IBM

LY21-0050-0

File No. S34-36

IBM System/34 System Support Program Logic Manual: System

Program Number 5726-SS1

SSP LOGIC: SYSTEM

LY21-0050-0

File No. S34-36

Licensed Material
Property of IBM

IBM System/34
System Support Program
Logic Manual: System

Program Number 5726-SS1

SSP LOGIC: SYSTEM

First Edition (December 1977)

This edition applies to version 01, modification 00 of the IBM System/34 System Support Program Product (Program Number 5726-SS1) and to all subsequent versions and modifications until otherwise indicated in new editions or technical newsletters. Changes are periodically made to the information herein; before using this publication, refer to the latest *IBM System/34 Bibliography,* GH30-0231, for the editions that are applicable and current.

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

A Reader's Comment Form is provided at the back of this publication. If the form has been removed, address your comments to IBM Corporation, Publications, Department 245, Rochester, Minnesota 55901.

Preface

This logic manual is designed to aid IBM support personnel in maintaining the IBM System/34 by serving as a recall mechanism and guide to program listings. This manual does not contain directions for programming or operating the system.

Note: The System Support Program Product is documented by two manuals: this manual and *IBM System/34 System Support Program Logic Manual: Data Communications*, LY21-0051.

The SSP (System Support Program Product) provides the programming support necessary to start the system, process commands, start a job, run a job, and terminate a job. Its functions include:

- Main storage IPL
- Command processing
- Initiator
- Allocate
- Open
- Data management
- Diskette end of volume
- Keysort
- Spool intercept
- Spool print writer
- Close
- Termination
- System service programs
- System maintenance programs
- Overlay linkage editor
- System utility programs

This manual describes the SSP modules by providing:

- A system flow overview of the entire SSP.
- An overview of each of the primary components of the SSP.
- A detailed description including, as required: a description of the function, a HIPO diagram, and a logic flow diagram for each function contained within the SSP.
- Significant data areas, as required. For data areas not described in this manual, see the *Data Areas Handbook*.

Note: The arrowheads (▶◀) that occasionally appear in the HIPO extended descriptions are provided to help you line up the descriptions with their respective module references.

RELATED PUBLICATIONS

This publication references the following publications:

- *IBM System/34 System Data Areas and Diagnostic Aids*, LY21-0049
- *IBM System/34 System Support Reference Manual*, SC21-5155
- *IBM System/34 System Support Program Logic Manual: Data Communications*, LY21-0051
- *IBM System/34 Overlay Linkage Editor Reference Manual*, SC21-7707

Contents

PART 4. SYSTEM UTILITY PROGRAMS

APPENDIXES

Diagram numbers are assigned by chapter (for example, all diagrams in chapter 1 are numbered 1.nn). Any diagram with a zero preceding the first period (Diagram 0.1, for example) is a table of contents diagram for the SSP. Similiarly, any diagram with a zero following the first period is a table of contents diagram for its respective chapter. Each entry in a table of contents diagram identifies a particular input — process — output type of diagram. The input — process — output diagrams describe functions and are found in the *Method of Operation* section of each chapter of this manual.

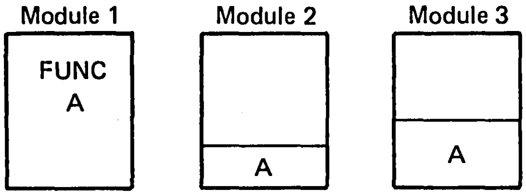
The main objective of the HIPO method of diagramming is to improve communication procedures and techniques through the effective use of functional operation diagrams.

HIPO, hierarchy plus input — process — output, is a method of graphically describing internal function by structuring a presentation from general to detailed levels in a set of method-of-operation diagrams.

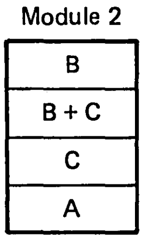
A visual table of contents is prepared for each set of functional diagrams. It shows:

- The structural relationships of the diagrams
- The contents of each of the diagrams
- A legend applying both to the individual function diagram and the total presentation

Each set of functional diagrams contains a visual table of contents, one or more overviews, and a number of low level diagrams showing the implementation and/or design of a function. The implementation of a major function can extend through many modules or routines.



A specific module might support parts of many functions. Sometimes these functions are not even related.

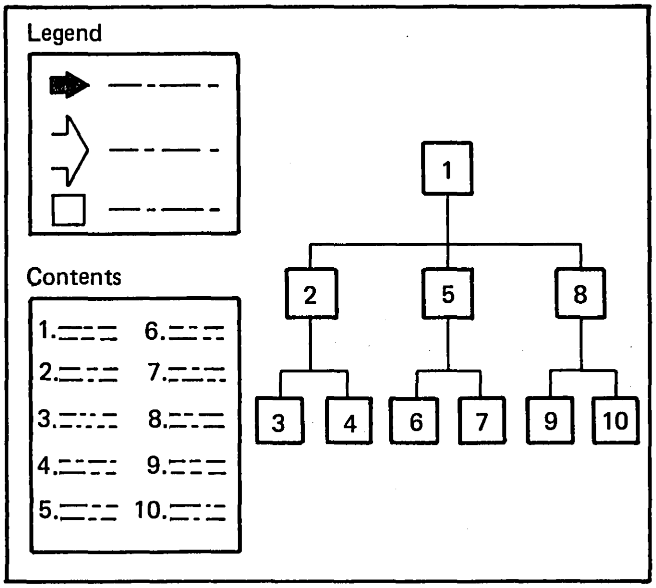


Someone who must modify an existing function, add a new function, test for regression, or fix the system must know:

- How the function is performed
- What other, perhaps nonrelated, functions use the same code

Graphic Content: The graphic content is determined by the situations shown.

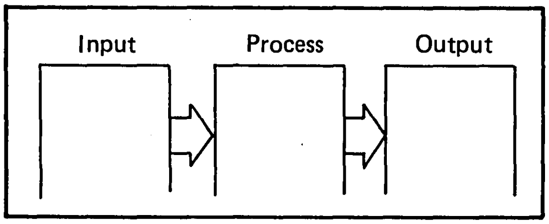
The table of content diagrams act as introductions to the functions and directors to the low level, detailed diagrams.



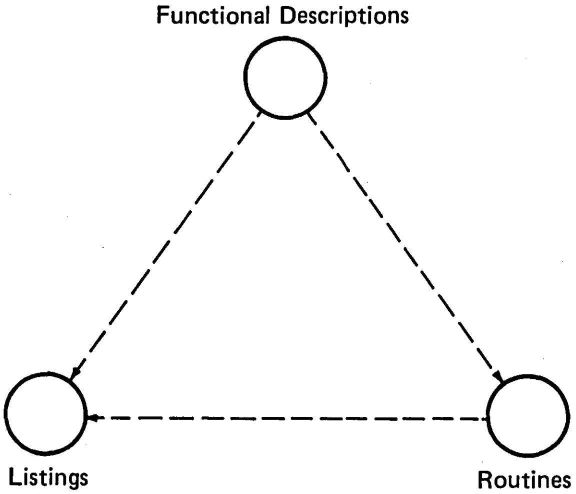
The low level diagrams contain unit level information (that level of information reflecting the actual workings of the system). Each low level diagram is arranged to best show:

- A process that supports the function being described
- Results of the process
- Requirements for processing

Stated graphically:



To tie together the program organization and functional areas, the function diagrams use cross-references from one area to another and to the listings.

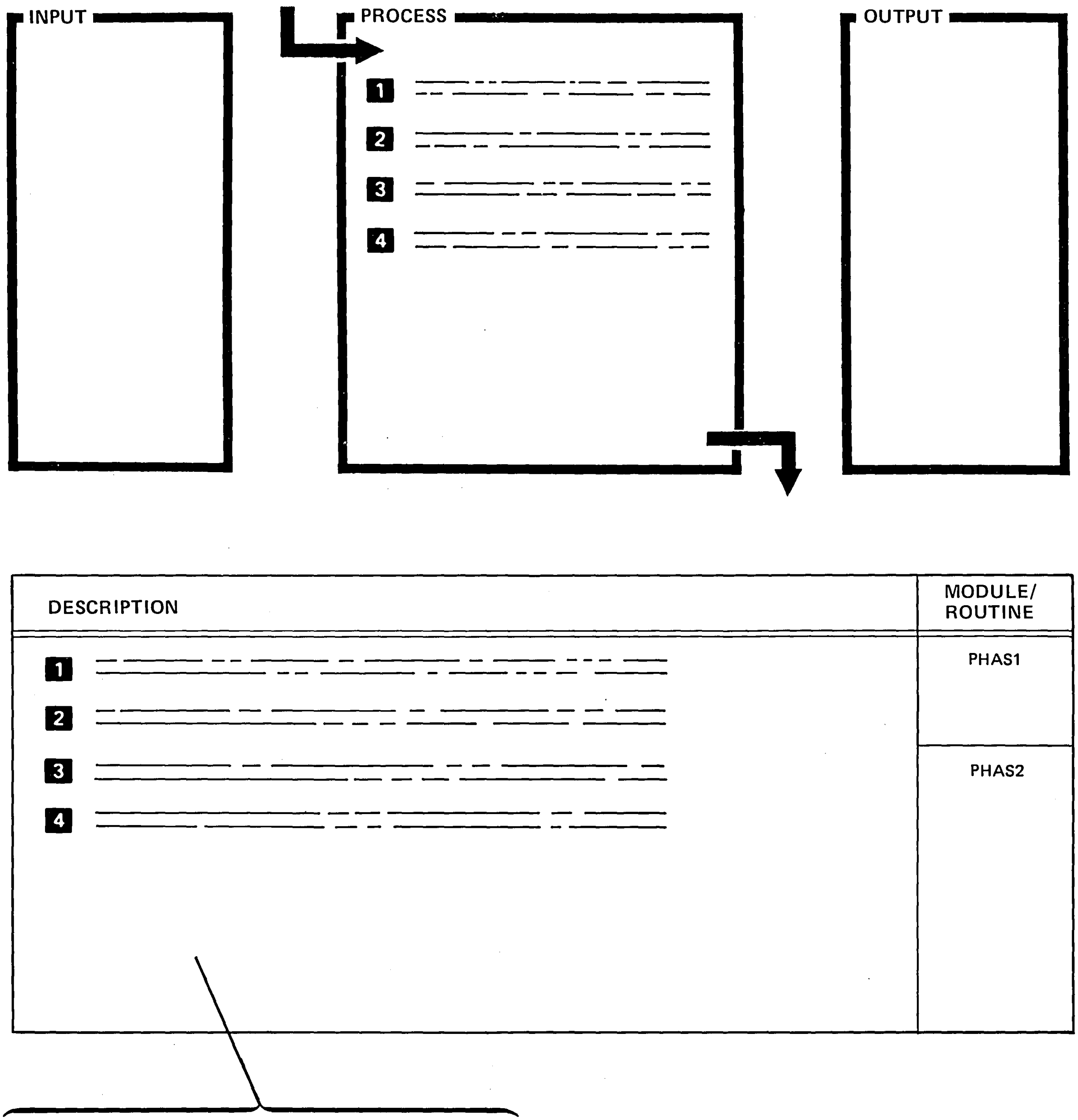


Informational Content: Functional diagrams should discuss *inputs, process,* and *results.* For the sake of clarity, these elements are presented in a normal reading sequence. Simple boxes are used to block off these three major areas of the diagram.

The picture area of the diagrams contains as few words as possible. There are two reasons for this:

- When the picture becomes cluttered with text, it loses some value as a recall mechanism.
- The degree of difficulty of maintaining the diagrams increases with increased number of words in the picture area.

In the diagrams, functions are related by cross-references to the module(s) supporting those functions. The description box of the diagram includes the references of the module(s) associated with the implementation steps that support the function.



Process steps 1 and 2 are performed in PHAS1. Before step 3, passing of control is implied by the horizontal line through the routine column. Steps 3 and 4 are performed in PHAS2.

Legend:

➡ Control flow (leads the reader through the process)

⇨ Data flow

1 Reference block

Part 1. Introduction

Introduction

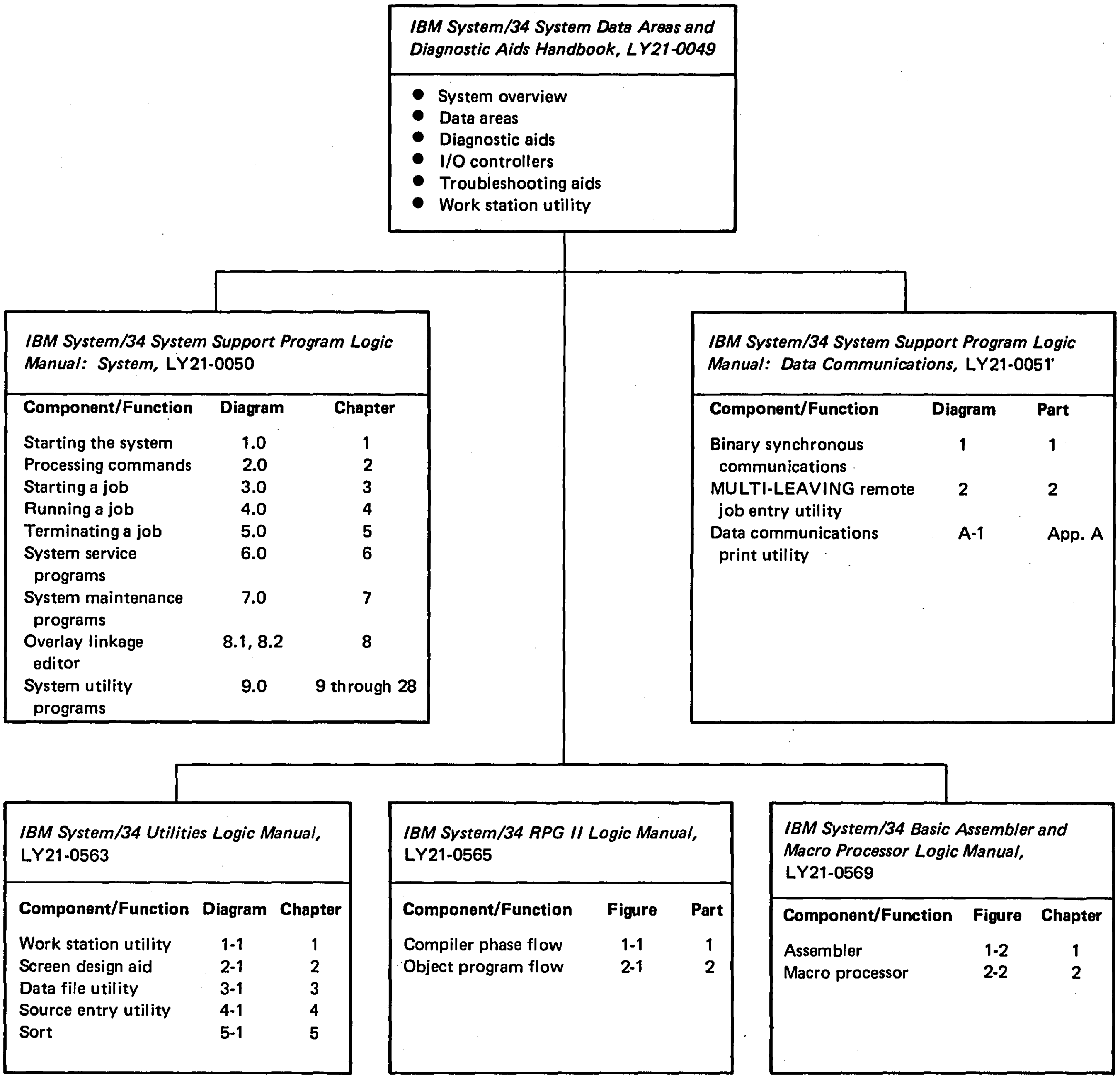
This program logic manual provides diagnostic information and serves as a recall mechanism for the IBM programming representative working on a System/34 problem. HIPO and control flow diagrams help guide the programming representative to the proper module on microfiche. If a main system module is not causing the problem, the directory in Appendix A can help locate additional information in this manual and on microfiche.

Figure 0-1 is an overview of the logic documentation available for System/34. It lists the major components of each PLM.

Figure 0-2 shows the system control flow and major divisions of the SSP. Part 2 of the manual follows the same major divisions as shown in this diagram.

Diagram 0.1 is an overview of the SSP. Within this overview are overviews of the system service programs, the system maintenance programs, the overlay linkage editor, and the utility programs.

Information presented in this manual, accompanied by information in the *Data Areas Handbook,* is used to interpret main storage dumps. Appropriate module listings on microfiche are then consulted as necessary to initiate a circumvention to the program problem and to write an APAR. The *Data Areas Handbook* can then be used to help apply the program temporary fix.



IBM System/34 System Data Areas and Diagnostic Aids Handbook, LY21-0049

- System overview
- Data areas
- Diagnostic aids
- I/O controllers
- Troubleshooting aids
- Work station utility

IBM System/34 System Support Program Logic Manual: System, LY21-0050

Component/Function	Diagram	Chapter
Starting the system	1.0	1
Processing commands	2.0	2
Starting a job	3.0	3
Running a job	4.0	4
Terminating a job	5.0	5
System service programs	6.0	6
System maintenance programs	7.0	7
Overlay linkage editor	8.1, 8.2	8
System utility programs	9.0	9 through 28

IBM System/34 System Support Program Logic Manual: Data Communications, LY21-0051

Component/Function	Diagram	Part
Binary synchronous communications	1	1
MULTI-LEAVING remote job entry utility	2	2
Data communications print utility	A-1	App. A

IBM System/34 Utilities Logic Manual, LY21-0563

Component/Function	Diagram	Chapter
Work station utility	1-1	1
Screen design aid	2-1	2
Data file utility	3-1	3
Source entry utility	4-1	4
Sort	5-1	5

IBM System/34 RPG II Logic Manual, LY21-0565

Component/Function	Figure	Part
Compiler phase flow	1-1	1
Object program flow	2-1	2

