERM	TERMINAL
	Entry	TERMINAL	IKJEFT80		TERM	TERMINAL
IKJEFT82	Object	PROFILE		IKJEFT82	PROF	PROFILE
	Entry	PROFILE	IKJEFT82		PROF	PROFILE
IKJEHAL1	Load		IKJEHAL1	IKJEHAL1	LISTA	LISTALC
	Object	IKJEHAL1		IKJEHAL1	LISTA	LISTALC
	Entry	IKJEHAL1	IKJEHAL1		LISTA	LISTALC
IKJEHDS1	Load		IKJEHDS1	IKJEHDS1	LISTDS	LISTDS
	Object	IKJEHDS1		IKJEHDS1	LISTDS	LISTDS
	Entry	IKJEHDS1	IKJEHDS1		LISTDS	LISTDS
IKJEHMEM	Load		IKJEHMEM	IKJEHMEM		LISTDS/LISTALC
	Object	IKJEHMEM		IKJEHMEM		LISTDS/LISTALC
	Entry	IKJEHMEM	IKJEHMEM			LISTDS/LISTALC
IKJEHPRO	Load		IKJEHPRO	IKJEHPRO	PROTECT	PROTECT
	Object	IKJEHPRO		IKJEHPRO	PROTECT	PROTECT
	Entry	IKJEHPRO	IKJEHPRO	IKJEHPRO	PROTECT	PROTECT
IKJEHREN	Load		IKJEHREN	IKJEHREN	RENAME	RENAME
	Object	IKJEHREN		IKJEHREN	RENAME	RENAME
	Entry	IKJEHREN	IKJEHREN		RENAME	RENAME
LINK	Load		LINK	LINK		LINK/LOADGO
	Object	LINK		LINK		LINK/LOADGO
	Entry	LINK	LINK			LINK/LOADGO
LISTA	Alias	IKJEHAL1	IKJEHAL1	IKJEHAL1		LISTALC
LISTB	Alias	LISTBC	IKJEES70	IKJEES70		LISTBC
LISTBC	Load		IKJEES70	IKJEES70	LISTB	LISTBC
			IKJEES74			LISTBC
			IKJEES75			LISTBC
LISTDS	Alias	IKJEHDS1	IKJEHDS1	IKJEHDS1		LISTDS
LOAD	Alias	LOADGO	LOADGO	LOADGO		LINK/LOADGO
LOADGO	Load		LOADGO	LOADGO	LOAD	LINK/LOADGO
	Object	LOADGO		LOADGO	LOAD	LINK/LOADGO
	Entry	LOADGO	LOADGO		LOAD	LINK/LOADGO
OPER	Alias	OPERATOR	IKJEE100	IKJEE100		OPERATOR
OPERATOR	Load		IKJEE1A0			OPERATOR
			IKJEE100	IKJEE101		OPERATOR
				IKJEE100	OPER	OPERATOR
			IKJEE150			OPERATOR

Name	Type	Load	Object	Entry	Alias	M.O. Diag.
OUT	Alias	OUTPUT	IKJCT466	IKJCT466		OUTPUT
OUTPUT	Load		IKJCT466	IKJCT466	OUT	OUTPUT
PROF	Alias	PROFILE	IKJEFT82	IKJEFT82		PROFILE
PROFILE	Load		IKJEFT82	IKJEFT82	PROF	PROFILE
PROTECT	Alias	IKJEHPRO	IKJEHPRO	IKJEHPRO		PROTECT
R	Alias	IKJEFR00	IKJEFR00			RUN
RENAME	Alias	IKJEHREN	IKJEHREN	IKJEHREN		RENAME
RUN	Alias	IKJEFR00	IKJEFR00	IKJEFR00		RUN
SE	Alias	SEND	IKJEES10	IKJEES10		SEND
SEND	Load		IEEVSDIO			SEND
			IKJEES10	IKJEES10	SE	SEND
			IKJEES11			SEND
			IKJEES20			SEND
SUB	Alias	SUBMIT	IKJEFF01	IKJEFF01		SUBMIT
SUBMIT	Load		IKJEFF01	IKJEFF01	SUB	SUBMIT

Chapter 24. Data Area Usage

This section presents data area usage for the EDIT command, followed by the rest of the commands described in this book.

EDIT Command

This section lists data areas that the EDIT command processor uses, and it describes the Syntax Checker control blocks, Communication Area, and Option Word.

The list of data areas is ordered by acronym and gives the macro name, the common name for the data area, and the EDIT modules that create or alter the data area. If no modules are indicated, the data area was created by a module other than an EDIT module.

All data areas in the list are documented in the *MVS/XA Data Areas*.

Acronym	Macro	Common Name	Module/Access
CA	IKJEBECA	EDIT Communication Area	IKJEBEIN (create), all other edit modules can alter the CA area.
CPPL	IKJCPPL	Command Processor	IKJEBECI (create)
CSOA	IKJCSOA	Command Scan Output Area	IKJEBECI (create) IKJEBEMA (create)
CVT	CVT	Communication Vector Table	
DAPB08	IKJDAP08	DAIR Entry Code 08	IKJEBEDA (create) IKJEBEIN (create) IKJEBESA (create)
DAPB18	IKJDAPB18	DAIR Entry Code 18	IKJEBEDA (create) IKJEBEEN (create) IKJEBEIN (create) IKJEBESA (create)
DAPB2C	IKJDAP2C	DAIR Entry Code 2C	IKJEBECI (create) IKJEBEDA (create)
DAPL	IKJDAPL	Dynamic Allocation Parameter List	IKJEBECI (alter) IKJEBEDA (alter) IKJEBEEN (alter) IKJEBEIN (create) IKJEBESA (alter)
DCB	DCBD	Data Control Block	IKJEBEAA (alter) IKJEBECO (alter) IKJEBEFC (alter) IKJEBEIN (alter) IKJEBESA (alter)
DFPB	IKJDFPB	Default Parameter Block	IKJEBEIN (create)
DFPL	IKJDFPL	Default Parameter List	IKJEBEIN (create)

Acronym	Macro	Common Name	Module/Access
DSCB	IECSDSL1	Data Set Control Block	
ECT	IKJECT	Environmental Control Table	IKJEBEAT (alter) IKJEBECI (alter) IKJEBEMA (alter)
IOPL	IKJIOPL	I/O Service Routine Parameter List	IKJEBECI (alter) IKJEBEEN (alter) IKJEBEIN (create) IKJEBEMA (alter) IKJEBEMS (alter) IKJEBESA (create)
PGPB	IKJPGPB	PUT/GET Parameter Block	IKJEBECI (create) IKJEBEEN (create) IKJEBEIN (create) IKJEBEMA (create)
PPL	IECDPPL	Purge Parameter List	IKJEBEIN (create) IKJEBESA (create)
PTPB	IKJPTPB	PUTLINE Parameter Block	IKJEBEMS (create) IKJEBESA (create)
SDWA	IHASDWA	System Diagnostic Work Area	
TAIE	IKJTAIE	Terminal Attention Interrupt Element	
TCB	IKJTCB	Task Control Block	
TIOT	IEFTIOT1	Task I/O Terminal	
UCB	IEFUCBOB	Unit Control Block	
UPT	IKJUPT	User Profile Table	
UTILWORK	IKJEBEUW	EDIT Utility Work Area	IKJEBEAA (alter) IKJEBECO (alter) IKJEBEEX (alter) IKJEBEMA (alter) IKJEBEUI (create) IKJEBEUT (alter)

Syntax Checker Control Blocks

The following tables describe the contents of the control blocks (Buffer, Syntax Checker Communication Area, and Option Word) the syntax checker parameter list points to. If either the DELETE or RUN subcommand calls the IPLI or BASIC syntax checker, the first word of the parameter list points to the command interface instead of the buffer.

Buffer

Disp Dec.	Disp Hex.	Field Name	Field Size	Contents
0	0	C	1	Number of records in buffer (maximum is 127); set bit zero to 1 when the syntax checker has scanned all records in the buffer.
1	1	Chain	3	Address of next buffer; set to zero if this is last buffer in chain.
4	4	Record	Variable	Line or lines of source input data to have syntax checked; can be fixed- or variable-length, numbered or unnumbered.

Command Interface (DELETE Subcommand)

Disp Dec.	Disp Hex.	Field Name	Field Size	Contents
0	0	WORD1	4	Reserved.
4	4	STARTV	4	Binary value of starting line number.
8	8	STOPV	4	Binary value of ending line number.