IBM System/34 Basic Assembler and Macro Processor Logic Manual, LY21-0569

Component/Function	Figure	Chapter
Assembler	1-2	1
Macro processor	2-2	2

Figure 0-1. System/34 Logic Documentation Overview

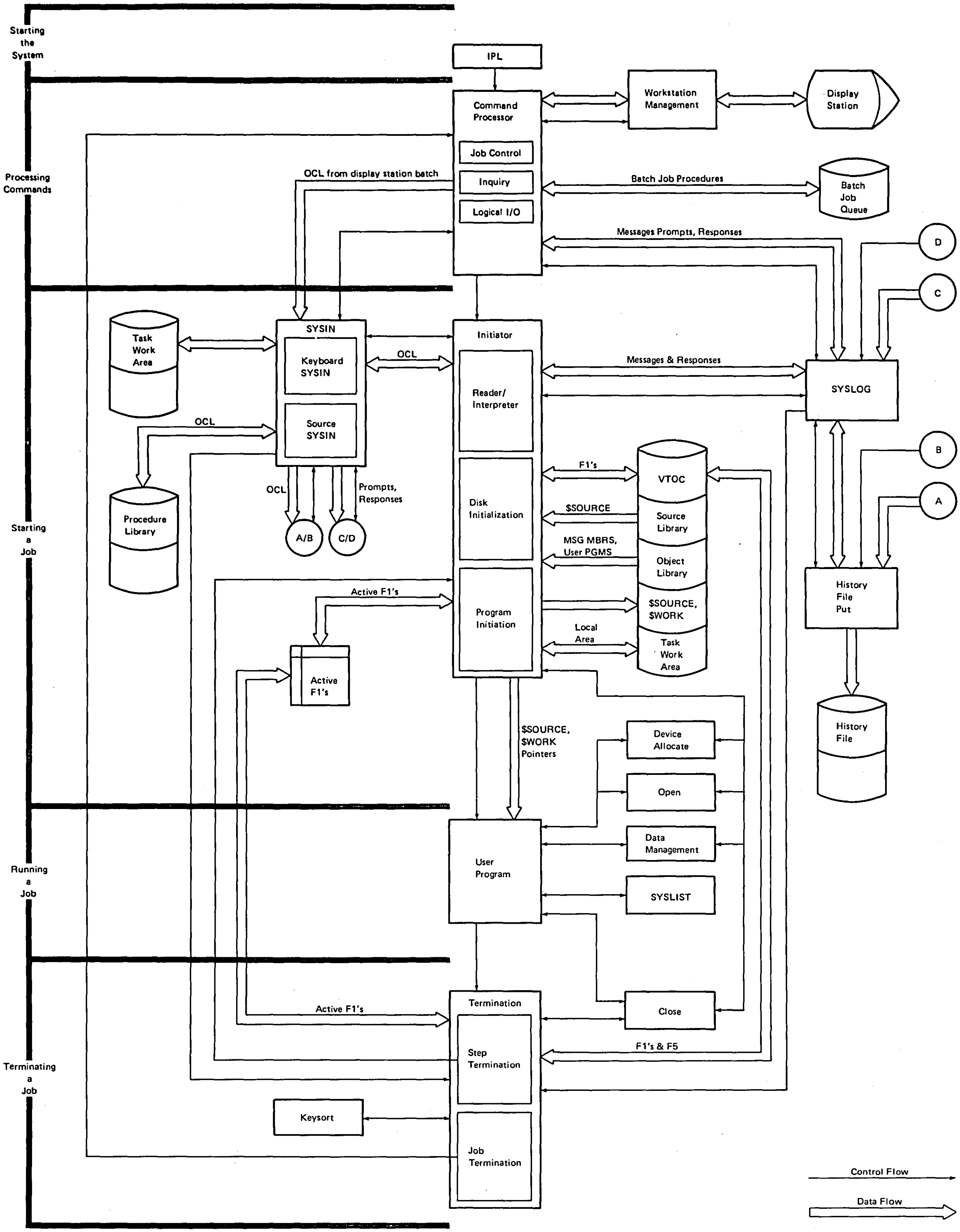


Figure 0-2. System Control Flow Overview

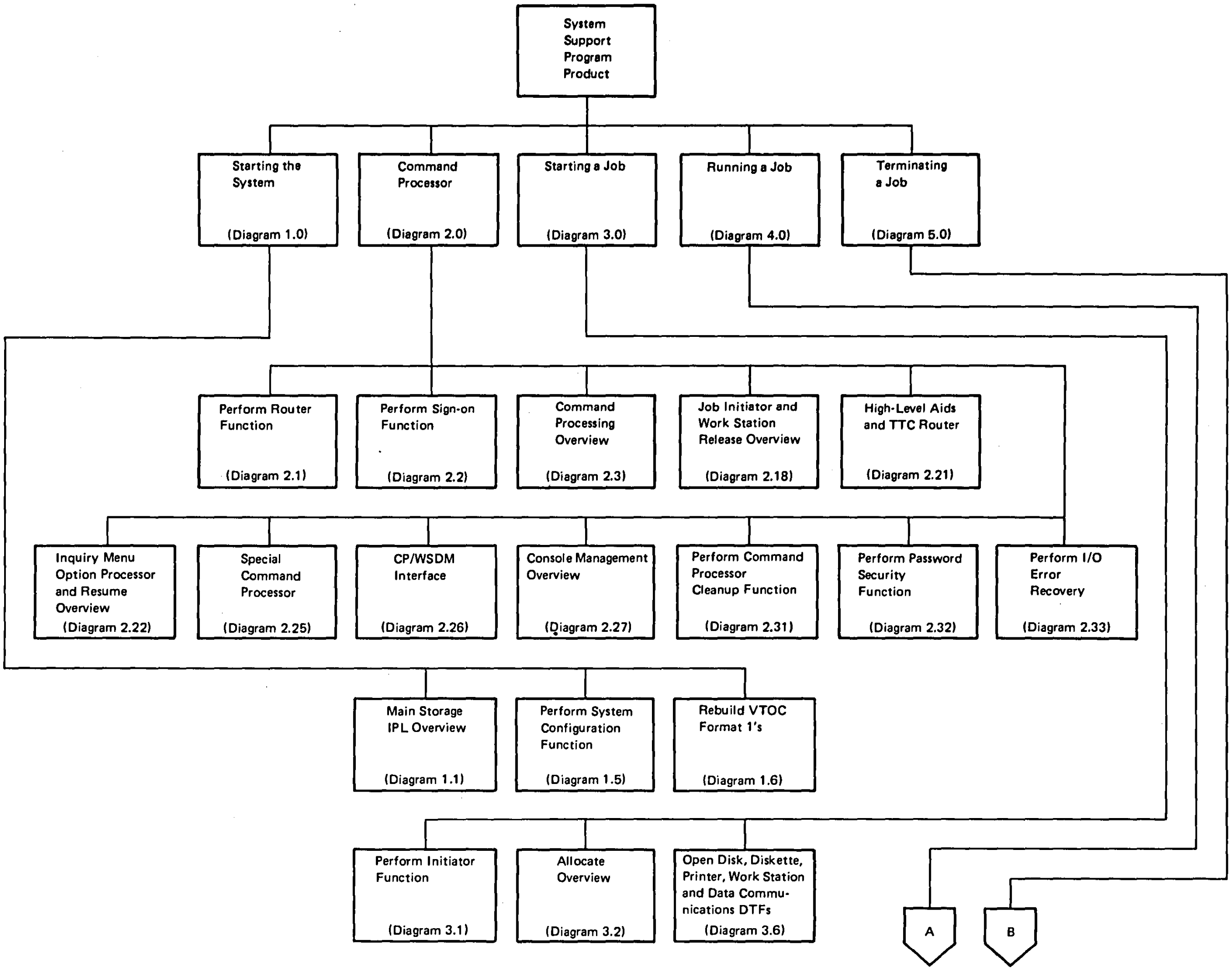

System Support Program Product
Starting the System (Diagram 1.0)
Command Processor (Diagram 2.0)
Starting a Job (Diagram 3.0)
Running a Job (Diagram 4.0)
Terminating a Job (Diagram 5.0)
Perform Router Function (Diagram 2.1)
Perform Sign-on Function (Diagram 2.2)
Command Processing Overview (Diagram 2.3)
Job Initiator and Work Station Release Overview (Diagram 2.18)
High-Level Aids and TTC Router (Diagram 2.21)
Inquiry Menu Option Processor and Resume Overview (Diagram 2.22)
Special Command Processor (Diagram 2.25)
CP/WSDM Interface (Diagram 2.26)
Console Management Overview (Diagram 2.27)
Perform Command Processor Cleanup Function (Diagram 2.31)
Perform Password Security Function (Diagram 2.32)
Perform I/O Error Recovery (Diagram 2.33)
Main Storage IPL Overview (Diagram 1.1)
Perform System Configuration Function (Diagram 1.5)
Rebuild VTOC Format 1's (Diagram 1.6)
Perform Initiator Function (Diagram 3.1)
Allocate Overview (Diagram 3.2)
Open Disk, Diskette, Printer, Work Station and Data Communications DTFs (Diagram 3.6)
A
B

Diagram 0.1 (Part 1 of 6). Functional Overview

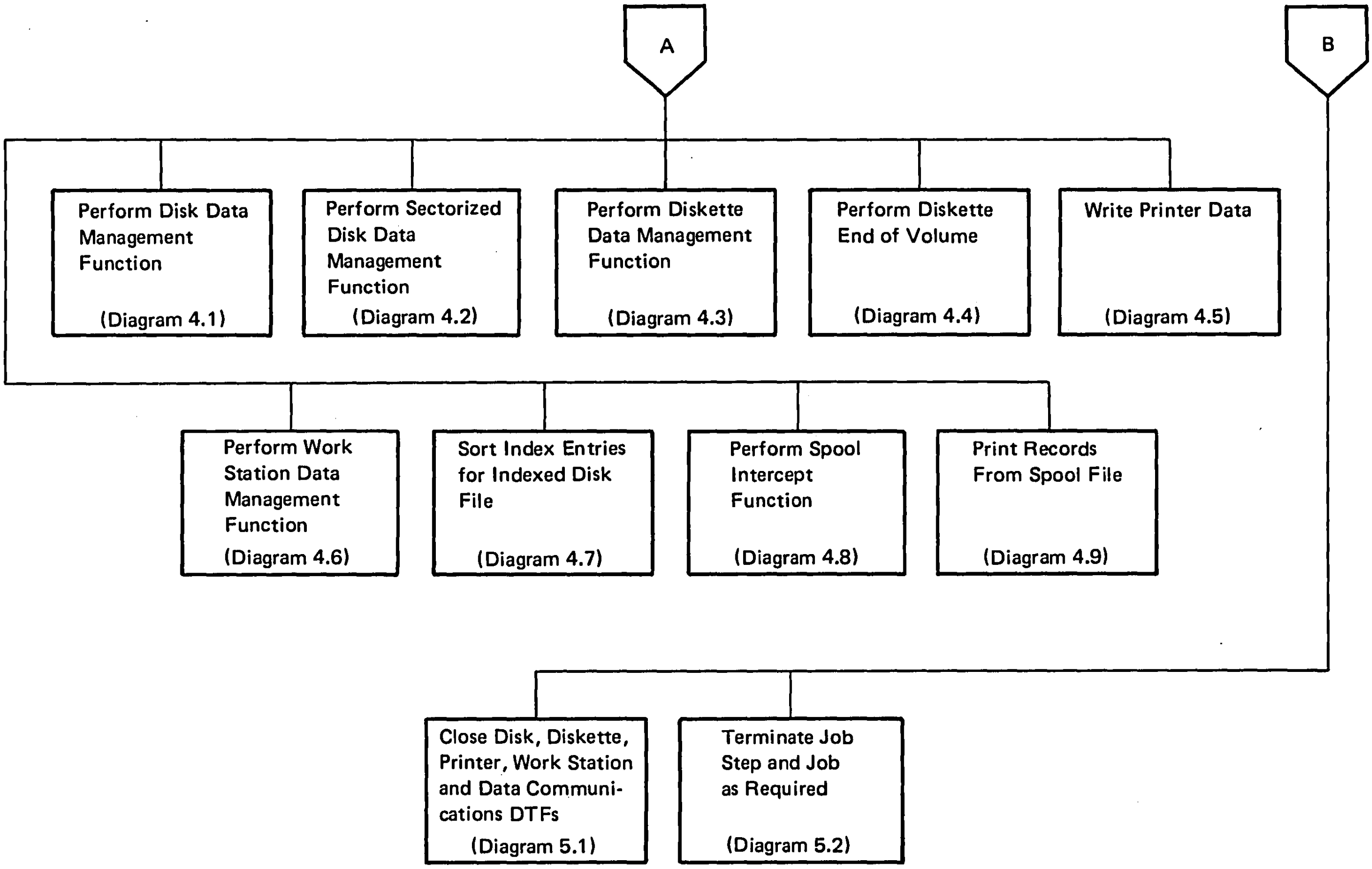

A
B
Perform Disk Data Management Function (Diagram 4.1)
Perform Sectorized Disk Data Management Function (Diagram 4.2)
Perform Diskette Data Management Function (Diagram 4.3)
Perform Diskette End of Volume (Diagram 4.4)
Write Printer Data (Diagram 4.5)
Perform Work Station Data Management Function (Diagram 4.6)
Sort Index Entries for Indexed Disk File (Diagram 4.7)
Perform Spool Intercept Function (Diagram 4.8)
Print Records From Spool File (Diagram 4.9)
Close Disk, Diskette, Printer, Work Station and Data Communications DTFs (Diagram 5.1)
Terminate Job Step and Job as Required (Diagram 5.2)

Diagram 0.1 (Part 2 of 6). Functional Overview

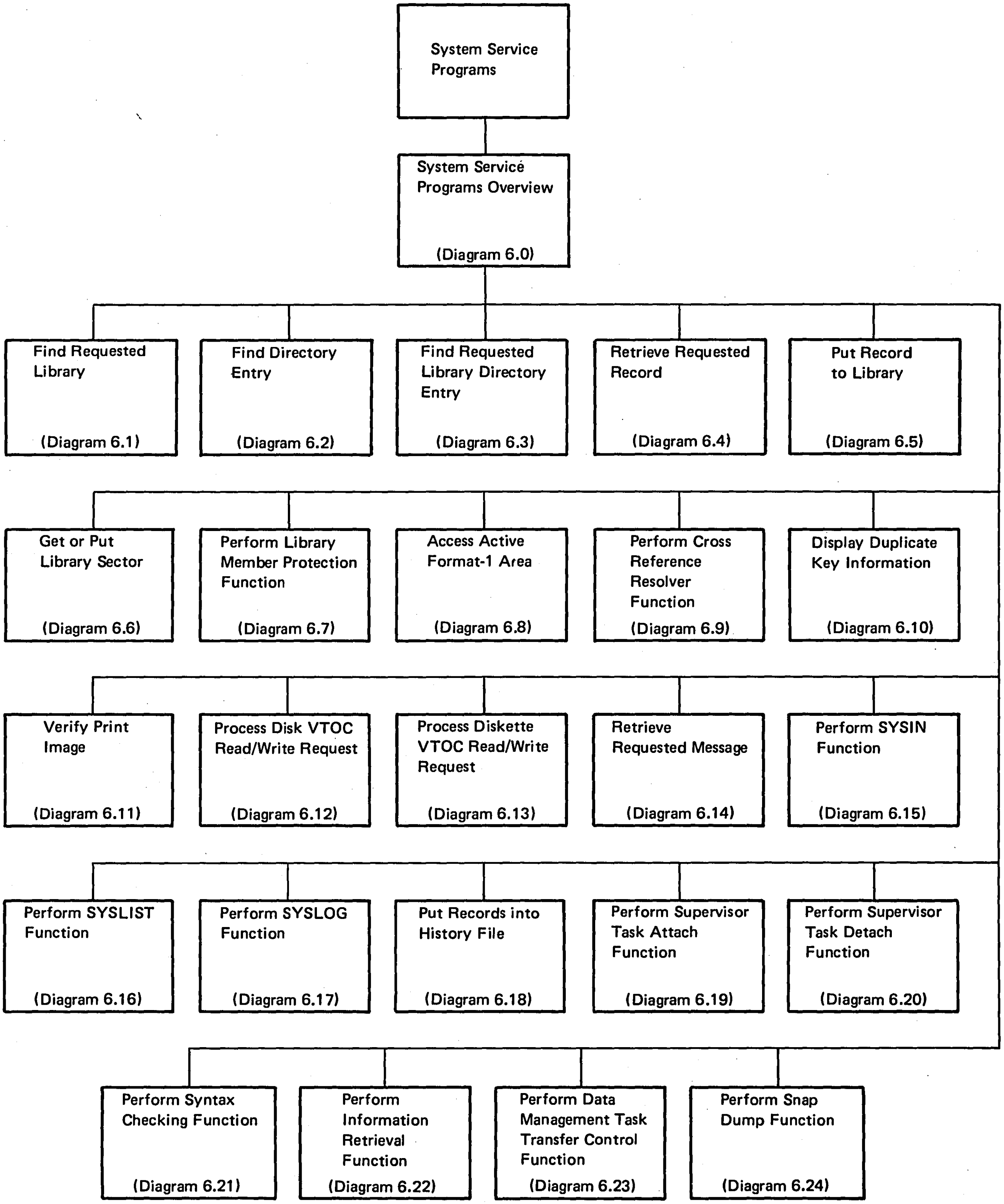

System Service Programs
System Service Programs Overview (Diagram 6.0)
Find Requested Library (Diagram 6.1)
Find Directory Entry (Diagram 6.2)
Find Requested Library Directory Entry (Diagram 6.3)
Retrieve Requested Record (Diagram 6.4)
Put Record to Library (Diagram 6.5)
Get or Put Library Sector (Diagram 6.6)
Perform Library Member Protection Function (Diagram 6.7)
Access Active Format-1 Area (Diagram 6.8)
Perform Cross Reference Resolver Function (Diagram 6.9)
Display Duplicate Key Information (Diagram 6.10)
Verify Print Image (Diagram 6.11)
Process Disk VTOC Read/Write Request (Diagram 6.12)
Process Diskette VTOC Read/Write Request (Diagram 6.13)
Retrieve Requested Message (Diagram 6.14)
Perform SYSIN Function (Diagram 6.15)
Perform SYSLIST Function (Diagram 6.16)
Perform SYSLOG Function (Diagram 6.17)
Put Records into History File (Diagram 6.18)
Perform Supervisor Task Attach Function (Diagram 6.19)
Perform Supervisor Task Detach Function (Diagram 6.20)
Perform Syntax Checking Function (Diagram 6.21)
Perform Information Retrieval Function (Diagram 6.22)
Perform Data Management Task Transfer Control Function (Diagram 6.23)
Perform Snap Dump Function (Diagram 6.24)

Diagram 0.1 (Part 3 of 6). Functional Overview

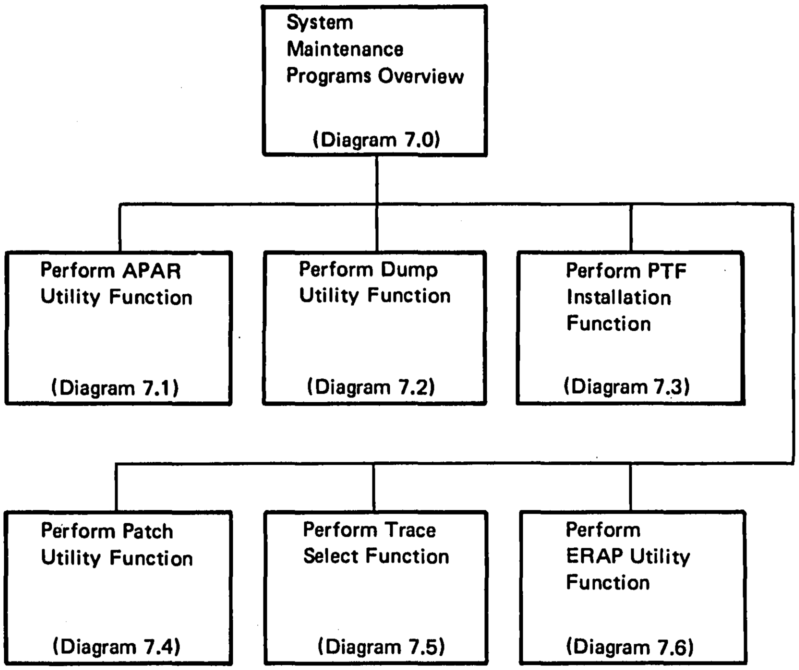

System Maintenance Programs Overview
(Diagram 7.0)
Perform APAR Utility Function
(Diagram 7.1)
Perform Dump Utility Function
(Diagram 7.2)
Perform PTF Installation Function
(Diagram 7.3)
Perform Patch Utility Function
(Diagram 7.4)
Perform Trace Select Function
(Diagram 7.5)
Perform ERAP Utility Function
(Diagram 7.6)

Diagram 0.1 (Part 4 of 6). Functional Overview

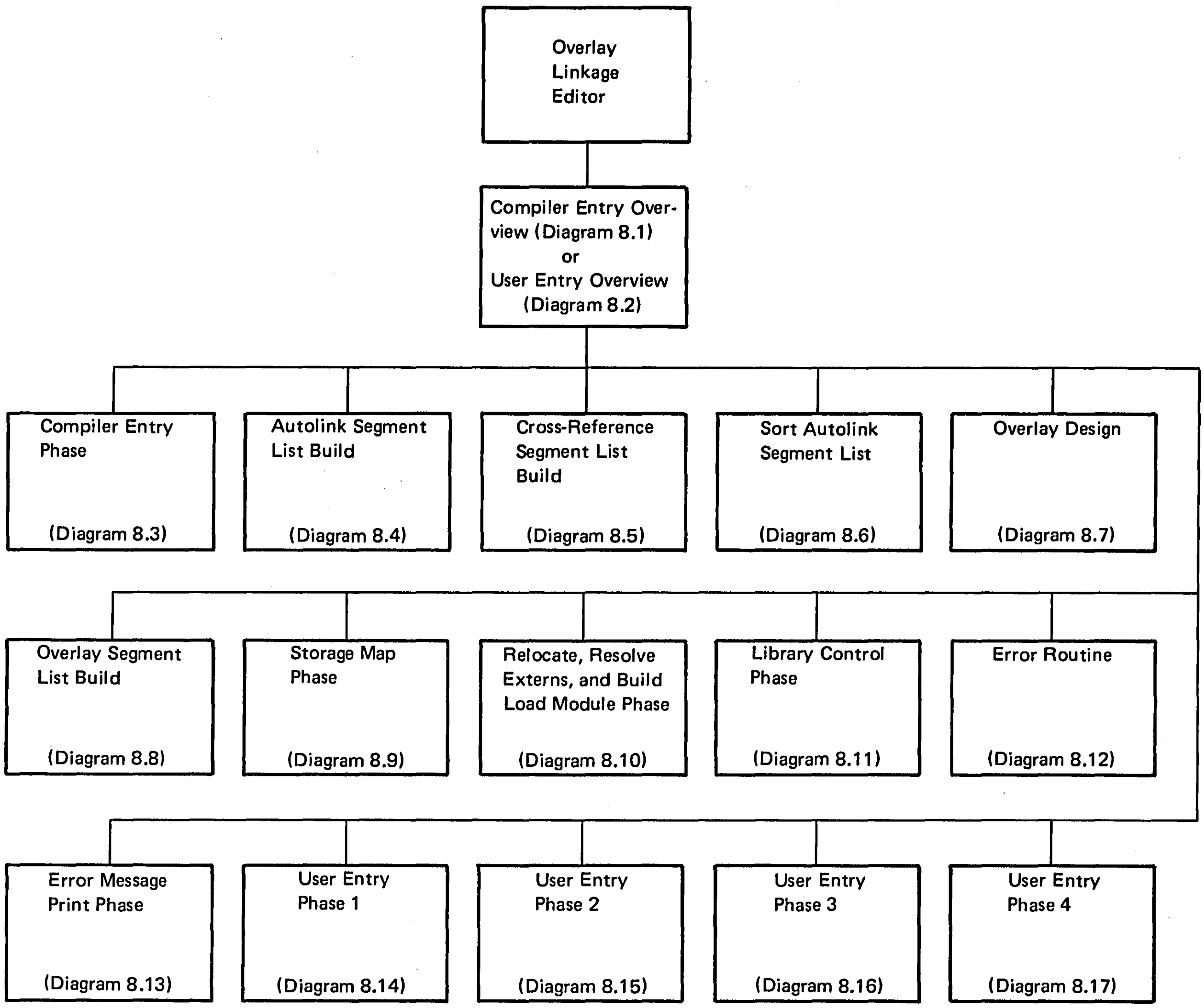

Overlay Linkage Editor
Compiler Entry Overview (Diagram 8.1) or User Entry Overview (Diagram 8.2)
Compiler Entry Phase (Diagram 8.3)
Autolink Segment List Build (Diagram 8.4)
Cross-Reference Segment List Build (Diagram 8.5)
Sort Autolink Segment List (Diagram 8.6)
Overlay Design (Diagram 8.7)
Overlay Segment List Build (Diagram 8.8)
Storage Map Phase (Diagram 8.9)
Relocate, Resolve Externs, and Build Load Module Phase (Diagram 8.10)
Library Control Phase (Diagram 8.11)
Error Routine (Diagram 8.12)
Error Message Print Phase (Diagram 8.13)
User Entry Phase 1 (Diagram 8.14)
User Entry Phase 2 (Diagram 8.15)
User Entry Phase 3 (Diagram 8.16)
User Entry Phase 4 (Diagram 8.17)