Command Interface (RUN Subcommand)

Disp Dec.	Disp Hex.	Field Name	Field Size	Contents
0	0	WORD1	4	Reserved.
4	4	CMDPTR	4	Pointer to command (if high-order byte is X'80', operands are present).
8	8	TMPPRM	4	Pointer to Terminal Monitor Program parameter list.

Syntax Checker Communication Area

Disp Dec.	Disp Hex.	Field Name	Field Size	Setting	Contents Meaning (Instructions to Syntax Checker)
0	0	None	1	bits 0-3	(where n=0 or 1).
				0nnn	First entry - obtain and initialize work area. If a buffer chain is supplied, set the relative line number counter to zero.
				1n1n	Last entry - release the work area and return; do not check syntax.
				1000	Normal entry - set relative line number counter to zero; check syntax.
				110n	Entry after return code of 8 (error - buffer checking incomplete) - continue to check syntax.
				1001	Entry after return code of 12 (complete statements have been checked, but last statement in input buffer is incomplete) - if there is no more input (chain address of last buffer or buffer address is zero), check the syntax of the incomplete statement and return; if there is a new buffer chain, that is, more input (chain address or buffer address is not zero), resume checking syntax at the incomplete statement.
				bits 4-7	Reserved.
1	1	None	4	xxxx	Address of work area stored by syntax checker on first entry.
4	4	None	4	xxxx	Initial entry - maximum statement size specified at SYSGEN (if 0, checker assumes sufficient storage for largest legal statement is available); entry after return code of 4 (error detected, syntax checking complete, second-level message present), or 8 (error detected, syntax checking incomplete) - address of error message area.
8	8	None	4	xxxx	Initial entry - Temporary work area; subsequent entries - address of second error message, if any.
12	C	None	4	xxxx	Temporary storage area used for GETMAIN.

Option Word

Disp	Disp	Field	Field		Contents	
Dec.	Hex.	Name	Size	Setting	Meaning	Syntax Checker
0	0	None	1	X'01.'	FORTRAN H level	FORTRAN
				X'01'	FORTRAN E level	FORTRAN
				X'02'	FORTRAN G level	FORTRAN
				X'03'	GOFORT	FORTRAN
				X'04'	FORTRAN G1	FORTRAN
				X'00'	IPLI level	IPLI
				X'01'	BASIC level	BASIC
				xxxx	Value of left source margin	PL1F
1	1	None	1	bits 0-5	Reserved	FORTRAN
				bit 6=1	FORTRAN G/G1/H Code and Go definition to be loaded on initial entry	FORTRAN
				bit 6=0	FORTRAN G/G1/H Code and Go definition not to be loaded on initial entry	FORTRAN
				bit 7=1	FORTRAN E definition to be loaded on initial entry	FORTRAN
				bit 7=0	FORTRAN E definition not to be loaded on initial entry	FORTRAN
				bit 0=1	Entry from INPUT, Insert, linenum, *, CHANGE	IPLI or BASIC
				bit 1=1	Entry from DELETE	IPLI or BASIC
				bit 2=1	Entry from MERGE or RENUM	IPLI or BASIC
				bit 3=1	Translation already complete	IPLI or BASIC
				bit 4=1	Entry from RUN	IPLI or BASIC
				bit 5=1	Reserved	IPLI or BASIC
				xxxx	Value of right source margin	PL1F
2	2	None	1	xxxx	Record length of fixed-length records; binary zero, if variable-length records.	All
3	3	None	1	bit 0=0	CHAR 60	PL1 or IPLI
				bit 0=1	CHAR 48	PL1 or IPLI
				bit 1=0	Line-numbered data set	All
				bit 1=1	Data set not line-numbered	All
				bit 2	Reserved	All
				bit 3=0	Diagnose an incomplete statement	All
				bit 3=1	Delayed scan - return with code of 12 if last statement in input buffer is incomplete; immediate scan - possible incomplete statement in buffer.	All
				bit 4=0	Fixed-length records	All
				bit 4=1	Variable-length records	All
				bit 5=0	Standard form source input	All
				bit 5=1	Free form source input	All
				bit 6=0		
				bit 6=1		
				bit 7=0	SCAN or SCAN ON specified	All
				bit 7=1	NO SCAN or SCAN OFF specified	All

Commands E - S

This section presents the area usage for the TSO commands E through S documented in this book, except for the EDIT command.

Acronym	Macro	Common Name	Command Processor	Module/Access
ACB	IFGACB	VSAM Access Method Control BLock	OUTPUT	IKJCT462 (create)
			SUBMIT	IKJEFF15 (create)
COMPROC	IKJEXEC	Command Procedure Storage Block	EXEC	IKJCT440 (create) IKJCT430 (alter) IKJCT432 (alter)
CONTAB	IKJEFFCT	SUBMIT Internal Control Table	SUBMIT	IKJEFF04 (create) IKJEFF15 (alter)
CPPL	IKJCPPL	Command Processor Parameter List	OUTPUT	IKJCT463 (create)
CPRB	MVSSERV	Connectivity Programming Request Block	MVSSERV	
CTGFL	IEZCTCFL	VSAM Catalog Control Field List	LISTALC	IKJEHAL1 (create)
			LISTDS	IKJEHDS1 (create)
			RENAME	IKJEHCIR (create)
CTGPL	IEZCTGPL	VSAM Catalog Parameter List	LISTALC	IKJEHAL1 (create)
			LISTDS	IKJEHDS1 (create)
			RENAME	IKJEHCIR (create)
DAPB00	IKJDAP00	DAIR Parameter Block 00	EXEC	IKJCT430 (create)
DAPB0C	IKJDAP0C	DAIR Parameter Block 0C	LINK/LOADGO	AKJLKL01 (create)
DAPB10	IKJDAP10	DAIR Parameter Block 10	LINK/LOADGO	AKJLKL01 (create) AKJLKL02 (create)
DAPB18	IKJDAP18	DAIR Parameter Block 18	EXEC	IKJCT430 (create)
			LINK/LOADGO	AKJLKL01 (create)
			OUTPUT	IKJCT473 (create)
			PROTECT	IKJEHPRO (create)
			RENAME	IKJEHREN (create)
DAPB1C	IKJDAP1C	DAIR Parameter Block 1C	LINK/LOADGO	AKJLKL01 (create)
			OUTPUT	IKJCT473 (create)
DAPB24	IKJDAP24	DAIR Parameter Block 24	HELP	IKJEFH01
DAPB28	IKJDAP28	DAIR Parameter Block 28	LINK/LOADGO	AKJLKL01 (create)
			SUBMIT	IKJEFF04 (create)
DAPB2C	IKJDAP2C	DAIR Parameter Block 2C	LISTDS	IKJEHDS1 (create)
			OPERATOR	IKJEE100 (create)
			OUTPUT	IKJCT463 (create)
DAPB34	IKJDAP34	DAIR Parameter Block 34	LINK/LOADGO	AKJLKL01 (create)
DCB	DCBD	Data Control Block DSECT	HELP	IKJEFH01 (create)
			LISTBC	IKJEES75 (create)
			OUTPUT	IKJCT463 (create) IKJCT469 (create) IKJCT471 (alter)
			SEND	IEEVSDI0 (create)
			SUBMIT	IKJEFF05 (create)
DFPB	IKJDFPB	Default Parameter Block	LISTDS	IKJEHDS1 (create)
			PROTECT	IKJEHPRO (create)
			RENAME	IKJEHREN (create)
DFPL	IKJDFPL	Default Parameter List	LISTDS	IKJEHDS1 (create)
			PROTECT	IKJEHPRO (create)
			RENAME	IKJEHREN (create)
D08ADDED	IKJEFFD8	SUBMIT Extension to DAPB08	SUBMIT	IKJEFF04 (alter) IKJEFF16 (create)