Diagram 0.1 (Part 5 of 6). Functional Overview

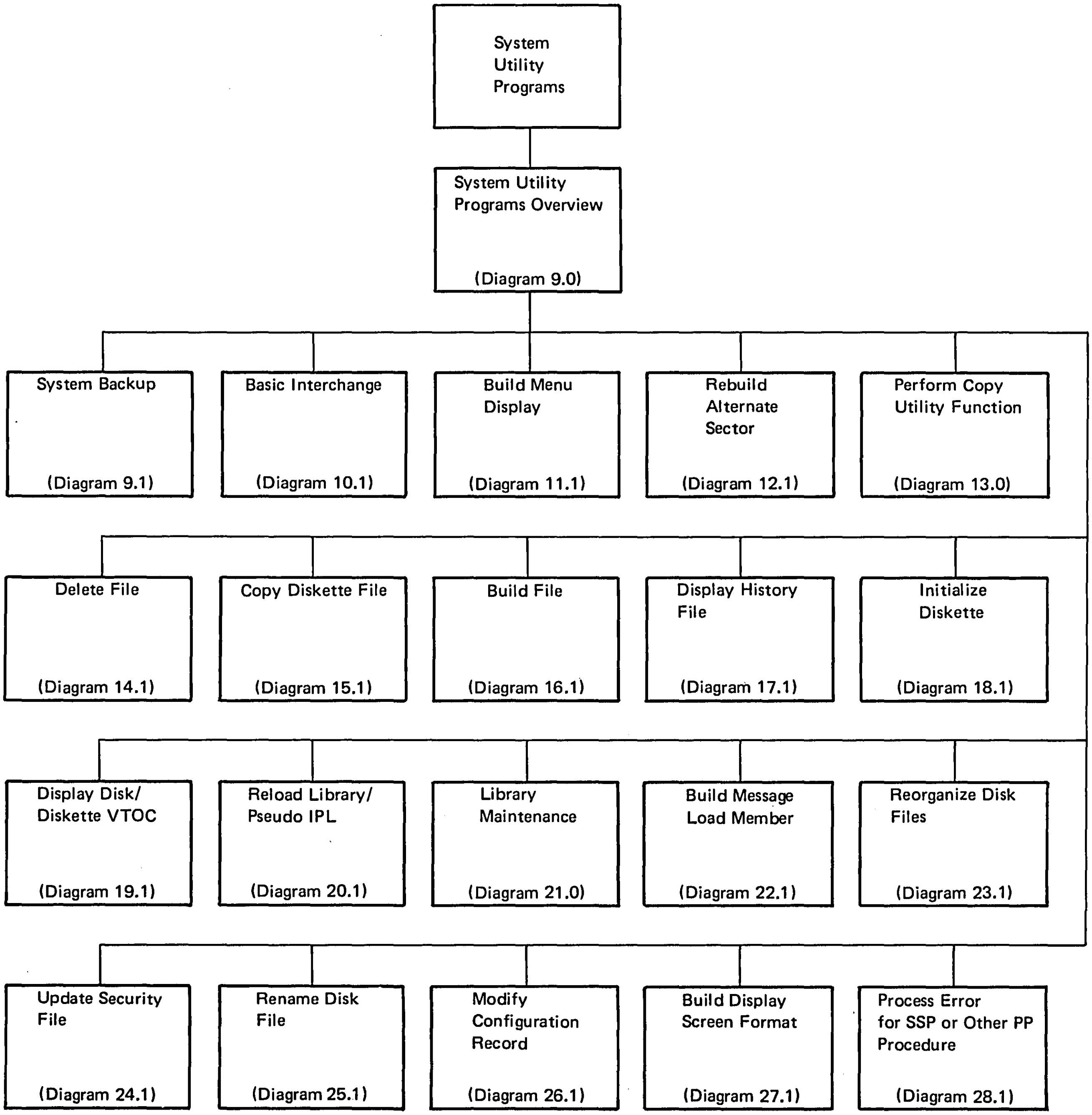

System Utility Programs
System Utility Programs Overview (Diagram 9.0)
System Backup (Diagram 9.1)
Basic Interchange (Diagram 10.1)
Build Menu Display (Diagram 11.1)
Rebuild Alternate Sector (Diagram 12.1)
Perform Copy Utility Function (Diagram 13.0)
Delete File (Diagram 14.1)
Copy Diskette File (Diagram 15.1)
Build File (Diagram 16.1)
Display History File (Diagram 17.1)
Initialize Diskette (Diagram 18.1)
Display Disk/ Diskette VTOC (Diagram 19.1)
Reload Library/ Pseudo IPL (Diagram 20.1)
Library Maintenance (Diagram 21.0)
Build Message Load Member (Diagram 22.1)
Reorganize Disk Files (Diagram 23.1)
Update Security File (Diagram 24.1)
Rename Disk File (Diagram 25.1)
Modify Configuration Record (Diagram 26.1)
Build Display Screen Format (Diagram 27.1)
Process Error for SSP or Other PP Procedure (Diagram 28.1)

Diagram 0.1 (Part 6 of 6). Functional Overview

Part 2. System Support Program Product

Introduction

The functions needed to start the system are:

- Perform initial program load (IPL)
- Perform system configuration
- Rebuild IPL format 1, if needed

MAÍN STORAGE INITIAL PROGRAM LOAD (IPL)

System/34 initial program load (IPL) consists of two parts: control storage IPL and main storage IPL.

The function of control storage IPL is to initialize main and control storage common areas to a sufficient degree to allow the control storage supervisor to operate. Control storage IPL first loads control storage and performs a basic system checkout of the processing unit and I/O functions. It then loads the control storage nucleus. After loading the work station and printer control programs, control storage IPL loads main storage IPL phase 1 (#MSNIP) and the first two sectors of the configuration record into the main storage transient area and passes control to #MSNIP.

Main storage IPL completes initialization of the System Support Program Product (SSP). Main Storage IPL consists of three phases: main storage IPL phase 1 (#MSNIP), phase 2 (#MSTWA), and phase 3 (#MSIPL). (See Figure 1-1 for main storage IPL logic flow.)

#MSNIP performs initial main storage IPL processing. Its main functions are:

- Initialize the system communication area
- Build the resident library format 1
- Determine defective main storage locations
- Initialize the transient/transfer control table
- Resolve disk addresses as needed (load and execute #MAXRF)
- Set the command processor task control block (TCB) to reflect any defective 2K main storage blocks
- Increase assign/free area size to allow for main storage allocation

#MSNIP passes control to #MSTWA.

#MSTWA performs phase 2 main storage IPL processing. Its main functions are:

- Initialize the transfer control table for resident routines
- Initialize the task work area index
- Initialize the terminal unit blocks
- Initialize the task work area for each work station
- Build the device allocate table
- Initialize command reject file data areas

Before it passes control to #MSIPL, #MSTWA updates the instruction address register (IAR) in the request block (RB) stack to pass control to the command processor resident router (#CPML) when IPL is completed.

#MSIPL controls final main storage IPL processing. The main storage IPL phase 3 function uses additional main storage IPL modules. They are: process overrides (#MSRID), process overrides (#MSOER), spool file IPL (#MSSP), input job queue IPL (#MSJQ), and main storage allocate (#MSSQS). The primary functions of main storage IPL phase 3 are:

- Perform main storage IPL sign-on
- Process override information if OVERRIDE-YES is entered on the IPL sign-on display (#MSRID, #MSOER)
- Initialize the system print spool function if print spool is supported (#MSSP)

- Initialize the job queue if job queue is supported (#MSJQ)

- Build the assign/free area (#MSSQS)

- Build the user main storage area (#MSSQS)

- Enable all system terminals

#MSIPL then calls the supervisor task attach transient (#SVAT) to attach a TCB to run file rebuild (#MSBLD). Control eventually passes to #CPML.

SYSTEM CONFIGURATION ($CNFIG)

System configuration is performed when the system is initially installed or any time a system or feature change requires reconfiguration. The configuration information is saved in the configuration records. (See the *Data Areas Handbook* for a description of the configuration records.) The system user may override certain configuration options at IPL (see #MSIPL) or alter the configuration of each work station by using $SETCF.

System configuration is initiated with the CNFIGSSP command following the initial RELOAD of the base system support program to disk. (See *IBM System/34 Program Product Installation and Modification Reference Manual,* SC21-7689, for more information about the CNFIGSSP procedure.)

CNFIGSSP loads and runs the system configuration program ($CNFIG). The main functions performed by $CNFIG are:

- Read, modify, and write system configuration records.

- Set values/parameters in the configuration records based on operator responses to configuration prompts.

- Validate the operator's input values and ensure system operation.

- Set UPSI switches to control CNFIGSSP procedure flow.

$CNFIG validates the operator's responses to prompts and updates the appropriate configuration record after each configuration display is presented. After all operator responses are processed, $CNFIG returns the updated configuration records to disk and passes control to the end-of-job transient ($EOJ).

REBUILD VTOC FORMAT 1'S

The rebuild VTOC format 1's routine (#MSBLD) examines each format 1 in the disk VTOC to ensure that the pointers accurately reflect the status of the data set contents. The routine also checks the format 5 to ensure that disk reorganization ($PACK or $FREE) was not previously interrupted.

#MSBLD is called by main storage IPL phase 3 (#MSIPL). The system operator may request that #MSBLD not be run. If run, the system operator directs #MSBLD to perform one of the following:

- Delete all files in error

- Examine old files, as well as new

- Prompt on each file in error for retention or deletion

Method of Operation

This section contains functional diagrams for routines needed to start the system. They are:

- Main storage IPL
- System configuration
- Rebuild VTOC format 1

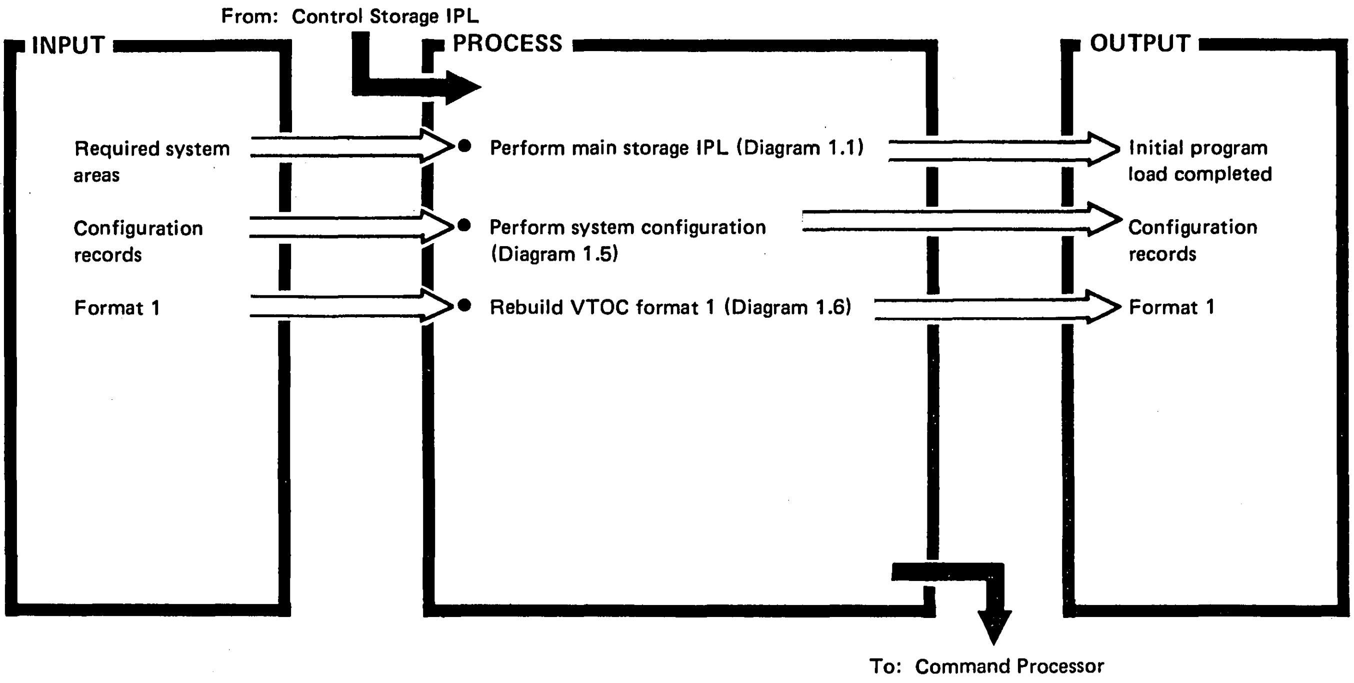

From: Control Storage IPL
INPUT
PROCESS
OUTPUT
Required system areas
Perform main storage IPL (Diagram 1.1)
Initial program load completed
Configuration records
Perform system configuration (Diagram 1.5)
Configuration records
Format 1
Rebuild VTOC format 1 (Diagram 1.6)
Format 1
To: Command Processor

Diagram 1.0. Overview of Starting the System

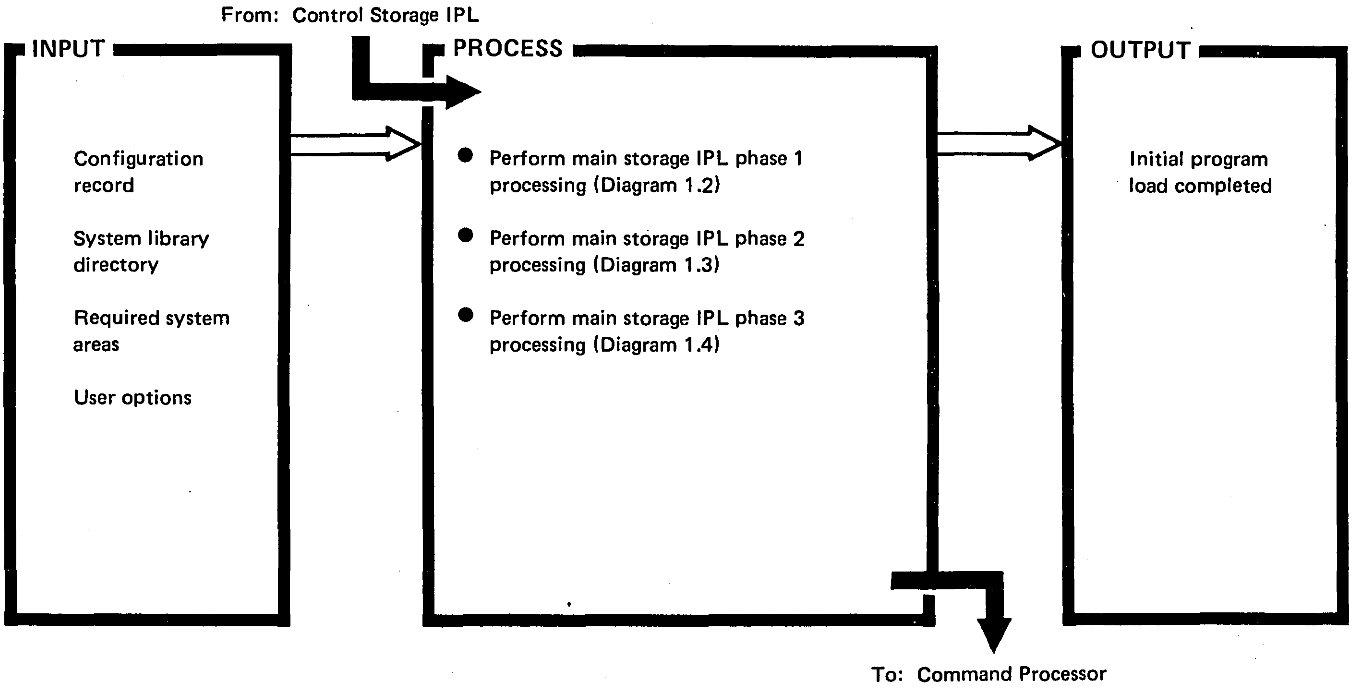
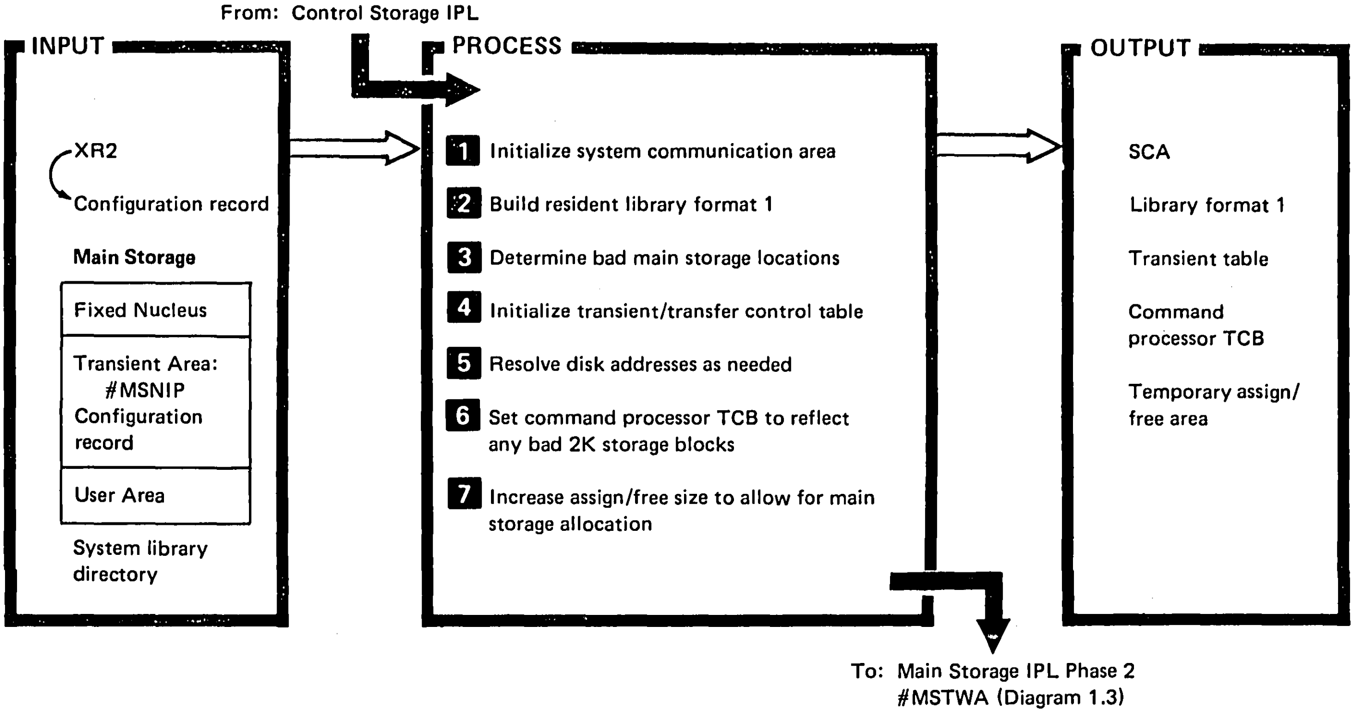

From: Control Storage IPL
INPUT
PROCESS
OUTPUT
Configuration record
System library directory
Required system areas
User options
Perform main storage IPL phase 1 processing (Diagram 1.2)
Perform main storage IPL phase 2 processing (Diagram 1.3)
Perform main storage IPL phase 3 processing (Diagram 1.4)
Initial program load completed
To: Command Processor

Diagram 1.1. Overview of Main Storage IPL



From: Control Storage IPL

INPUT

XR2

Configuration record

Main Storage

Fixed Nucleus
Transient Area: #MSNIP Configuration record
User Area

System library directory

PROCESS

1 Initialize system communication area

2 Build resident library format 1

3 Determine bad main storage locations

4 Initialize transient/transfer control table

5 Resolve disk addresses as needed

6 Set command processor TCB to reflect any bad 2K storage blocks

7 Increase assign/free size to allow for main storage allocation

OUTPUT

SCA

Library format 1

Transient table

Command processor TCB

Temporary assign/ free area

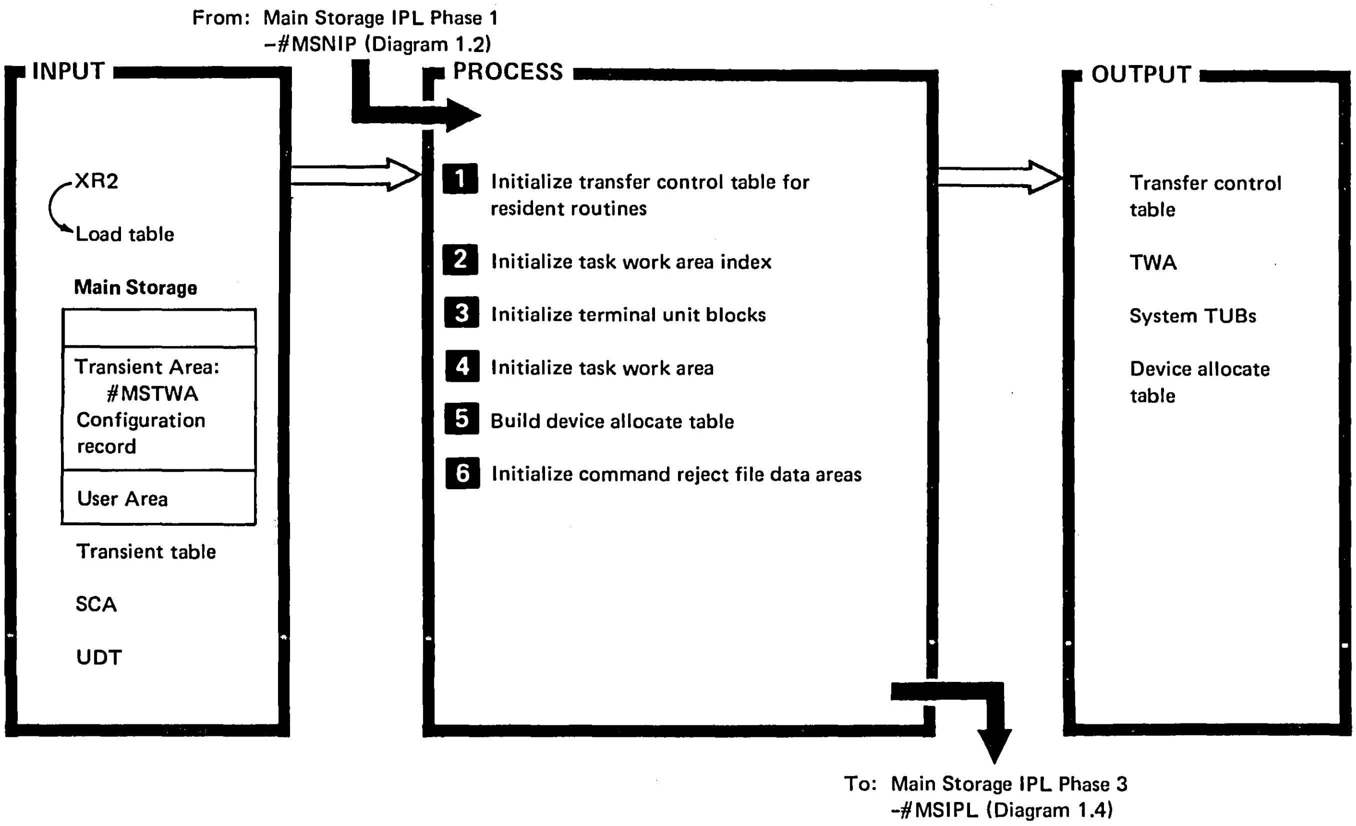
To: Main Storage IPL Phase 2 #MSTWA (Diagram 1.3)

DESCRIPTION	MODULE/ ROUTINE
1 Initialize system communication area (SCA) from configuration record information:	#MSNIP
● Task work area (TWA) address.	Disk IOS
● TWA size.	#MSNIP
● Disk VTOC address.	
● Disk VTOC size.	
● Diskette VTOC work area disk address.	
● Diskette VTOC work area size.	
● Configuration record start address.	
● History file start address.	
● History file size.	
● Release and modification level.	
● Control storage configuration size.	
● DUMP indicator.	
● System configuration bytes.	
▶ ● Communications configuration from UDT.	Disk IOS ◀
Resolve current history file pointer:	#MSNIP
● Scan history file looking for file ID (X'FE') as start of history file sector.	Disk IOS
● If history file current pointer found, put disk address in SCA (SCAHFCUR).	#MSNIP
● If history file current pointer not found, set history file error flag in SCA (SCAHFERR).	
2 Read system VTOC, first sector, to get library format 1.	Disk IOS
Move library format 1 to nucleus.	#MSNIP
Save spool buffer size.	

Diagram 1.2 (Part 1 of 2). Perform Main Storage IPL Phase 1 Processing

DESCRIPTION	MODULE/ ROUTINE
3 Count number of bad 2K blocks of main storage. Assign save area for bad 2K block numbers. Move bad 2K blocks to save area. **4** Move dump SVC address into transient/transfer control table for each table entry.	#MSNIP
Read in block (one track) of library directory sectors.	Disk IOS
Look for name table entry in library directory. If name table entry found: ● Place SSN in transient table if necessary. ● Place SS in SCA (message member) if necessary. ● Build loader parameter list if necessary. If name table entry not found: ● Determine whether it is required by SSP. ● If required, call dump main storage transient task (9 control storage function) to abend system (MIC 249).	#MSNIP
5 Resolve disk addresses by executing cross reference resolver.	#MAXRF
6 Determine whether any bad 2K storage blocks exist (SCA2KBAD). Find task control block (TCB) address translate registers (ATRs). Flag bad ATRs. Set good ATR numbers in TCB. **7** Allocate temporary assign/free space for duration of main storage IPL. Load and pass control to IPL phase 2 (#MSTWA).	#MSNIP

Diagram 1.2 (Part 2 of 2). Perform Main Storage IPL Phase 1 Processing



From: Main Storage IPL Phase 1 -#MSNIP (Diagram 1.2)

INPUT

XR2

Load table

Main Storage

Transient Area: #MSTWA Configuration record

User Area

Transient table

SCA

UDT

PROCESS

1 Initialize transfer control table for resident routines

2 Initialize task work area index

3 Initialize terminal unit blocks

4 Initialize task work area

5 Build device allocate table

6 Initialize command reject file data areas

OUTPUT

Transfer control table

TWA

System TUBs

Device allocate table

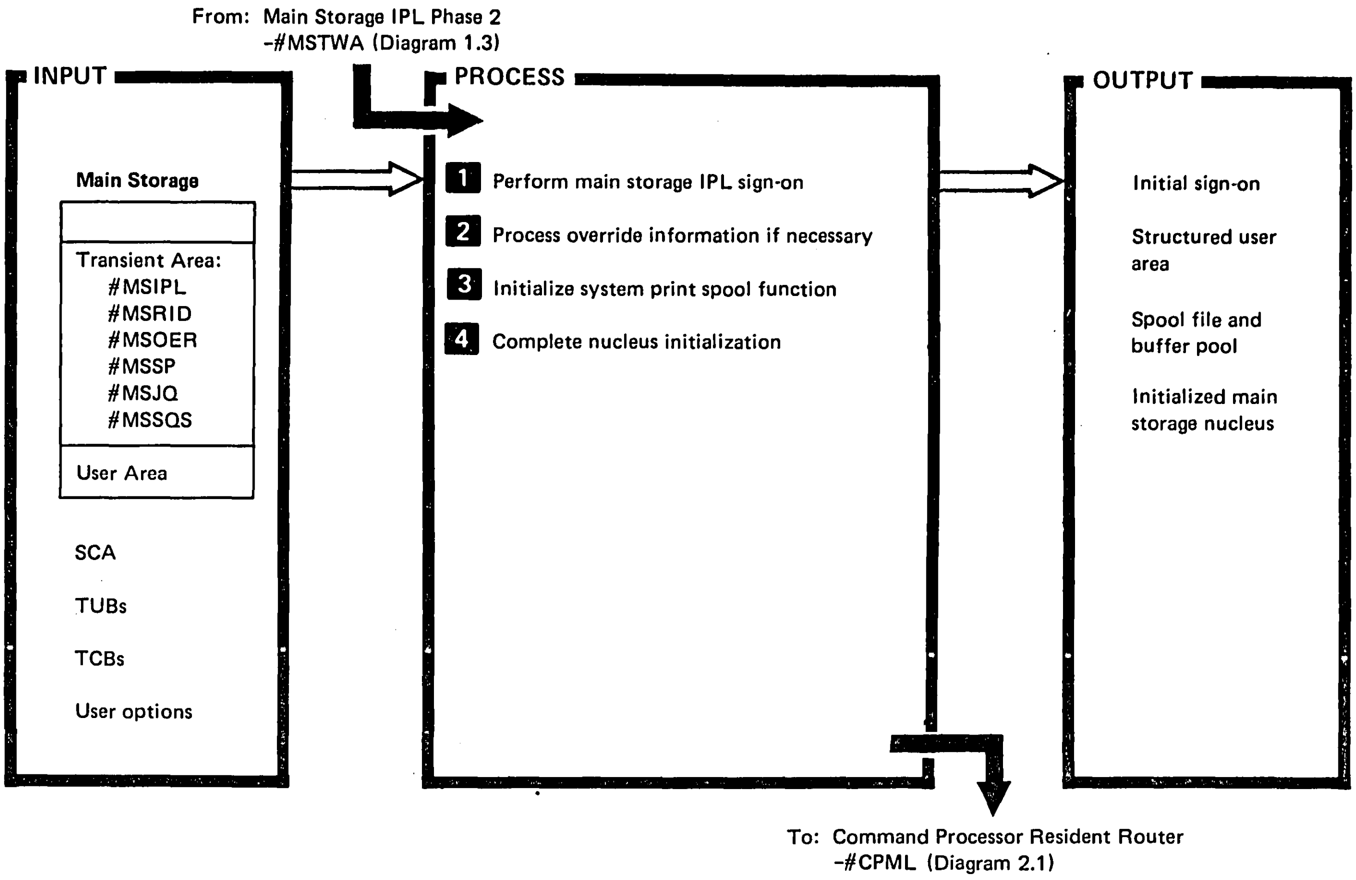
To: Main Storage IPL Phase 3 -#MSIPL (Diagram 1.4)

DESCRIPTION	MODULE/ ROUTINE
1 Scan load table built during directory scan by #MSNIP.	#MSTWA
Load resident routines: ● Calculate next load address. ● If request indicator byte (RIB) given, put its address in transfer control table. ● Call main storage relocating loader (control storage function) to load the resident routine.	
2 Find space for command processor work area.	
Build command processor image matrix and order index.	
Reset task work area index to all available work space.	Disk IOS
Reset task work area (first track) to binary zeros.	
Find space for temporary work station queue (to be used during IPL).	#MSTWA
Allocate space from TWA for command reject file.	
If no space available for work station queues, call dump main storage/terminate task routine (control storage function) to eliminate IPL procedure.	

Diagram 1.3 (Part 1 of 2). Perform Main Storage IPL Phase 2 Processing

DESCRIPTION	MODULE/ ROUTINE
3 Read terminal information blocks into work area.	Disk IOS
Initialize terminal unit blocks (TUB) based on system configuration record information: ● Set work station ID. ● Initialize common TUB fields. ● Set system printer TUB address in SCA. ● Set task work area address.	#MSTWA
4 Allocate work space in task work area.	
Write work station configuration record to disk.	Disk IOS
Initialize local area of task work area.	#MSTWA
5 Build entries for diskette from unit definition table (UDT) information.	
Build communication entries from communication configuration record information.	
6 Initialize command reject file data areas.	
Build and queue error recovery blocks (ERB) for disk and diskette.	
Load and pass control to main storage IPL phase 3 (#MSIPL).	

Diagram 1.3 (Part 2 of 2). Perform Main Storage IPL Phase 2 Processing



From: Main Storage IPL Phase 2
-#MSTWA (Diagram 1.3)

INPUT

Main Storage

Transient Area:
#MSIPL
#MSRID
#MSOER
#MSSP
#MSJQ
#MSSQS

User Area

SCA

TUBs

TCBs

User options

PROCESS

1 Perform main storage IPL sign-on

2 Process override information if necessary

3 Initialize system print spool function

4 Complete nucleus initialization

OUTPUT

Initial sign-on

Structured user area

Spool file and buffer pool

Initialized main storage nucleus

To: Command Processor Resident Router
-#CPML (Diagram 2.1)

DESCRIPTION	MODULE/ ROUTINE
1 Post task control block (TCB) for request-enter sign-on. Search terminal unit block (TUB) chain for system and alternate consoles. Check system console for errors. If no system console errors, simulate REQ-ENT request. Wait for console input.	#MSIPL
If request-enter is from console, process REQ-ENT request.	#CPTC
If request-enter is not from console: ● Build assign accept/input parameter list.	#MSIPL
● Accept input from console.	#WDDA
● If alternate console, enable request-enter and call request enter transient (#CPTC). ● If master console, call sign-on transient (#CPON) to process sign-on information.	#MSIPL
● Prompt for file rebuild options. Check for override request-entered at sign-on.	#MSRID
2 Build override format index:	#MSRID
● Read override format index from disk.	Disk IOS
● Build in-core format index.	#MSRID

Diagram 1.4 (Part 1 of 6). Perform Main Storage IPL Phase 3 Processing

DESCRIPTION	MODULE/ ROUTINE
Prompt for general system parameters:	
● Issue invite to display unit.	#WDDA
▶ ● Build output work station data management parameter list.	#MSRID ◀
● Prompt for: – Date. – Single program mode. – Console status. – Command language.	#DWDM
● Build input work station data management parameter list.	#MSRID
● Issue call to work station I/O to accept input.	#WDDA
● Set appropriate system communication area (SCA) indicators for answers to general system prompts.	#MSRID
Prompt for work station data management options (transient or resident).	
Call transient #MSOER to prompt for job queue and spool parameters.	
Ensure that job queue is configured before issuing prompt for job queue parameters.	#MSOER
Build output parameter list for work station data management to prompt for: ● Job queue size. ● Job queue reformat. ● Job queue start.	
Display prompts.	#DWDM
Build work station data management parameter list to accept answers to prompts.	#MSOER
Issue call to work station I/O for accept-input.	#DWDM
Check job queue size.	#MSOER
Set appropriate SCA indicators for answers to job queue prompts.	
Display prompt for: ● Cancel spool? ● Cancel job queue? ● Delete spool file? ● Delete job queue?	#DWDM
Build parameter list to accept answer to prompt.	#MSOER
Issue call to work station I/O for accept-input.	#DWDM
If cancel spool – yes, return to #MSRID	#MSOER
If cancel spool – no: ● Build work station data management parameter list to prompt for: – Resident spool writer. – Spool writer priority. – Intercept buffer size. – Spool writer buffer size.	
● Display prompts.	#DWDM
● Build input parameter list.	#MSOER
▶ ● Issue call to work station I/O for accept-input.	#DWDM ◀
● Ensure legitimate answers to prompts and set appropriate system indicators.	#MSOER
● Build work station data management parameter list to prompt for print spooling disk parameters: – Spool file size. – Reformat spool file at IPL. – Disk A or disk B reference.	

Diagram 1.4 (Part 2 of 6). Perform Main Storage IPL Phase 3 Processing

DESCRIPTION	MODULE/ ROUTINE
● Issue a call to work station I/O to display prompts.	#DWDM
● Build an input parameter list.	#MSRID
● Issue a call to work station I/O for accept-input.	#DWDM
● Ensure legitimate answers to prompts and set appropriate system indicators. Return to #MSRID	#MSOER
Build a work station data management parameter list to prompt for performance parameters: ● Work station queue space size. ● Assign/free request size. ● Trace buffer size.	#MSRID
Issue call to work station I/O to issue prompts.	#DWDM
Build input parameter list.	#MSRID
Issue call to work station I/O for accept-input.	#DWDM
Ensure legitimate answers to prompts and that enough main storage space exists. Set appropriate system indicators for prompt answers. Return control to #MSIPL.	#MSRID
Build second TUB for command processor. Log control storage processor errors. Put system date in configuration record:	#MSIPL
● Read configuration record from disk.	Disk IOS
▶ ● Update configuration record.	#MSIPL ◀
● Write configuration record back to disk.	Disk IOS
Change command language for command processor to native/English if necessary: ● Find command processor router routine – #CPRT (command processor load module). ● #CPRT contains commands to be changed. Start control address of #CPRT indicates start of command table. Each entry is 14 bytes long. ● Retrieve commands from the message member (##MSG2). ● Update commands as required. **3** Ensure that spool is on system (SCAMSPOL) and job queue is on system (SCAMJOBQ). Calculate size of spool buffer pool. Assign spool buffer pool space. Save pool size and writer buffer size. Set up spool intercept function:	#MSIPL
● Find spool intercept routine (#SPINT).	#MASFN
● Assign space to load #SPINT. ● Load #SPINT using main storage relocating loader – SVC 52 (control storage function). **4** Call main storage allocate transient (#MSSQS) to resolve main storage areas.	#MSIPL
Assign storage for work station queue space: ● Free work station queue space assigned for IPL sign-on. ● Assign work station queue space for system operation. Build system queue space (control storage microcode function).	#MSSQS

Diagram 1.4 (Part 3 of 6). Perform Main Storage IPL Phase 3 Processing

DESCRIPTION	MODULE/ ROUTINE
Build user main storage area: ● Use last FQE accessed to find first 2K block of user main storage following assign/free area. ● Chain user main storage 2K blocks together, excluding any bad 2K blocks. ● Clear 2K blocks to binary zeros as blocks are added to chain. ● Check for the following conditions: — Any bad 2K main storage blocks (SCA2KBAD). — Insufficient region size (SCADRGSZ). — Maximum nucleus size exceeded. ● Build error message parameter list, including error message identification code (MIC).	#MSSQS
● Issue error message.	#CMCU
Return control to #MSIPL.	#MSSQS
Put up sign-on display screen: ● Set IPL sign on code (CPCODE). ● Display sign on screen.	#CPON
Clean up any outstanding invites on other terminals: ● Examine TUBs. ● If work station TUB, enable system request. ● If console, signed on console and issue invite.	#MSIPL
● Stop outstanding invites.	#CPTC
Get JCB space (JCB will be used to attach file rebuild (#MSBLD)). Build attach parameter list. Indicate file rebuild called (SCAMBLD). Indicate IPL sign-on complete (SCAMIPL). Call attach transient (#SVAT) to attach #MSBLD. Perform command processor resident functions (#CPML) until #MSBLD is finished. Call spool file IPL (#MSSP) to allocate and format spool file and spool buffer pool.	#MSIPL
If spool not supported (SCADSSPR), go to 4 A. If delete or cancel request (SCADSSPR):	#MSSP
● Read VTOC to find file to delete.	#CSVF
▶ ● Build file specification block (FSB).	#CAS1 ◀
● Delete file.	#CAD1
● Set flag to 00 (SCADSSPR). ● Go to 4 A. Check for existing #SPOOL1 file: ● Build disk VTOC read/write parameter list.	#MSSP
● Read disk VTOC. If #SPOOL1 file found:	#CSVF
● Read #SPOOL1 master index from disk.	#DWDM
● If no entries on print queue or request to delete file (SCAMCRAN):	#MSSP
— Read VTOC to find file to delete.	#CSVF
— Build file specification block (FSB).	#CAS1
— Delete file.	#CAD1

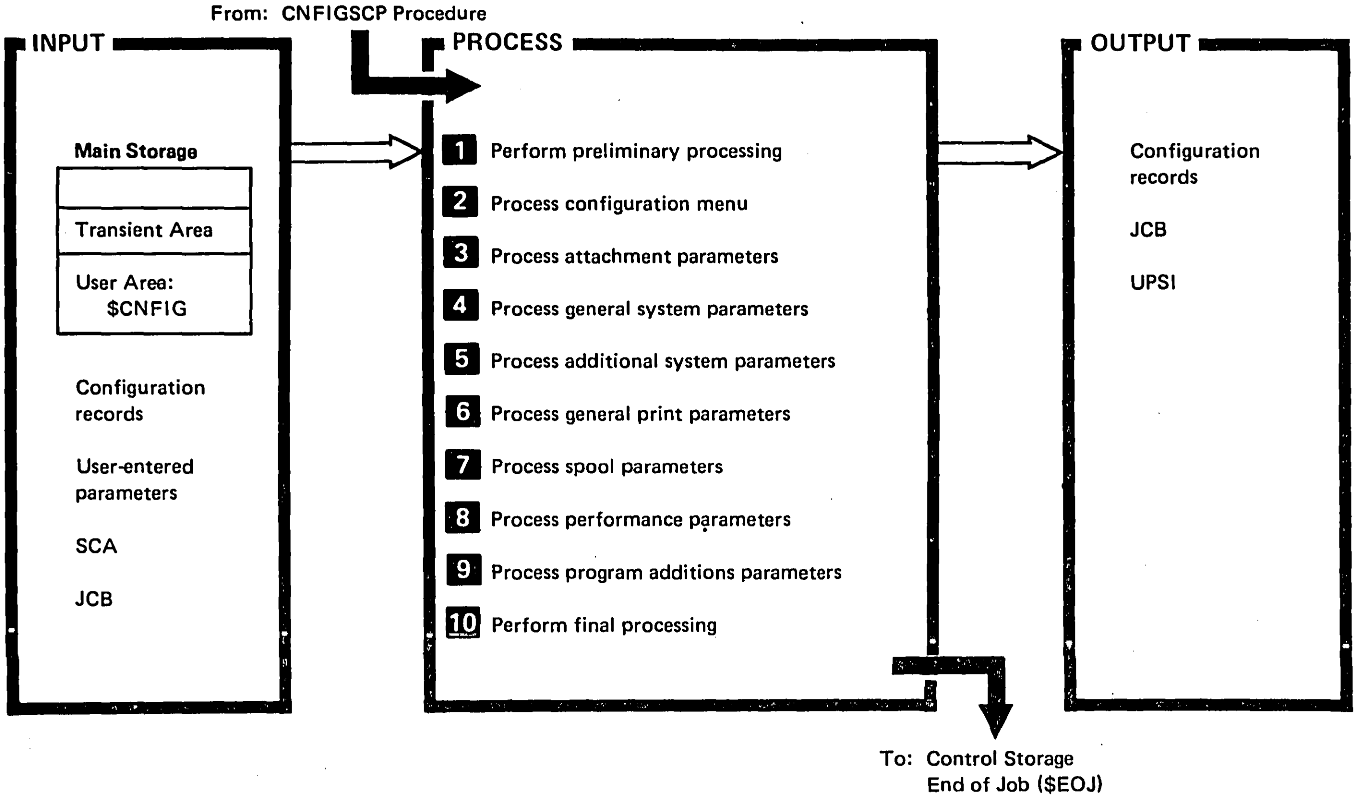
Diagram 1.4 (Part 4 of 6). Perform Main Storage IPL Phase 3 Processing

DESCRIPTION	MODULE/ ROUTINE
Allocate primary print spool file:	#MSSP
● Set up special allocate DTF to allocate space for size given in IPLWKSIZ.	
● Allocate file space.	#CAS1
● If file space not available, decrease file size by six blocks and attempt to allocate. If minimum size (12 blocks) is not available, issue message, SPOOL NOT SUPPORTED THIS IPL, and go to 4 A.	#CMCU
Format print spool file:	
● Calculate maximum file size.	#MSSP
● Calculate size of each extent.	
● Calculate index size.	
● Set up fields to indicate number of index entries needed for primary file and extents.	
● Format master index and chain index entries together.	
● Write index entries out to print spool file.	Disk IOS
● Write master index out to first sector of primary spool file.	
Set address of #SPOOL1 in SYSCOM.	#MSSP
Update master index:	
▶ ● Find extents (#SPOOL2-6).	#CSVF ◀
● Find highest spool jobname.	#SPQMG
● Update master index with new information.	#MSSP
● Write master index back to primary spool file.	Disk IOS
Format buffer pool by calculating number of intercept buffers.	#MSSP
Look for job-in-process or active bits on in spool queue and set off if found.	#SPQMG
A Call input job queue IPL routine (#MSJQ) to format input job queue.	#MSSP
If job queue not supported (SCADSSJQ), return to #MSIPL.	#MSJQ
If delete or cancel request (SCADSSJQ):	
● Build file specification block (FSB).	#CAS1
● Deallocate existing #JOBQ file.	#CAD1
● Return to #MSIPL.	#MSJQ
Check for existing #JOBQ file:	
● Build VTOC read/write parameter list.	
● Read disk VTOC.	#CSVF
▶ If #JOBQ found:	#MSJQ ◀
● Read first sector of #JOBQ file to check for data.	Disk IOS
● If no data exists or reformat request:	#MSJQ
– Build FSB.	#CAS1
– Deallocate JOBQ file.	#CAD1
Allocate #JOBQ file:	#MSJQ
● Build DTF with #JOBQ file information.	
● Allocate #JOBQ file.	#CAS1
If not enough space available for #JOBQ file requested:	#MSJQ
● Decrease file size by 2 blocks and attempt to allocate.	
● If minimum size (4 blocks) is not available, issue message, INPUT JOBS NOT SUPPORTED THIS IPL, and return to #MSIPL.	#CMCU

Diagram 1.4 (Part 5 of 6). Perform Main Storage IPL Phase 3 Processing

DESCRIPTION	MODULE/ ROUTINE
Format JOBQ file entries: ● Set up IOB with attributes for #JOBQ file.	#MSJQ
● Write formatted sectors to disk.	Disk IOS
Update SCA with #JOBQ file address. Return to #MSIPL.	#MSJQ
If stop system command, call #CPTC to	#MSIPL
set IPL processing complete (SCAMIPLC).	#CPTC
Pass control to #CPML.	#MSIPL

Diagram 1.4 (Part 6 of 6). Perform Main Storage IPL Phase 3 Processing



From: CNFIGSCP Procedure

INPUT

Main Storage

Transient Area

User Area:
$CNFIG

Configuration records

User-entered parameters

SCA

JCB

PROCESS

1 Perform preliminary processing

2 Process configuration menu

3 Process attachment parameters

4 Process general system parameters

5 Process additional system parameters

6 Process general print parameters

7 Process spool parameters

8 Process performance parameters

9 Process program additions parameters

10 Perform final processing

OUTPUT

Configuration records

JCB

UPSI

To: Control Storage
End of Job ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Prepare disk IOB to read configuration records from disk.	$CNFIG
Perform read operation.	Disk IOS
Open work station data management (WSDM) DTF.	#DMOP
2 Issue PUT to display configuration menu (display 1).	$CNFIG
Display configuration menu.	#DWDM
Issue GET to accept operator replies.	$CNFIG
▶ Retrieve operator response:	#DWDM ◀
If option 1 (perform system configuration), set off display only switch (CNDSP).	$CNFIG
If option 2 (perform limited configuration with reset), set off display only switch and go to 4.	
If option 3 (perform limited configuration without reset), set off display only switch and go to 4.	
If option 4 (perform attachments change only), set off display only switch and go to 3.	
If option 5 (only display configuration parameters), set on display only switch.	
If response was not option 1 through 5, prepare operator prompt (?) and return to 2 to redisplay configuration menu.	