Acronym	Macro	Common Name	Command Processor	Module/Access
ECDA	IKJEXEC	Phase 1 Exit Common Data Area	EXEC	IKJCT440 (create)
				IKJCT431 (alter)
				IKJCT432 (alter)
EXECDATA	IKJEXEC	EXEC Command Control Data Area	EXEC	IKJCT440 (create)
ESTAEWA		ESTAE Exit Work Area	OUTPUT	IKJCT460 (alter)
				IKJCT464 (alter)
				IKJCT469 (create)
EXITL	IKJEFFIE	FIB Installation Exit Parameter List	SUBMIT	IKJEFF09 (create)
				IKJEFF10 (alter)
FFB2	IKJEFFB2	FIB Module's Parameter list from SVC 100	SUBMIT	IKJEFF04 (alter)
GFPARMS	IKJEFFGF	GNRLFAIL and VSAMFAIL (IKJEFF19) Parameter List	EXEC	IKJCT440 (create)
				IKJCT431 (create)
				IKJCT432 (create)
			OUTPUT	IKJCT467 (create)
			SUBMIT	IKJEFF05 (create)
				IKJEFF15 (create)
GTPB	IKJGTPB	GETLINE Parm Block	EXEC	IKJCT440 (create)
			LINK/LOADGO	AKJLKMSG (create)
			OPERATOR	IKJEE150 (create)
HISTORY	IKJEFFHT	SUBMIT Internal History Table	SUBMIT	IKJEFF04 (create)
				IKJEFF05 (alter)
				IKJEFF07 (alter)
				IKJEFF08 (alter)
				IKJEFF09 (alter)
				IKJEFF13 (alter)
				IKJEFF15 (alter)
				IKJEFF20 (alter)
IKJWHEN	IKJWHEN	WHEN Common Data Area	WHEN/END	IKJEFE11 (create)
				IKJEFE15 (alter)
INITTERM		Enhanced Connectivity Facility Area	MVSSERV	
LSD	IKJLSD	List Source Descriptor	EXEC	IKJCT440 (create)
			RUN	IKJEFR00 (create)
			WHEN/END	IKJEFE11 (create)
OUTCOMTB	IKJOCMTB	Output Communications Table	OUTPUT	IKJCT460 (alter)
				IKJCT462 (alter)
				IKJCT463 (alter)
				IKJCT464 (alter)
				IKJCT466 (create)
				IKJCT467 (alter)
				IKJCT469 (create)
				IKJCT470 (alter)
				IKJCT471 (alter)
				IKJCT472 (alter)
				IKJCT473 (alter)
PGPB	IKJPGPB	PUTGET Parameter Block	LINK/LOADGO	AKJLKMSG (create)
			OPERATOR	IKJEE100 (create)
			OUTPUT	IKJCT467 (create)
			SUBMIT	IKJEFF02 (create)
PTPB	IKJPTPB	PUTLINE Parameter Block	OPERATOR	IKJEE100 (create)
				IKJEE150 (create)
				IKJEE1A0 (create)
			SUBMIT	IKJEFF02 (create)
			TERMINAL	IKJEFT80 (create)
			WHEN/END	IKJEFE15 (create)

Acronym	Macro	Common Name	Command Processor	Module/Access
RPL	IFGRPL	VSAM Request Parameter List	OUTPUT	IKJCT462 (create) IKJCT470 (alter)
			SUBMIT	IKJEFF05 (alter) IKJEFF15 (create)
RTCT	IHARTCT	Recovery/Termination Control Table	OPERATOR	IKJEFF00 (alter)
SDWA	IHASDWA	System Diagnostic Work Area	OUTPUT	IKJCT460 (alter)
			SUBMIT	IKJEFF02 (alter)
SNTAB	IKJEXEC	Symbol Name Table	EXEC	IKJCT440 (create) IKJCT432 (alter)
STE	IHASTE	SLIP TSO Element	OPERATOR	IKJEFF00 (create)
SVTAB	IKJEXEC	Symbol Value Table	EXEC	IKJCT440 (create)
SWAEPA	IEFZB505	SWA Manager Parameter List	SUBMIT	IKJEFF04 (create)
TCB	IKJTCB	Task Control Block	OUTPUT	IKJCT463 (alter)
UPT	IKJUPT	User Profile Table	PROFILE	IKJEFT82 (alter)
WPL	IEZWPL	WTO/WTOR/MLWTO/WTP Parameter List	SUBMIT	IKJEFF02 (create)

Index

TSO Extensions
System Diagnosis:
Command Processors,
E-S

READER'S
COMMENT
FORM

LY28-1415-0

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Note: ***Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.***

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name, company, mailing address, and date:

____

____

____

____

What is your occupation? ____________________________________

How do you use this publication? ____________________________________

Number of latest Newsletter associated with this publication: ____________________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

TSO Extensions System Diagnosis: Command Processors, E-S

LY28-1415-0

S370-39

Reader's Comment Form

Cut or Fold Along Line

Fold and tape   Please Do Not Staple   Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department D58, Building 921-2
PO Box 390
Poughkeepsie, New York 12602

Fold and tape   Please Do Not Staple   Fold and tape

Printed in U.S.A.

IBM®

IBM®

Program Number
5665-285

File Number
S370-39

LY28-1415-0

Printed in U.S.A.