Diagram 1.5 (Part 1 of 5). Perform System Configuration Processing ($CNFIG)

DESCRIPTION	MODULE/ ROUTINE
3 Prepare attachment parameters display (display 2). A Issue PUT to output display 2.	$CNFIG
Display attachment parameters screen consisting of: ● Unit address. ● Device type. ● Logical ID. ● Attribute. ● Default PRT.	#DWDM
Prepare WSDM to retrieve operator replies.	$CNFIG
Retrieve operator replies from screen. If display only request (CNDSP), go to 4.	#DWDM
Validate operator replies: ● Confirm unit addresses and device types. ● Check logical ID format. ● Check for duplicate ID. ● Check attribute format. ● Check attribute. ● Check default printer logical ID format. ● Check default printer assignment. If errors detected, prepare operator error message and return to 3 A. Modify configuration record based on operator replies: ● Set logical IDs. ● Set WS/PRT attributes. ● Set default printer logical IDs. 4 Prepare general system parameters display (display 3). A Prepare WSDM DTF to output display 3.	$CNFIG
Display configuration defaults consisting of: ● Date format. ● Single program mode? ● Work station status after IPL. ● Command language.	#DWDM
Prepare WSDM DTF to retrieve operator replies.	$CNFIG
Retrieve operator replies from screen.	#DWDM
If display only request (CNDSP), go to 5. Validate operator replies: ● Single program mode — Y or N. ● Console mode after IPL — A or B. ● Command language — A or B. ● Date format. If errors detected, prepare operator error message and return to 4 A to redisplay with error message. Modify configuration record based on operator replies.	$CNFIG

Diagram 1.5 (Part 2 of 5). Perform System Configuration Processing ($CNFIG)

DESCRIPTION	MODULE/ ROUTINE
5 Prepare additional general system parameters display (display 4). Ⓐ Prepare WSDM DTF to output display 4.	$CNFIG
Display configuration defaults consisting of: ● Password security feature? – Security officer ID. – Security officer password. – Security file size (1 to 14 blocks). a. If PSF = No: 1. Put zero in configuration record as file size. 2. Delete security file if one exists. b. If JOBQ = No, set zero in configuration record as job queue size. ● Job queue support? – Job queue size (20 to 120 jobs).	#DWDM
Prepare WSDM DTF to retrieve operator replies.	$CNFIG
Retrieve operator replies from screen.	#DWDM
If display only request (CNDSP), go to **6**. Validate operator replies: ● If job queue yes, verify job queue size. ● If password security yes: – Verify security file size.	$CNFIG
▶ – Allocate security file if size on.	#CAS1 ◀
– Set officer ID in profile record. – Set officer password.	$CNFIG
– Write profile record to disk.	Disk IOS
If errors detected, prepare operator error message and return to **5** Ⓐ to redisplay with error message. Modify configuration record based on operator replies: ● Set job queue size. ● Set security file size. **6** Prepare general print parameters display (display 5). Ⓐ Prepare WSDM DTF to output display 5.	$CNFIG
Display configuration defaults consisting of: ● Standard forms ID. ● Lines per page (1 to 112). ● Print belt image. ● Print spooling? (Y,N).	#DWDM
Prepare WSDM DTF to retrieve operator responses.	$CNFIG
Retrieve operator replies from screen.	#DWDM
If display only request (CNDSP), go to **7**. Validate operator replies: ● Lines per page – 1 to 112. ● Print belt image – A, B, C, D, or E. ● Spool – Y or N. If errors detected, prepare operator error message and return to **6** Ⓐ to redisplay with error message.	$CNFIG

Diagram 1.5 (Part 3 of 5). Perform System Configuration Processing ($CNFIG)

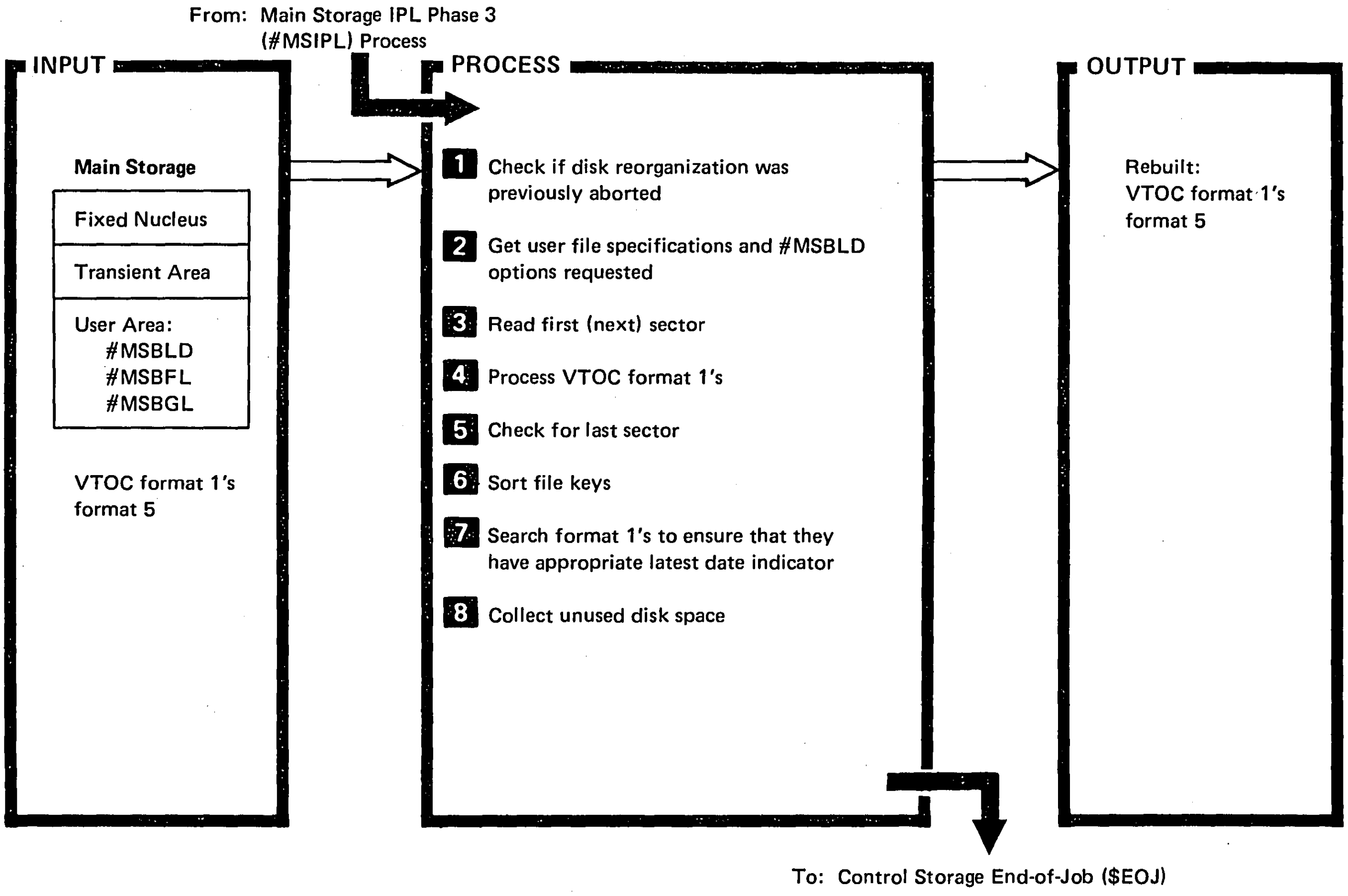
DESCRIPTION	MODULE/ ROUTINE
Modify configuration record based on operator replies: ● Set lines/page value in hexadecimal. ● Set print belt image in configuration record: – Use system find to locate source library get (#MASYL). – Use main storage relocating loader (SVC 52) to load #MASYL.	$CNFIG
– Retrieve requested print belt member.	#MASYL
– Convert to hexadecimal if necessary. – Move image into configuration record. ● Check spool yes or no: – If yes, set on spool indicator. – If no, clear spool parameters in configuration record and go to 7. 7 If spool not specified (CNSPFLAG) go to 8. Prepare spool parameters display (display 6). A Prepare WSDM DTF to output display 6.	$CNFIG
Display configuration defaults consisting of: ● Resident spool writer (Y,N). ● Spool writer priority (Y,N). ● Autowriter (Y,N). ● Spool intercept buffer size (1 to 8-1/2K). ● Spool file size (12 to 192 blocks). ● Spool writer buffer size (1 to 4-1/2K).	#DWDM
Prepare WSDM DTF to retrive operator replies.	$CNFIG
Retrieve operator replies from screen.	#DWDM
If display only request (CNDSP), go to 8. Validate operator replies: ● Resident spool writer – Y or N. ● Spool writer priority – Y or N. ● Autowriter – Y or N. ● Spool intercept buffer size – 1 to 8 1/2 K segments. ● Spool file size – 12 to 192 blocks. ● Spool writer buffer size – 1 to 4 1/2 K segments. If errors detected, prepare operator error message and return to 7 A to redisplay with error message. Modify configuration record based on operator replies: ● Set spool writer priority – yes or no. ● Set autowriter – yes or no. ● Set spool intercept buffer size: – Convert segments to sectors (2 sectors per 1/2 K segment). – Put size in configuration record. ● Set spool file size: – Convert blocks to hexadecimal. – Put size in configuration record. ● Set writer buffer size: Convert segments to sectors (2 sectors per 1/2 K segment). 8 Prepare performance display (display 7). A Prepare WSDM DTF to output display 7.	$CNFIG
Display configuration defaults consisting of: ● Display data management mode; A – transient, B – resident. ● Display station buffer size (6 to 16 1/2 K). ● System assign/free size (6 to 64 1/2 K).	#DWDM

Diagram 1.5 (Part 4 of 5). Perform System Configuration Processing ($CNFIG)

DESCRIPTION	MODULE/ ROUTINE
Prepare WSDM DTF to retrieve operator replies.	$CNFIG
Retrieve operator replies from screen.	#DWDM
If display only request (CNDSP) go to 9.	$CNFIG
Validate operator replies: ● Display data management mode. ● Display station buffer size — to 16 1/2 K segments. ● System assign/free size — 6 to 64 1/2 K segments.	
If errors detected, prepare operator error message and return to 8 A redisplay with error message.	
Modify configuration record based on operator replies: ● Display data management mode. ● Work station buffer size: – Convert to 1/4 K blocks. – Convert to hexadecimal. – Set size in configuration record. ● System assign/free size: – Convert value to 1/4 K blocks. – Convert value to hexadecimal. – Set converted 1/4 K blocks value in configuration record.	
9 Prepare program additions display — display 8.	
A Prepare WSDM DTF to output display 8.	
Display configuration defaults consisting of: ● MRJE support? (Y,N). ● BSC support? (Y,N). ● OLE support (Y,N).	#DWDM
Prepare WSDM DTF to retrieve operator replies.	$CNFIG
Retrieve operator replies from screen.	#DWDM
If display only request (CNDSP), go to 9 B.	$CNFIG
Validate operator replies (all replies must be Y or N).	
If errors detected, prepare operator error message and return to 9 A to redisplay with error message.	
Modify configuration record based on operator replies (set flags for Y or N).	
Move program addition flags to UPSI switch buffer (UPSI switches will be tested to direct the CNFIGSSP procedure).	
B If display only (CNDSP), zero UPSI switch buffer.	
Use information retrieval transient to set UPSI switches in job control block (JCB).	
10 If display only request (CNDSP), go to 10 A.	
Set on configuration complete flag (CONMCFGS).	
Write modified configuration records to disk.	Disk IOS
A Close WSDM DTF.	#DMCL
Pass control to end of job transient ($EOJ).	$CNFIG

Diagram 1.5 (Part 5 of 5). Perform System Configuration Processing ($CNFIG)

From: Main Storage IPL Phase 3
(#MSIPL) Process



INPUT

Main Storage

Fixed Nucleus

Transient Area

User Area:
#MSBLD
#MSBFL
#MSBGL

VTOC format 1's
format 5

PROCESS

1 Check if disk reorganization was previously aborted

2 Get user file specifications and #MSBLD options requested

3 Read first (next) sector

4 Process VTOC format 1's

5 Check for last sector

6 Sort file keys

7 Search format 1's to ensure that they have appropriate latest date indicator

8 Collect unused disk space

OUTPUT

Rebuilt:
VTOC format 1's
format 5

To: Control Storage End-of-Job ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Read Format 5. If disk reorganization ($FREE) was interrupted, complete it.	#MSBLD $FREE
2 Get range of user file addresses. Get operator responses.	#MSBLD
3 Read first (next) sector in the user VTOC.	Disk IOS
4 Address format 1 entry: ● If this entry is unused, go to 4 A. ● If the file is not new and the operator does not want the format 1 processed, go to 4 A. ● If the file is sequential or indexed, ensure that the record number reflects the number of records in the file. ● If the file is indexed and the number of records was altered, reconstruct the index from the data records. Clear the format 1 if the user requests that the file in error be deleted and if any of the following error conditions exist: ● latest date indicator is invalid. ● disk reorganization utility ($FREE) could not successfully move this file. ● record length exceeds 4096 bytes. ● retention is other than temporary or permanent. ● type is other than sequential, direct, or indexed.	#MSBLD

Diagram 1.6 (Part 1 of 2). Rebuild VTOC Format 1's

DESCRIPTION	MODULE/ ROUTINE
● file is indexed and key position and/or key length are invalid. ● file is indexed and the number of keys is not equal to the number or records. ● file extents are not commensurate with the user disk space or with each other. A If there is another format 1 entry in the current sector, go to 4.	#MSBLD
5 Rewrite the sector into the VTOC. If there are more sectors in the VTOC, go to 3.	Disk IOS
6 Load Keysort (#DDKAA).	#MSBFL
A Read first (next) sector in user VTOC.	Disk IOS
B Address format 1 entry: If the sort/merge bits are on, call #DDKAA to sort the keys.	#MSBFL #DDKAA
If there is another format 1 in the sector, go to 6 B.	#MSBFL
Rewrite the sector into the VTOC.	Disk IOS
If there are more sectors in the VTOC, go to 6 A.	#MSBFL
7 Read first (next) sector in VTOC.	Disk IOS
A Address first (next) format 1: ● If this entry is unused or if a file with this same label has been processed, go to 7 A. ● If other files in this sector have the same label, scan until all entries have been compared with the current entry. ● Scan the remaining sectors of the VTOC for a file with the same label as the current entry. For each file with the same label, scan until all entries have been compared with the current entry. ● If the current format 1 has a later date, set the latest date indicator in the format 1; otherwise, set binary zeros. ● If this is not the last entry in the current sector, go to 7 A.	#MSBGL
Rewrite the sector into the VTOC.	Disk IOS
If there are more sectors in the VTOC, go to 7.	#MSBGL
8 Read the format 5.	Disk IOS
Indicate that $FREE should recover all free disk space.	#MSBGL
Rewrite the format 5.	Disk IOS
Call disk reorganization ($FREE) which passes control to the end-of-job transient ($EOJ).	#MSBGL

Diagram 1.6 (Part 2 of 2). Rebuild VTOC Format 1's

Program Organization

Figures 1-1 through 1-3 show the control flow required to start the system.

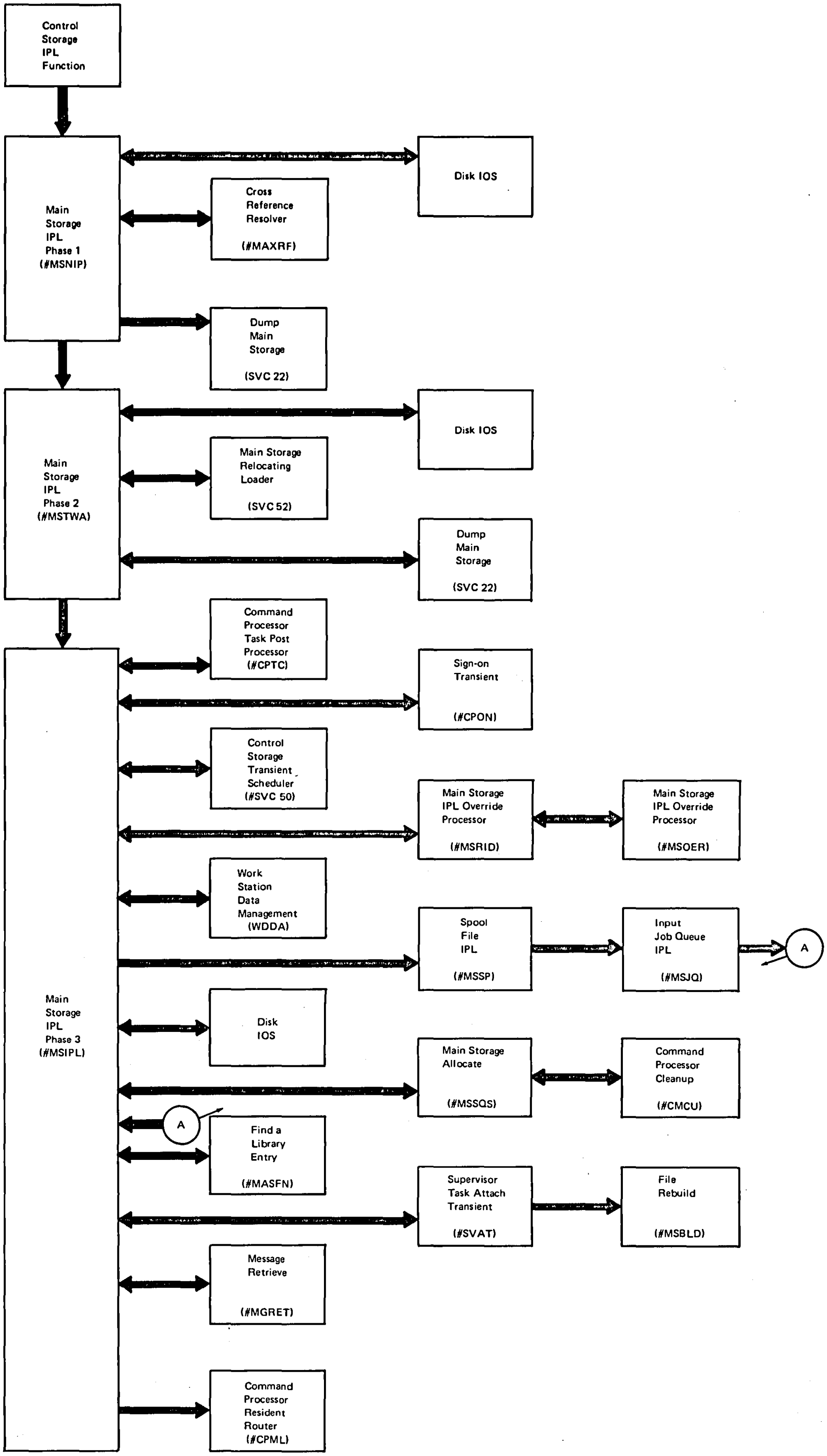

Control Storage IPL Function
Main Storage IPL Phase 1 (#MSNIP)
Disk IOS
Cross Reference Resolver (#MAXRF)
Dump Main Storage (SVC 22)
Main Storage IPL Phase 2 (#MSTWA)
Disk IOS
Main Storage Relocating Loader (SVC 52)
Dump Main Storage (SVC 22)
Main Storage IPL Phase 3 (#MSIPL)
Command Processor Task Post Processor (#CPTC)
Sign-on Transient (#CPON)
Control Storage Transient Scheduler (#SVC 50)
Main Storage IPL Override Processor (#MSRID)
Main Storage IPL Override Processor (#MSOER)
Work Station Data Management (WDDA)
Spool File IPL (#MSSP)
Input Job Queue IPL (#MSJQ)
A
Disk IOS
Main Storage Allocate (#MSSQS)
Command Processor Cleanup (#CMCU)
A
Find a Library Entry (#MASFN)
Supervisor Task Attach Transient (#SVAT)
File Rebuild (#MSBLD)
Message Retrieve (#MGRET)
Command Processor Resident Router (#CPML)

Figure 1-1. Main Storage IPL Control Flow

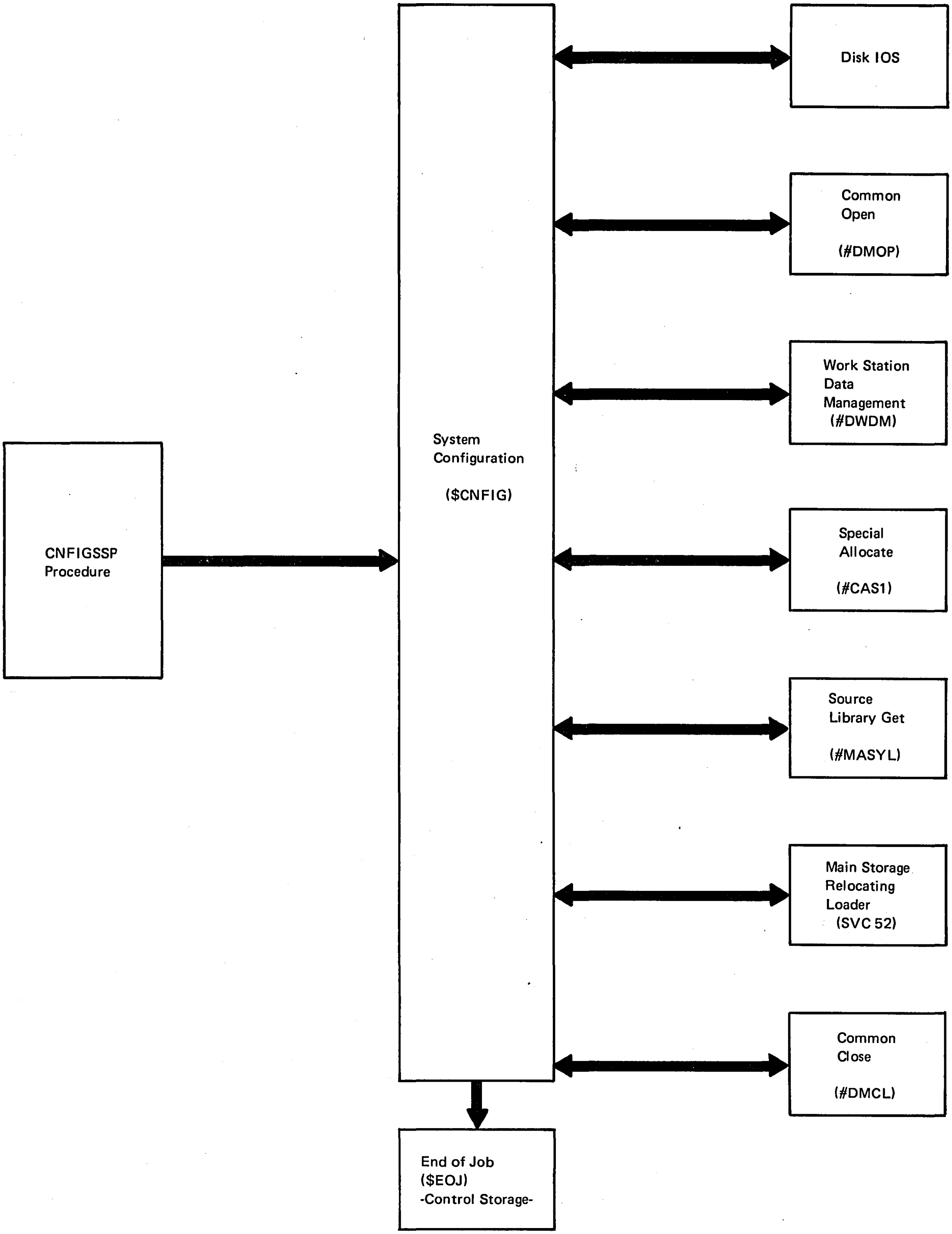

Disk IOS
Common Open (#DMOP)
Work Station Data Management (#DWDM)
System Configuration ($CNFIG)
CNFIGSSP Procedure
Special Allocate (#CAS1)
Source Library Get (#MASYL)
Main Storage Relocating Loader (SVC 52)
Common Close (#DMCL)
End of Job ($EOJ) -Control Storage-

Figure 1-2. System Configuration Logic Flow

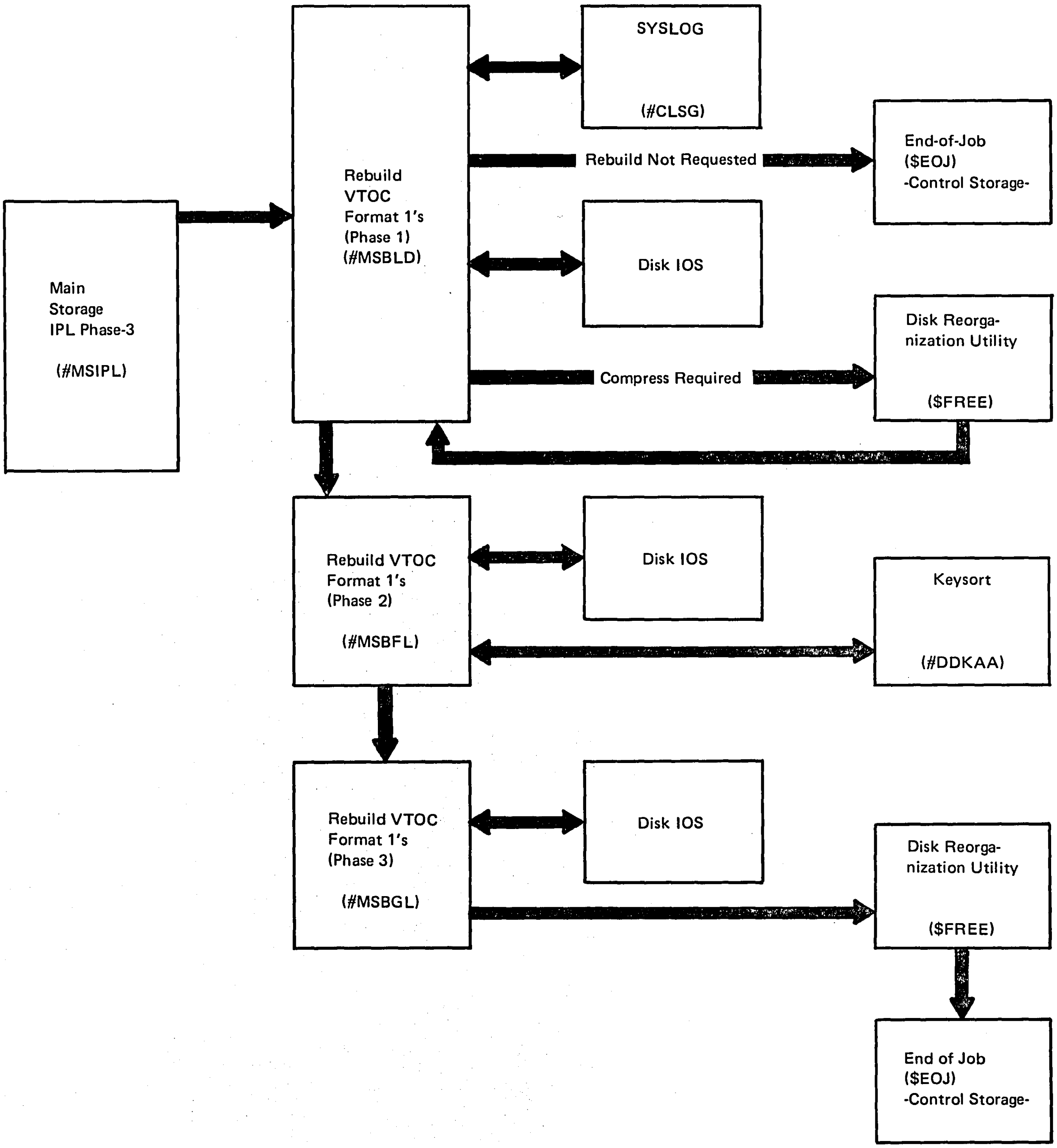

SYSLOG
(#CLSG)
Rebuild VTOC Format 1's (Phase 1) (#MSBLD)
Main Storage IPL Phase-3 (#MSIPL)
Rebuild Not Requested
End-of-Job ($EOJ) -Control Storage-
Disk IOS
Compress Required
Disk Reorganization Utility ($FREE)
Rebuild VTOC Format 1's (Phase 2) (#MSBFL)
Disk IOS
Keysort (#DDKAA)
Rebuild VTOC Format 1's (Phase 3) (#MSBGL)
Disk IOS
Disk Reorganization Utility ($FREE)
End of Job ($EOJ) -Control Storage-

Figure 1-3. Rebuild VTOC Format 1's Control Flow (#MSBLD)

Chapter 2. Processing Commands

Introduction

The functions needed to process commands are:

- Router
- Sign-on
- Process control commands
- Job initiation and work station release
- High-level aids and task-to-task communications router
- Inquiry menu option processor and resume
- Special command processor
- Command processor/work station data management interface
- Console management
- Cleanup
- Password security
- I/O error recovery

COMMAND PROCESSOR

The command processor provides an operator interface for display station and system operators to direct the System/34 SSP in performing the operator's tasks.

The command processor provides control over an operator's work session with the System/34 SSP by:

- Controlling the format of the display screen displays when the display station is not in use by an application program.
- Providing an interface for operators to carry on a dialogue with the System/34 SSP in order to submit procedures and OCL statements for execution.
- Providing operator commands that are immediate commands not requiring the initiator function for execution.
- Providing error messages and prompts to help the operator conduct a work session and log related information to the system history file.

Figure 2-1 shows the functions provided by the command processor.

Router

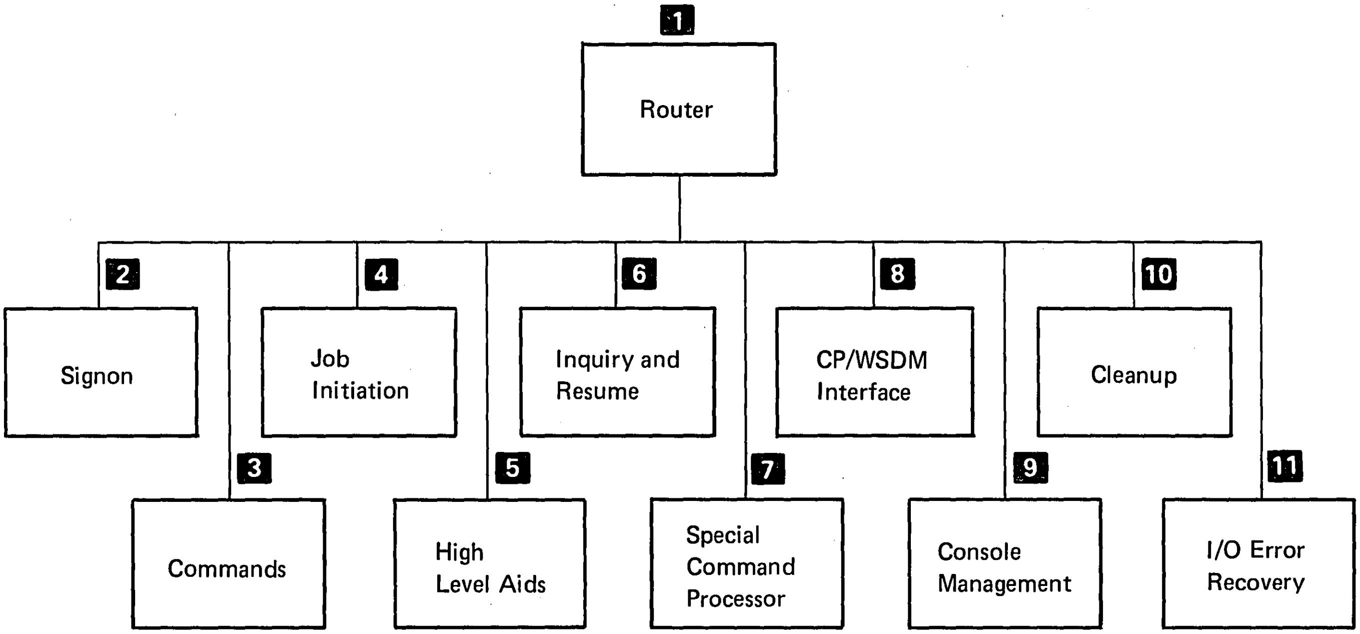
The command processor router function waits for events to occur and routes control to the proper command processor transient module to process the event.

Events that cause the router to gain control are:

- Invite op-end – an operator has pressed a command key or entered a command or OCL statement.
- Attn post – an operator has requested the inquiry function by pressing the Attn key.
- Sys req post – an operator has pressed the System Request and Enter/Rec Adv keys to get a sign-on display or to change interfaces at the system console.
- Task-to-task communications post.

Sign on

Before beginning a work session, the operator must perform a sign-on procedure. The sign-on procedure either initializes a display station for entering commands and OCL, or initializes the display station for acquisition by a user program (standby mode). Only display stations in standby mode can be acquired by user programs. In addition, the sign-on procedure is used to validate the operator as an authorized user (password security).



Description:

1 The router determines which event to process and passes control to the appropriate command processor function.

2 The sign-on function establishes the interface between the display station operator and the System/34 SSP.

3 The command function includes control commands for the display station and system operator, spool, and input JOBQ.

4 The job initiation and display station release function provides for the initiation and termination of the scheduler.

5 The high level aids and task-to-task communications router function performs the system request, attention, and help functions along with other task-to-task communications functions.

6 The inquiry menu option processor and resume function provide the operator the capability to interrupt a currently executing program, initiate a new request, and later resume the interrupted program.

7 The special command processor function provides command key processing and other miscellaneous functions.

8 The command processor/work station data management function provides a common module to display several system formats.

9 The console management function provides user and system tasks a method of writing a message to a display station (or to the system console) and receiving a reply.

10 The cleanup function outputs error detected by the command processor, controls the format of the screen, and outputs session history.

11 The I/O error recovery function allows I/O devices to issue error messages to the system console and attempts recovery from certain I/O errors.

Figure 2-1. Command Processor Functions

The command processor provides a Sign on display to the operator to aid in the sign-on procedure. The Sign-on display appears at a display station, following IPL, when the operator presses the System Request key followed by the Enter/Rec Adv key, or whenever an operator signals the end of a work session with the OFF command. All operators including the system operator must perform the sign-on procedure.

To perform the sign-on procedure, the operator must fill in the necessary fields defined on the Sign-on display and press the Enter/Rec Adv key.

Process Control Commands

The control commands provide an interface between the display station and system operator and the System/34 system support program (SSP). Control commands are immediate commands that do not require scheduler functions for execution.

Control commands are provided in two categories, display station control commands and system console control commands. Some commands, however, can be used at both the display station and system console.

The following chart indicates which control commands can be used at the display station, system console, or both.

Command Name	Display Station Command	System Console Command
ASSIGN		X
CANCEL	X	X
CHANGE		X
CONSOLE	X	
HOLD		X
IDELETE	X	X
JOBQ	X	
MENU	X	
MODE	X	
MSG	X	X
OFF	X	
PRTY	X	X
RELEASE		X
REPLY		X
RESTART		X
START		X
STATUS	X	X
STOP		X
TIME	X	X
VARY		X

Figure 2-2 is a summary of control commands. It provides a brief description of each command function, indicates the processing modules used, and shows the diagram number.

Refer to the *System Support Reference Manual,* and *IBM System/34 System Operator's Guide,* SC21-5158 for command formats and operating procedures.

Job Initiation and Work Station Release

The job initiation and work station release function receives control from the resident router (#CPML) when the operator enters a procedure or an OCL statement while the display station is in command mode. The job start function is responsible for:

- Starting the initiator (#CIML) to process the operator entered procedure or OCL statement.
- Attaching a work station to an active multiple requester terminal (MRT) task.

The release function receives control from the command processor resident router (#CPML) when work station data management posts the command processor to perform a release. The release function is responsible for:

- Reestablishing the command interface at end-of-job for a command work station.
- Reestablishing the standby interface at step or job end for a data work station.
- Attaching the initiator to process the next job step when a requester is released from a MRT task or a released program task.

High-Level Aids and Task-to-Task Communications Router

The keys that invoke high-level aids are: the Attn key, the Sys Req key and Enter key, and the Help key (when in operator error mode). The command processor receives control whenever one of these keys is pressed.

Command	Function	Module	Diagram
ASSIGN	Used by the system operator to temporarily exchange the IDs of two display stations or two printer, or to temporarily assign a printer as the system printer.	#CCAS	2.4
CANCEL	Used by the system operator to cancel any of the following: ● A specified job on the input job queue ● All jobs on the input job queue ● A specified job on the spool file ● All jobs on the spool file ● A currently executing job Used by the display station operator to cancel a job on the input job queue.	#CCCM, #CCJQ, and #CCCP	2.5
CHANGE	Used by the system operator to change the following: ● The position of a job on the input job queue or the spool file ● The number of copies to be printed for a job on the spool file ● The forms number to be used for the job on the spool file	#CCJQ and #CCGP	2.6
CONSOLE	Used by the system operator to cause an alternative console to become the system console.	#CCCO	2.7
HOLD	Used by the system operator to temporarily prevent a specified job or all jobs on the spool file from being printed.	#CCHO	2.8
IDELETE	Used by the system operator to specify whether informational messages directed to the system console from a procedure should be automatically responded to. Used by the display station operator to specify whether informational messages directed to the display station from procedures are to be displayed.	#CCID	2.9
JOBQ	Used by the display station operator to place a job on the input job queue.	#CCJQ	2.6
MENU	Used by the display station operator to activate the menu function and to display the specified menu.	#CCMU	2.10
MODE	Used by the display station operator to change from command mode to standby mode or from standby mode to command mode.	#CCOF	2.12
MSG	Used by the system operator to send a message to all display stations or to a selected display station or display station operator. Used by the display station operator to send a message to the system console or to a selected display station operator; or, when entered with no parameters, to display any messages pending at the display station.	#CCMG	2.11
OFF	Used by the display station operator to terminate a display station session.	#CCOF	2.12

Figure 2-2 (Part 1 of 3). Control Command Summary

Command	Function	Module	Diagram
PRTY	Used by the system operator to change the dispatching priority of a currently executing job.	#CCPY	2.13
	Used by the display station operator to assign a priority to the next job run from the display station or the next job placed on the input job queue.		
RELEASE	Used by the system operator to release for printing either the entire spool file, a job on the spool file, or all jobs on the spool file that were individually held or that were placed on the spool file with PRIORITY-0.	#CCHO	2.8
REPLY	Used by the system operator to do one of the following: ● Respond to all informational messages on the display screen ● Compress the display so that only messages still needing a response are displayed ● Respond to an individual message	#CCRE	2.14
RESTART	Used by the system operator to restart the printing of a job from the spool file.	#CCRT and #CCRR	2.15
START	Used by the system operator to do one of the following: ● Start the printing of jobs from the spool file ● Allow the initiation of jobs from all display stations or from a specified display station ● Start the running of jobs from the input job queue ● Resume the system activity that was stopped by a STOP SYSTEM control command ● Resume the execution of a job, or all jobs, that were stopped by a STOP JOB control command	#CCRT, #CCRR, and #CCJS	2.15
STATUS	Used by the system operator to display any of the following: ● The entries on the spool file ● The status of jobs running on the system ● Any entries on the input job queue ● Status information about the display stations, printers, and the diskette drive	#CCSM, #CCSP, #CCSW, #CCSU, #CCU2, and #CCSJ	2.16
	Used by the display station operator to display any of the following: ● Status of the display station session ● Input job queue entries for jobs submitted from the display station by the current display station operator ● Status of the display stations, printers and diskette drive	#CCSS, #CCSW, #CCS2, #CCS3, #CCS4, and #CCSJ	

Figure 2-2 (Part 2 of 3). Control Command Summary

Command	Function	Module	Diagram
STOP	Used by the system operator to do any of the following: ● Stop the printing of jobs from the spool file ● Stop the initiation of jobs from all display stations except the system console, or stops the initiation of jobs from a specified display station ● Stop the initiation of jobs from the input job queue ● Begin an orderly shutdown of the system with or without a key sort ● Stops the execution of all jobs or a specified job	#CCRT, #CPTC, #CCRR, and #CCJQ	2.15
TIME	Used by the system operator and the display station operator to display the time of day and the system date.	#CCTD	2.17
VARY	Used by the system operator to change the status of the display station, a printer, the system printer, or the diskette drive from online to offline or from offline to online.	#CCAS	2.4

Figure 2-2 (Part 3 of 3). Control Command Summary

The operator interrupts an executing program with the Attn key. The inquiry display that results from the interrupt allows the operator to: (1) resume the interrupted program; (2) return to a command interface where he can run another program; (3) cancel the interrupted program; or, (4) set inquiry condition and resume.

The Sys Req key and Enter key allow the system operator to switch the system console between work station mode and system console mode.

The Help key displays a description of the flashing error number that appears in the bottom left corner of the display screen.

When control is received from the task-to-task communications router, the command processor checks the event control mask (ECM) to determine if the call is due to an I/O error; if it is, control is given to I/O error recovery (#SVERP). If the call was not due to an I/O error, the command processor checks a series of pending functions to determine if control was received to process the functions. The functions that can be performed are, release, console SYSLOG, task suspend, stop system, and JOBQ detach.

Inquiry Menu Option Processor and Resume

The command processor inquiry menu option processor and resume function allows the operator to interrupt executing programs to submit new jobs or commands and then later resume the interrupted program.

The inquiry menu processor receives control when the operator selects an option from the inquiry menu. The inquiry menu processor handles the RESUME, INQUIRY, CANCEL, RELEASE, and inquiry condition options.

The resume function, initiated by command key 1 from the command display, reattaches the terminal to the interrupted program.

Special Command Processor

The special command processor (#CPSP) handles router accept errors and command key functions. It also receives control when the Enter key is pressed (1) after a program has released a display station at end of job, (2) after messages were displayed from the MSG command, or (3) after a second level message was displayed at the system console.

Command Processor/Work Station Data Management Interface

Command processor/work station data management interface (#CPIO) is called with a RIB to indicate which type of I/O is requested. #CPIO builds a parameter list and passes control to work station data management (#DWDM).

Console Management

Command processor console management provides a way to route messages to the system console or work stations and, if necessary, retrieve a response from the operator.

The main functions performed by console management are:

- Move message elements from the user's system-log sector to the console system-log disk queue.
- Free system-log queue sector for reuse.
- Display messages at the system console.
- Perform End-of-Job processing for console management.

The work station logical I/O interface provides logical I/O keyboard data management services for SYSIN and logical SYSLOG/SYSLIST data management services.

The main data management services for SYSIN are:

- Save the user's display screen.
- Display logical I/O interface at the display station.
- Process user response data from the keyboard.

The main data management services for SYSLOG/SYSLIST are:

- Save the user's display screen.
- Display the logical I/O interface.
- Display a message or halt on the display screen.
- Display second-level messages as required.
- Process write to operator with reply (WTOR) and write to operator without reply (WTO) messages.
- Retrieve and return responses to the user.

Console management normally returns control to the command processor mainline module (#CPML) and work station logical I/O returns control to the calling program.

Command Processor Cleanup

The command processor cleanup routine provides cleanup activity and screen control for the other command processor transient modules.

The functions provided by the cleanup routine are:

- Log input area to the history file
- Retrieve messages by message identification code (MIC) or retrieve messages from main storage and display to specified display station
- Log messages to the printer when applicable
- Substitute data in message formats as required
- Place system console messages on the system console queue

Password Security

Password security helps the System/34 user prevent unauthorized use of the system.

The user must request password security at system configuration time to invoke the password security function. If the function is requested, system configuration routines will:

- Allocate security file space and initialize it to binary zero
- Build the master security officer record and write it to the security file
- Initialize the appropriate system configuration record indicators

At IPL time, system communication area (SCA) is initialized to indicate password security is active and security initialization is required. When system configuration is complete, the master security officer can sign on and authorize other system users by using the security file utility ($PROF) or the security file restore utility ($PRST). The master security officer can also use $PROF to alter fields within a particular security file record or prevent a particular system user sign on.

The security file contains information about system users authorized to sign on the system. A record for each authorized user contains information such as:

- User ID
- Password
- Security classification
- Comments

The security sign-on check is performed by security file data management (#PRSD). #PRSD is called by sign-on (#CPON). #PRSD searches the security file for the user ID and verifies the associated password and classification.

I/O Error Recovery

I/O error recovery issues error messages, recognizes operator responses, and handles exception conditions. The command processor serves as the interface between I/O error recovery (main storage) and the error recovery procedures (control storage and main storage).

I/O error recovery consists of:

- The error recovery router (#SVERP)
- The work station error message transient (#SVWER)
- The display station error recovery transient (#SVWSR)
- The display station error recovery for device not ready transient (#SVNRY)
- The printer prepare transient for matrix printers (#SVPRE)
- The command reject ready transient (#SVUR) and routine (#SVRD)

The command processor initiates I/O error recovery when the command processor error event control mask (ECM) is posted by a control storage routine. The command processor calls the I/O error recovery router (#SVERP), which determines what I/O error recovery functions are required.

If an error message is to be displayed, #SVERP calls the I/O error message transient (#SVWER). For display station errors, #SVERP calls display station error recovery (#SVWSR); for all other I/O device errors, #SVERP passes control to a control storage transient.

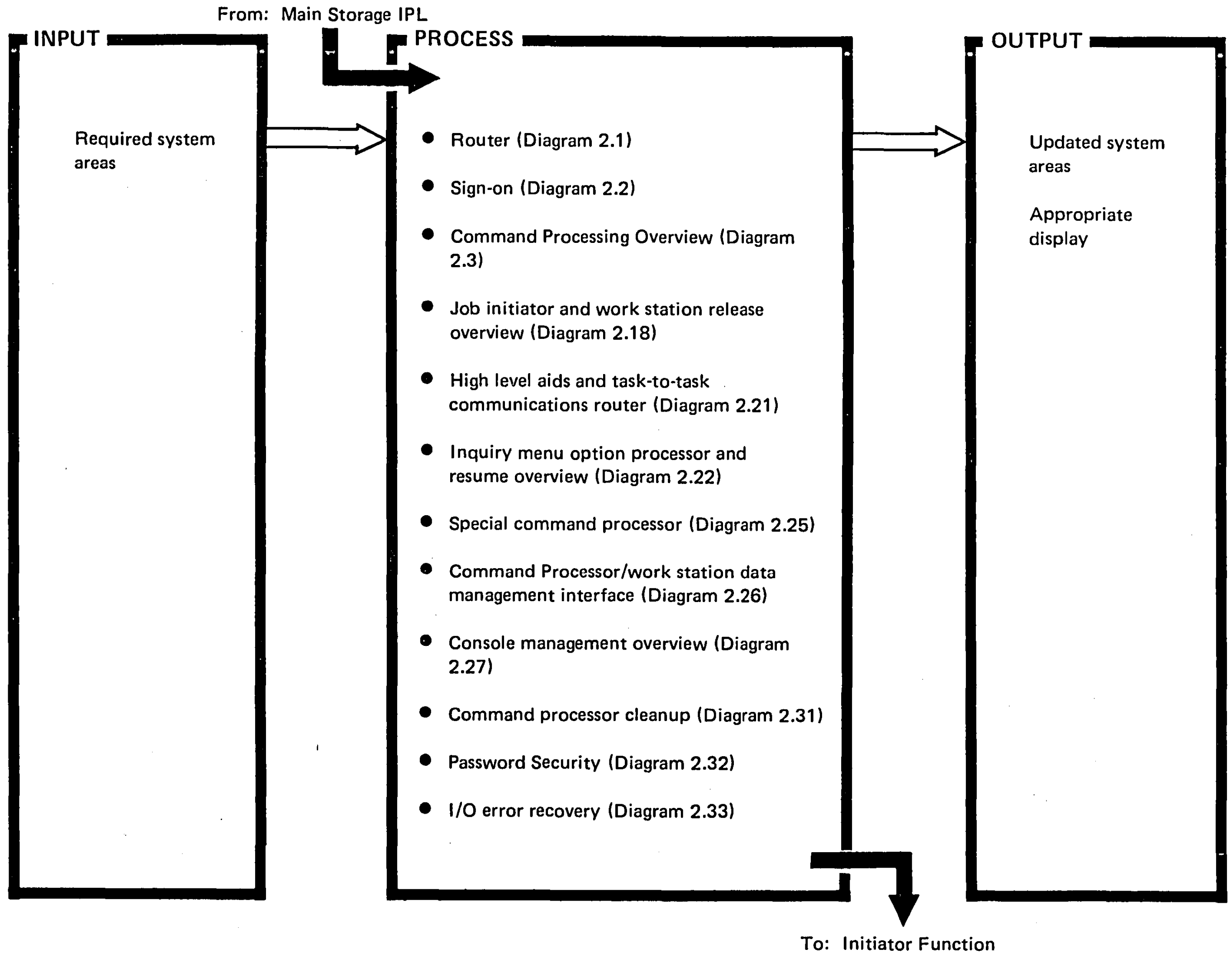
Display Station Error Recovery

The display station error recovery transient (#SVWSR) handles error recovery for display station hardware and program related errors. For device not ready errors, #SVWSR calls the display station error recovery for device not ready transient (#SVNRY). The command reject ready transient (#SVUR) pushes/pulls the user task for #SVRD, when required. The command reject ready routine (#SVRD) processes command rejected records.

Method of Operation

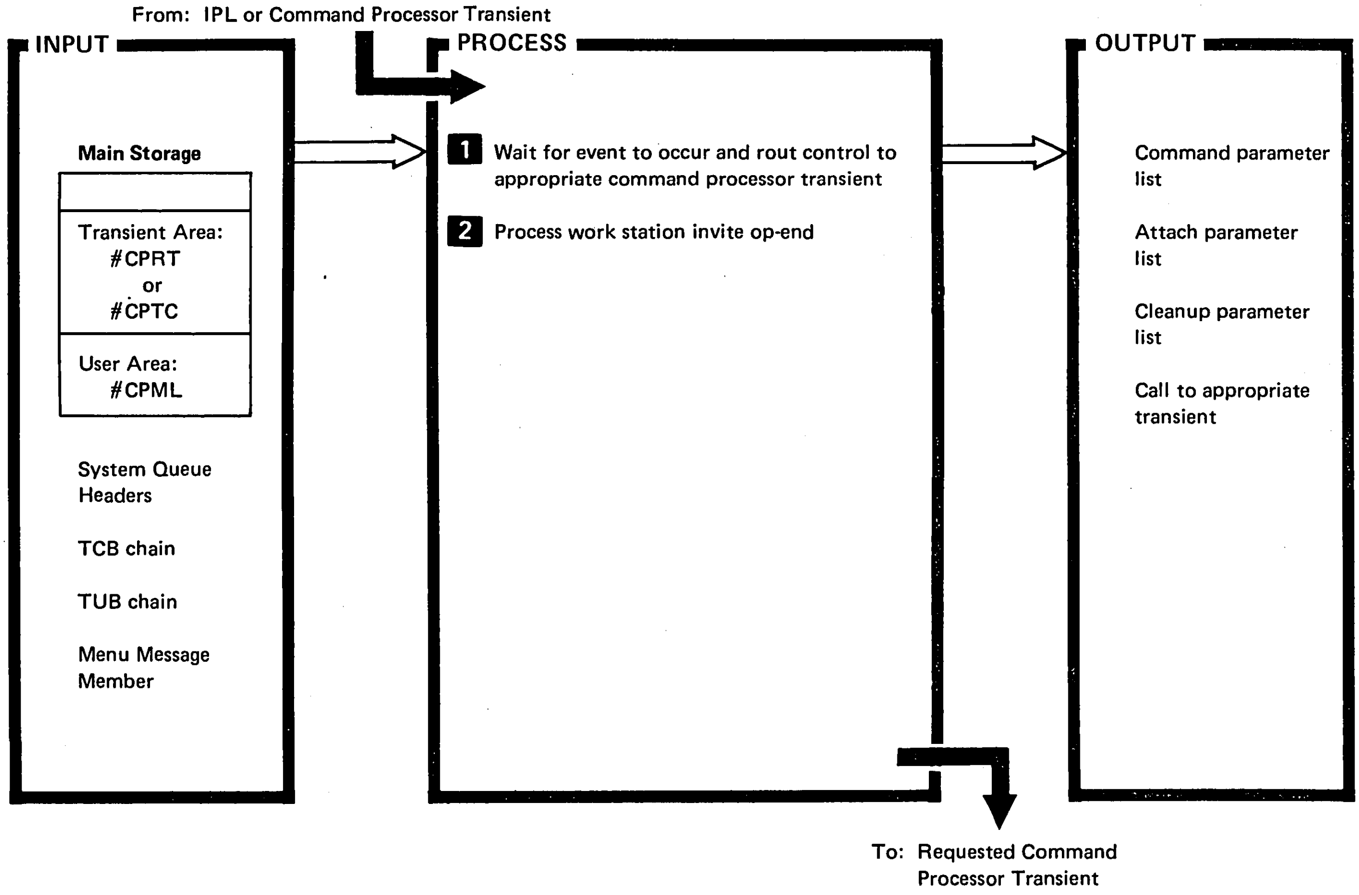
This section contains functional diagrams for the routines needed to process commands. They are:

- Router
- Sign-on
- Command processing
- Job initiation and display station release
- High level aids and task-to-task communications router
- Inquiry menu option and resume processing
- Special command processing
- Command processor/work station data management interface
- Console management
- Command processor cleanup
- Password security
- I/O error recovery
- Work station error recovery



- Router (Diagram 2.1)
- Sign-on (Diagram 2.2)
- Command Processing Overview (Diagram 2.3)
- Job initiator and work station release overview (Diagram 2.18)
- High level aids and task-to-task communications router (Diagram 2.21)
- Inquiry menu option processor and resume overview (Diagram 2.22)
- Special command processor (Diagram 2.25)
- Command Processor/work station data management interface (Diagram 2.26)
- Console management overview (Diagram 2.27)
- Command processor cleanup (Diagram 2.31)
- Password Security (Diagram 2.32)
- I/O error recovery (Diagram 2.33)

Diagram 2.0. Overview of Command Processor



From: IPL or Command Processor Transient

INPUT

Main Storage

Transient Area: #CPRT or #CPTC

User Area: #CPML

System Queue Headers

TCB chain

TUB chain

Menu Message Member

PROCESS

1 Wait for event to occur and rout control to appropriate command processor transient

2 Process work station invite op-end

OUTPUT

Command parameter list

Attach parameter list

Cleanup parameter list

Call to appropriate transient

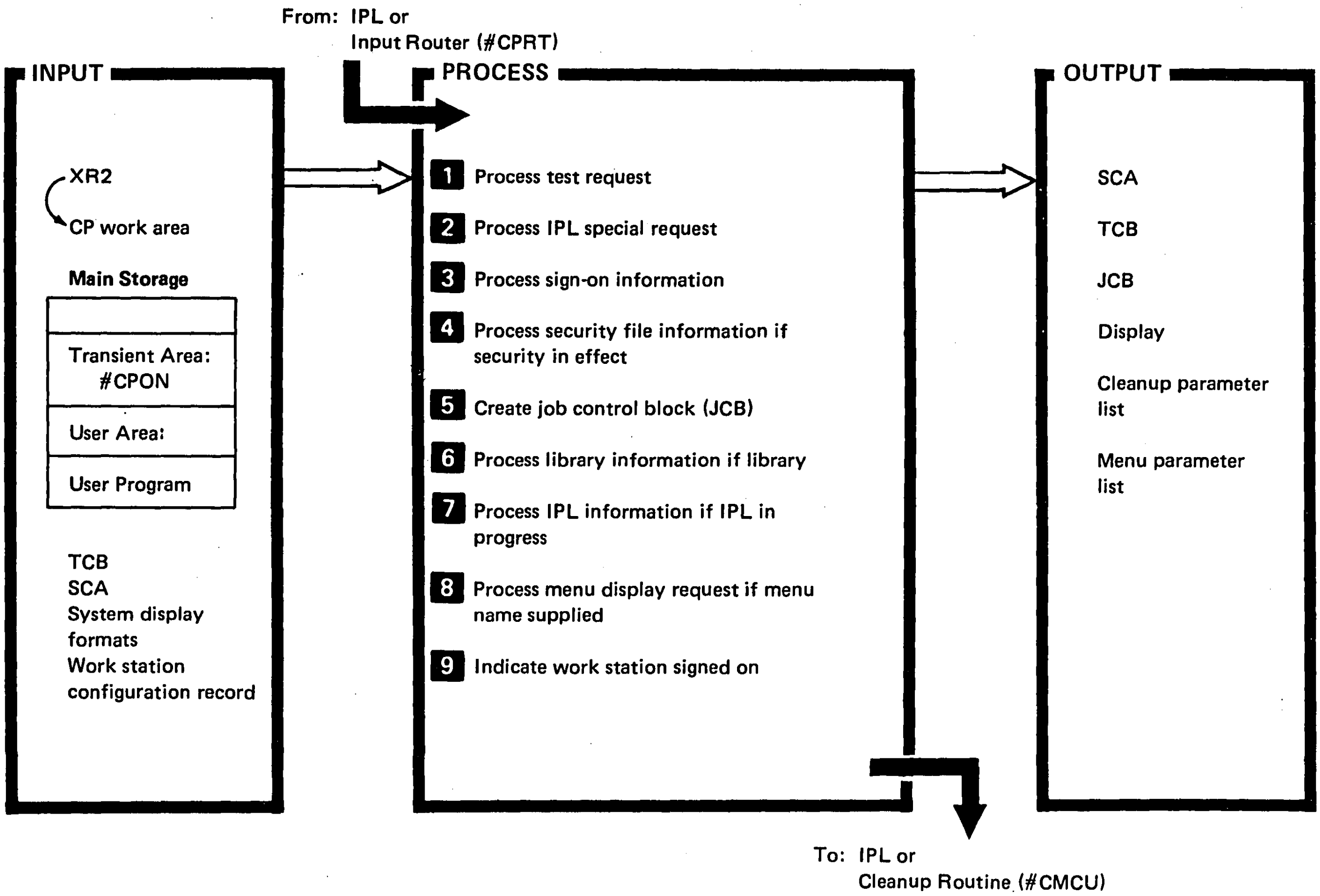
To: Requested Command Processor Transient

DESCRIPTION	MODULE/ ROUTINE
1 Wait for event to occur. If event is invite op-end, go to **2**. If event is high-level aids or task-to-task communications exit to #CPTC (Diagram 2.21).	#CPML
2 Check for special routing conditions (preaccept): ● Input not caused by Enter key. ● Input only to cause screen restore or refresh. ● Input is release acknowledgement. If any of the above conditions exist, exit to #CPSP. If invite op-end: ● Build work station data management parameter list.	#CPRT
● Accept input data.	#DWDM
● Process specials (post accept): – If input for sign on request, exit to sign on transient (#CPON). – If input from status display and not I, exit to status transient (#CCSM). – If console output, exit to console output transient (Logical I/O function – #CMCI).	#CPRT

Diagram 2.1 (Part 1 of 2). Perform Router Function

DESCRIPTION	MODULE/ ROUTINE
● Process menu input (if applicable): – Build message retrieve parameter list. – Convert input data to message identification code (MIC).	#CPRT
– Retrieve MIC and place in input buffer.	#MGRET
– If menu cancel request, exit to menu command processor (#CCMU). ● Scan input: – Isolate verb and operand data. – If input an inquiry menu option, exit to inquiry menu processor (#CPIQ). – If input a command: a. Verify that command allowed in present mode. b. Build command transient parameter list in command processor work area. c. Exit to command or function requested. – If input a job request: a. Ensure job initiation allowed at requesting work station. b. If inquiry and job control block (JCB) and work station work area not allocated, link to #CPIQ to allocate space (inquiry menu processing function).	#CPRT
c. If not multiple requester terminal (MRT), start initiator (job start function).	#SVAT
d. If MRT and work station can be attached, attach terminal unit block (TUB) to task (job start function). If errors detected: ● Build #CMCU parameter list. ● Exit to cleanup (#CMCU) to display message (Diagram 2.31).	#CPRT

Diagram 2.1 (Part 2 of 2). Perform Router Function



From: IPL or Input Router (#CPRT)

INPUT

XR2

CP work area

Main Storage

Transient Area: #CPON

User Area:

User Program

TCB
SCA
System display formats
Work station configuration record

PROCESS

1 Process test request

2 Process IPL special request

3 Process sign-on information

4 Process security file information if security in effect

5 Create job control block (JCB)

6 Process library information if library

7 Process IPL information if IPL in progress

8 Process menu display request if menu name supplied

9 Indicate work station signed on

OUTPUT

SCA

TCB

JCB

Display

Cleanup parameter list

Menu parameter list

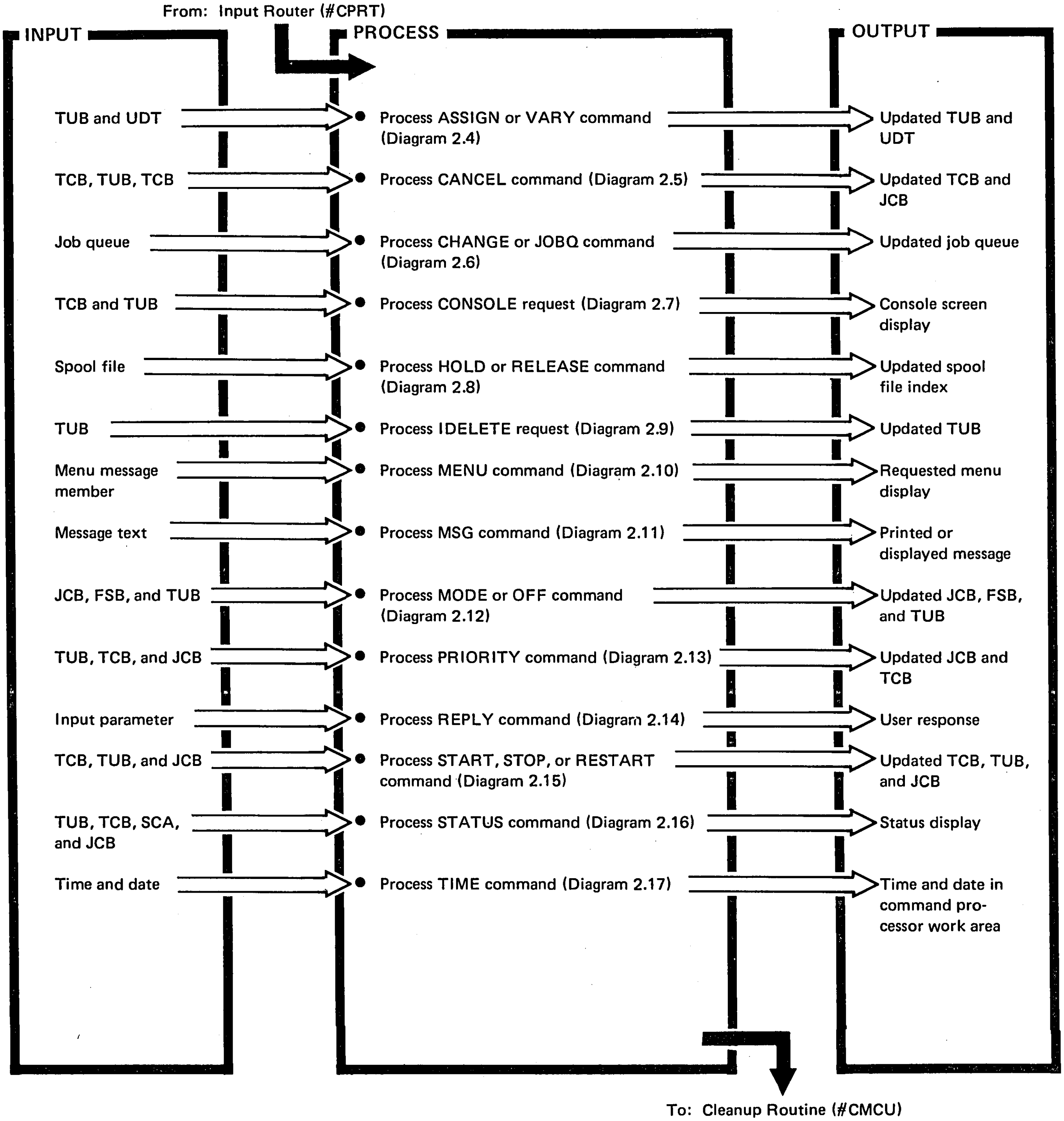
To: IPL or Cleanup Routine (#CMCU)

DESCRIPTION	MODULE/ ROUTINE
1 If test request call: ● Build JCB. ● Sign on work station. ● Attach the initiator to run test request program. ● If job submission halted, issue error message.	#CPON
2 If IPL special request, put up appropriate display (command or system).	
3 Verify USERID syntax and check that high-level dedication not in effect.	
4 Examine security file flag in system communication area (SCA) to determine if security in effect.	
Retrieve user's security profile.	#PRSD
Determine if: ● Security file was found. ● User ID in security file. ● Operator entered correct password.	#CPON
If security information not correct, issue error message.	#CMCU

Diagram 2.2 (Part 1 of 2). Perform Sign-on Function

DESCRIPTION	MODULE/ ROUTINE
5 If data work station: ● Set up work station data management parameter list for standby display. ● Indicate in terminal unit block (TUB) that work station signed on and in standby mode.	#CPON
▶ ● Display standby screen.	#DWDM ◀
● If messages pending for this display station,	#CPON
display them.	#CCMX
● Load and pass control to cleanup routine (#CMCU).	#CPON
If command work station: ● Assign job control block (JCB). ● Chain TUB to JCB. ● Initialize JCB with work station configuration record and default values. ● Put printer ID in TUB. **6** Determine if library name entered. If library name not 0, set up library find parameter list.	#CPON
Find specified library.	#MAFLB
Move library format 1 address into JCB.	#CPON
If library not found, issue error message.	#CMCU
7 Determine if IPL in progress. Move date into SCA if date specified. Set system timer if time is valid.	#CPON
Issue error message if invalid time entered.	#CMCU
Indicate overrides in SCA if OVERRIDES=Y.	#CPON
8 Determine if menu specified. Set up menu parameter list. Build menu format index.	
Ensure menu on.	#CCMU
9 Indicate display station signed on in TUB. Put user ID in TUB and system date in JCB. If IPL in progress, return to IPL procedure. If not IPL, build work station data management parameter list.	#CPON
Display requested screen: ● Command Display. ● Menu display.	#DWDM
If messages pending for this display station, ▶ display them.	#CPON #CCMX ◀
If region size was bad, issue warning. Load and pass control to the cleanup routine (#CMCU).	#CPON

Diagram 2.2 (Part 2 of 2). Perform Sign-on Function

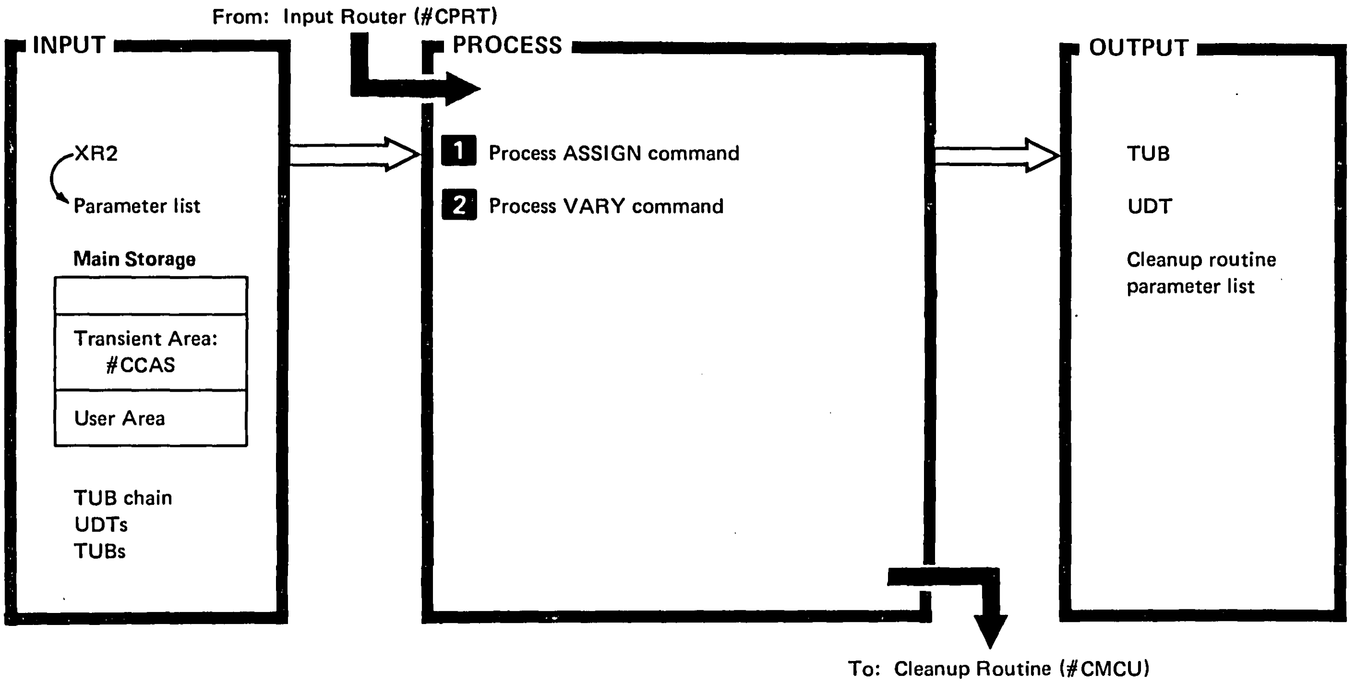


From: Input Router (#CPRT)

INPUT	PROCESS	OUTPUT
TUB and UDT	Process ASSIGN or VARY command (Diagram 2.4)	Updated TUB and UDT
TCB, TUB, TCB	Process CANCEL command (Diagram 2.5)	Updated TCB and JCB
Job queue	Process CHANGE or JOBQ command (Diagram 2.6)	Updated job queue
TCB and TUB	Process CONSOLE request (Diagram 2.7)	Console screen display
Spool file	Process HOLD or RELEASE command (Diagram 2.8)	Updated spool file index
TUB	Process IDELETE request (Diagram 2.9)	Updated TUB
Menu message member	Process MENU command (Diagram 2.10)	Requested menu display
Message text	Process MSG command (Diagram 2.11)	Printed or displayed message
JCB, FSB, and TUB	Process MODE or OFF command (Diagram 2.12)	Updated JCB, FSB, and TUB
TUB, TCB, and JCB	Process PRIORITY command (Diagram 2.13)	Updated JCB and TCB
Input parameter	Process REPLY command (Diagram 2.14)	User response
TCB, TUB, and JCB	Process START, STOP, or RESTART command (Diagram 2.15)	Updated TCB, TUB, and JCB
TUB, TCB, SCA, and JCB	Process STATUS command (Diagram 2.16)	Status display
Time and date	Process TIME command (Diagram 2.17)	Time and date in command processor work area

To: Cleanup Routine (#CMCU)

Diagram 2.3. Overview of Command Processing

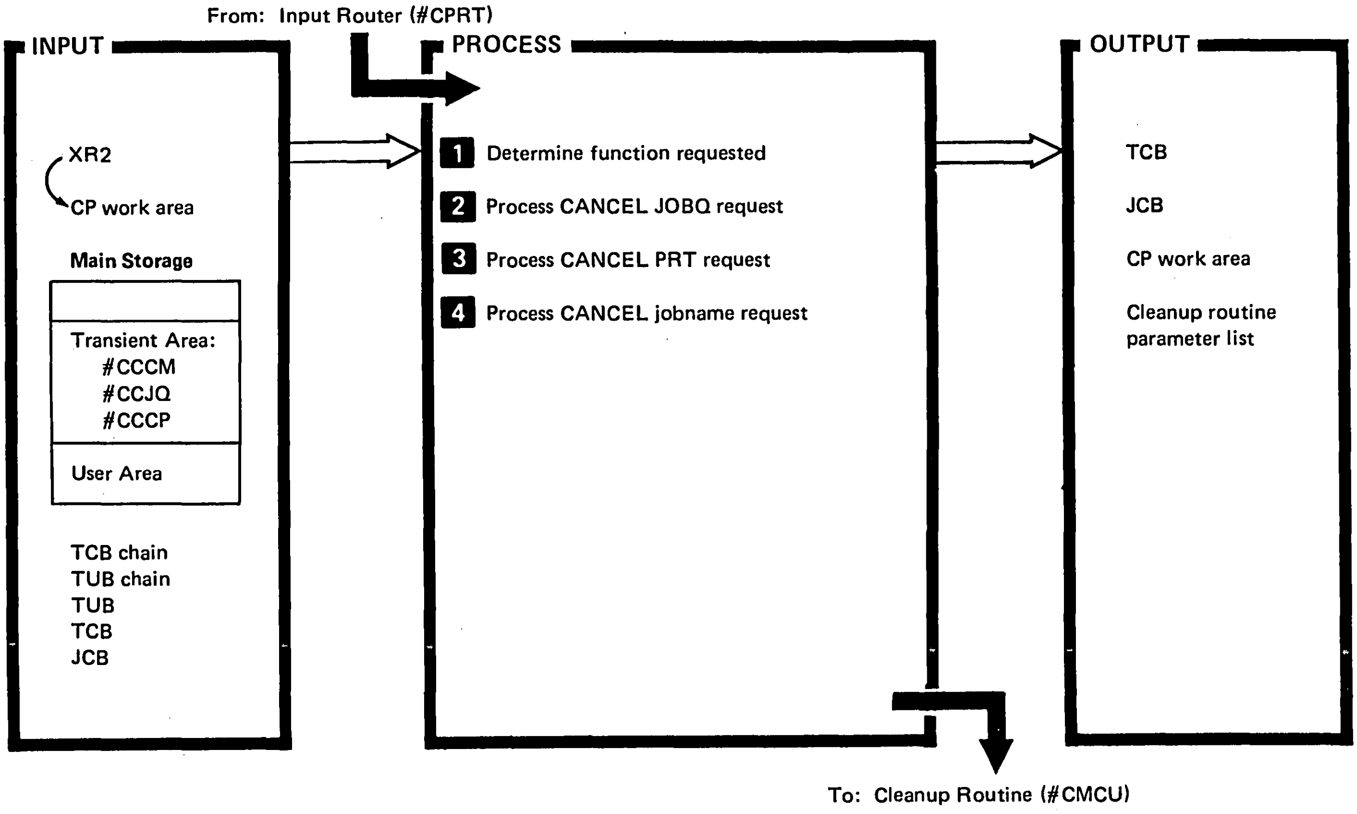


DESCRIPTION	MODULE/ ROUTINE
1 Determine if ASSIGN or VARY command entered. If VARY command, go to **2**. If operand one not PRT, check syntax and switch terminal IDs in terminal unit blocks (TUBs) if no errors exist. If operand one PRT: ● Syntax check operand two. ● Ensure operand two on TUB chain and printer. ● Indicate new printer in system communication area (SCA).	#CCAS
Issue error message if: ● From terminal ID invalid. ● From terminal online. ● To terminal ID invalid. ● To terminal online. ● Operand one printer but operand two not printer.	#CMCU
2 Verify operand two: ● Printer. ● Diskette. ● Work station ID. Verify operand one: ● ON. ● OFF.	#CCAS

Diagram 2.4 (Part 1 of 2). Perform ASSIGN and VARY Command Processing

DESCRIPTION	MODULE/ ROUTINE
If work station ID:	#CCAS
● Verify work station on TUB chain.	
● If VARY ON request:	
– Check TUB and if work station offline, update TUB to indicate work station online.	
– If work station already online, issue message.	#CMCU
● If VARY OFF request:	#CCAS
– Check TUB and if work station online, update TUB to indicate work station offline.	
– If TUB active, issue error message.	#CMCU
– If work station already offline, issue message,	
– If ACE on command processor, complete queue, deque, and free it.	
If VARY PRT request:	#CCAS
● If vary ON request:	
– Check TUB and if printer offline, update TUB to indicate printer online.	
– If printer already online, issue message.	#CMCU
● If VARY OFF request:	
– Check TUB and if printer online, update TUB to indicate printer offline.	#CCAS
– If TUB active, issue error message.	#CMCU
– If printer already offline, issue message.	
If VARY Diskette request:	#CCAS
● If vary ON request:	
– Check UDT and if diskette offline, update UDT to indicate diskette online.	
– If diskette already online, issue message.	#CMCU
● If VARY OFF request:	#CCAS
– Check UDT and if diskette online, update UDT to indicate diskette offline.	
	#CMCU
	#CCAS
– If diskette already offline, issue message.	#CMCU
Build cleanup routine parameter list.	#CCAS
Call and pass control to cleanup routine (#CMCU).	
Note: If errors occur, call #CMCU to issue error message.	

Diagram 2.4 (Part 2 of 2). Perform ASSIGN and VARY Command Processing



From: Input Router (#CPRT)

INPUT

XR2

CP work area

Main Storage

Transient Area:
#CCCM
#CCJQ
#CCCP

User Area

TCB chain
TUB chain
TUB
TCB
JCB

PROCESS

1 Determine function requested

2 Process CANCEL JOBQ request

3 Process CANCEL PRT request

4 Process CANCEL jobname request

OUTPUT

TCB

JCB

CP work area

Cleanup routine parameter list

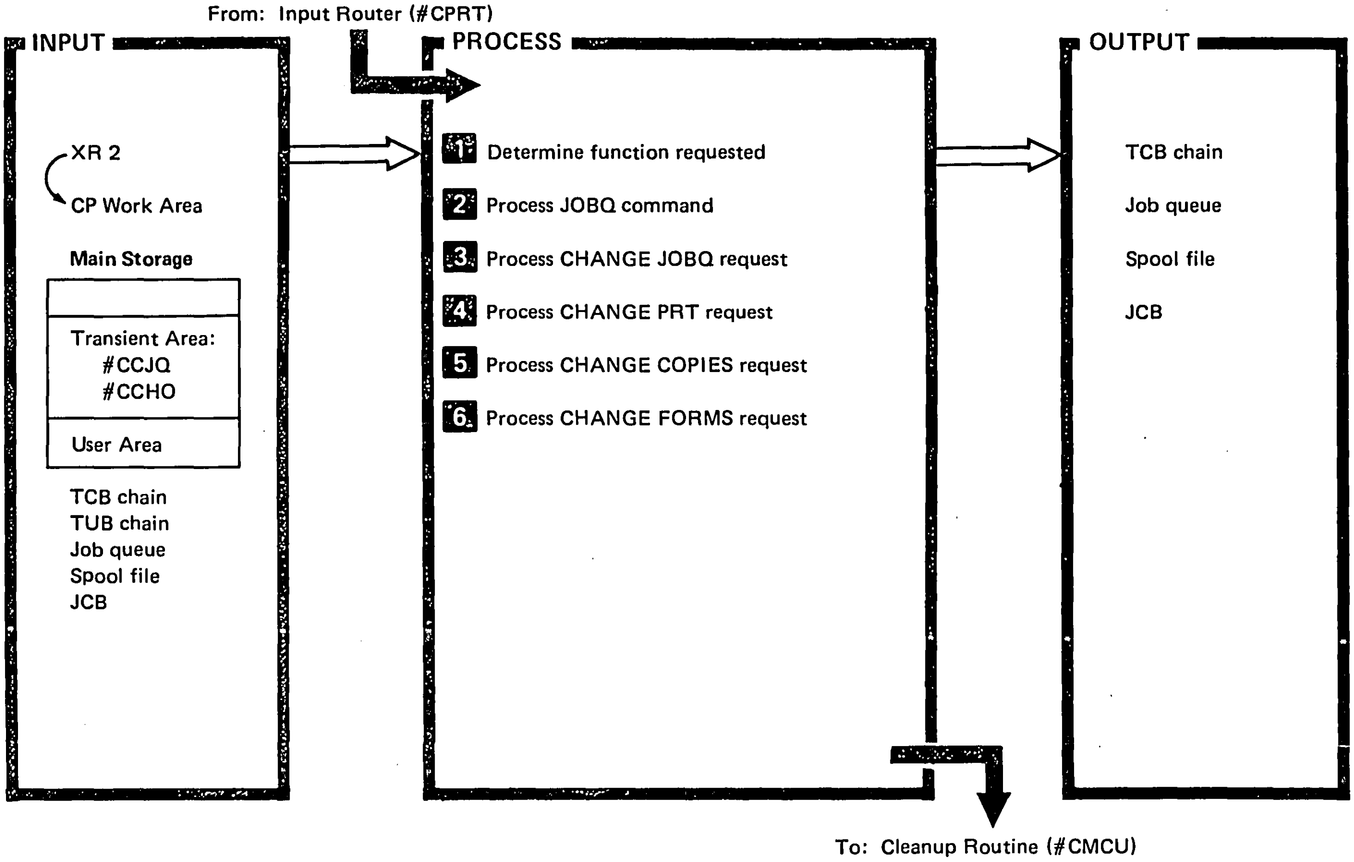
To: Cleanup Routine (#CMCU)

DESCRIPTION	MODULE/ ROUTINE
1 Search for operand 1 in cancel command operand table: ● If cancel JOBQ request go to **2**. ● If CANCEL PRT request, go to **3**. ● If CANCEL jobname, go to **4**. ● If inquiry, go to **4** Ⓐ . ● If error, go to **4** Ⓐ . **2** Verify correct number of operands (if ALL, verify in console mode). Load and pass control to input job queue transient (#CCJQ).	#CCCM
If jobname given: ● Locate specified jobname.	#CCJQ
● Remove specified entry from job queue. ● Chain specified entry to available queue.	FDIOS
● Set message identification code (MIC) to display cancel successful message. ● Call cleanup transient (#CMCU) and exit.	#CCJQ
If ALL given: ● Remove all entries from job queue. ● Chain all entries to available queue.	FDIOS
● Set MIC to display cancel successful message. ● Call cleanup transient (#CMCU) and exit. Verify correct number of operands and in console mode.	#CCJQ

Diagram 2.5 (Part 1 of 2). Perform CANCEL Command Processing

DESCRIPTION	MODULE/ ROUTINE
3 Load and pass control to spool command — cancel transient (#CCCP).	#CCCM
If jobname given: ● Locate specified jobname. ● Ensure OK to cancel job.	#CCCP
● Remove specified entry from print queue. ● Chain entry to available queue. ● Increment number of available spool file block-groups.	FDIOS
● Call cleanup transient (#CMCU) and exit. If jobname not given, but ALL is specified:	#CCCP
● Remove all entries (not executing) from the print queue. ● Chain all entries to available queue. ● Increment number available spool file block-groups.	FDIOS
● Call cleanup transient (#CMCU) and exit.	#CCCP
4 Locate jobname in task control block (TCB) chain. Verify operand length is eight characters. Ensure command issued from console. Ⓐ If no operand or 3 option, indicate 3 option in TCB. If 2 option, indicate 2 option in TCB. If D or DUMP option: ● Set dump indicator in TCB. ● Indicate 3 option in TCB. If not cancelable and not inquiry cancel: ● Indicate cancel pending in TCB. ● Enter job canceled MIC in JCB and indicate termination should display message. ● Call cleanup transient (#CMCU) to display cancel pending at system console and exit. If job in termination and 2 option taken, pend cancel as if not cancelable. Force TCB to call end of job. If inquiry, exit to caller.	#CCCM
If cancel command, issue message to operator indicating job being canceled.	#CMCU

Diagram 2.5 (Part 2 of 2). Perform CANCEL Command Processing

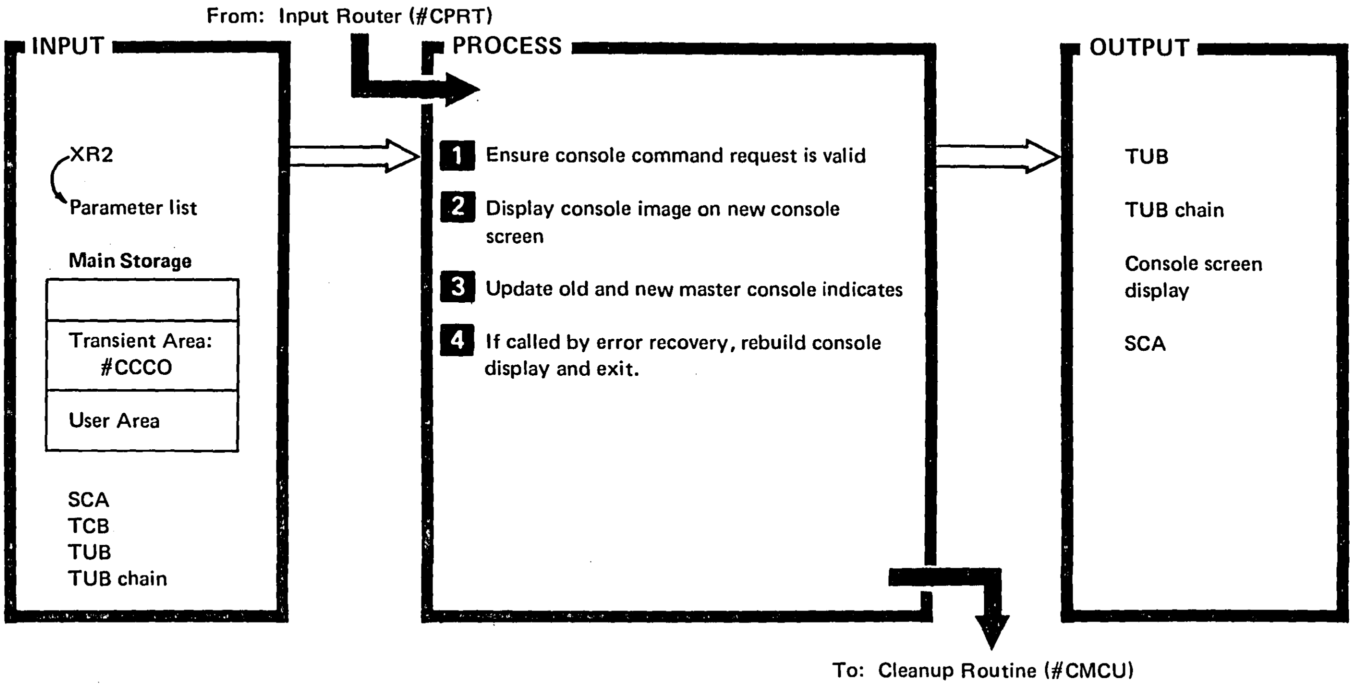

From: Input Router (#CPRT)
INPUT
XR 2
CP Work Area
Main Storage
Transient Area:
#CCJQ
#CCHO
User Area
TCB chain
TUB chain
Job queue
Spool file
JCB
PROCESS
1 Determine function requested
2 Process JOBQ command
3 Process CHANGE JOBQ request
4 Process CHANGE PRT request
5 Process CHANGE COPIES request
6 Process CHANGE FORMS request
OUTPUT
TCB chain
Job queue
Spool file
JCB
To: Cleanup Routine (#CMCU)

DESCRIPTION	MODULE/ ROUTINE
1 Search command routing code for requested function: ● If JOBQ command, go to **2**. ● If CHANGE JOBQ request, go to **3**. ● If CHANGE PRT request, go to **4**. ● If CHANGE COPIES request, go to **5**. ● If CHANGE FORMS request, go to **6**.	#CCJQ
2 Ensure enough room exists on queue for new entry.	
Ensure proc name parameter given.	
Chain new entry to job queue.	Disk IOS
Save any optional parameters supplied.	#CCJQ
Place jobname assigned to new entry in command processor work area.	
Call cleanup routine (#CMCU), pass job name to #CMCU for display, and exit to #CMCU.	
3 Ensure job queue exists.	
Ensure jobname supplied exits.	Disk IOS

Diagram 2.6 (Part 1 of 2). Perform CHANGE and JOBQ Command Processing

DESCRIPTION	MODULE/ ROUTINE
If second jobname(jobname1), chain jobname being changed behind jobname1.	#CCJQ
If second jobname not given or not found, chain jobname being changed to top of job queue.	Disk IOS
Call and pass control to #CMCU.	
4 Ensure jobname specified and exists.	#CCGP
Remove specified jobname from present position on print queue chain.	
If second jobname (jobname1) given, chain jobname being changed behind jobname1 on print queue and set priority of moved jobname equal to priority of jobname1.	Disk IOS
If second jobname not given, chain jobname being changed to top of print queue (first on print queue) and change priority to 5.	
Call #CMCU to display message CHANGE COMMAND SUCCESSFUL and exit.	#CCGP
5 Ensure jobname specified and exists.	
Ensure valid number of copies given.	
Change number of copies in spool file index to new number specified.	Disk IOS
Call #CMCU to display message CHANGE COMMAND SUCCESSFUL and exit.	#CCGP
6 Ensure jobname specified and exists.	#CCJQ
Ensure valid forms number supplied.	
Change forms number in spool file index to new forms number supplied.	Disk IOS
Call #CMCU to display message CHANGE COMMAND SUCCESSFUL and exit.	#CCJQ
Note: If error occurs, call #CMCU to issue error message.	

Diagram 2.6 (Part 2 of 2). Perform CHANGE and JOBQ Command Processing



From: Input Router (#CPRT)

INPUT

XR2

Parameter list

Main Storage

Transient Area: #CCCO

User Area

SCA
TCB
TUB
TUB chain

PROCESS

1 Ensure console command request is valid

2 Display console image on new console screen

3 Update old and new master console indicates

4 If called by error recovery, rebuild console display and exit.

OUTPUT

TUB

TUB chain

Console screen display

SCA

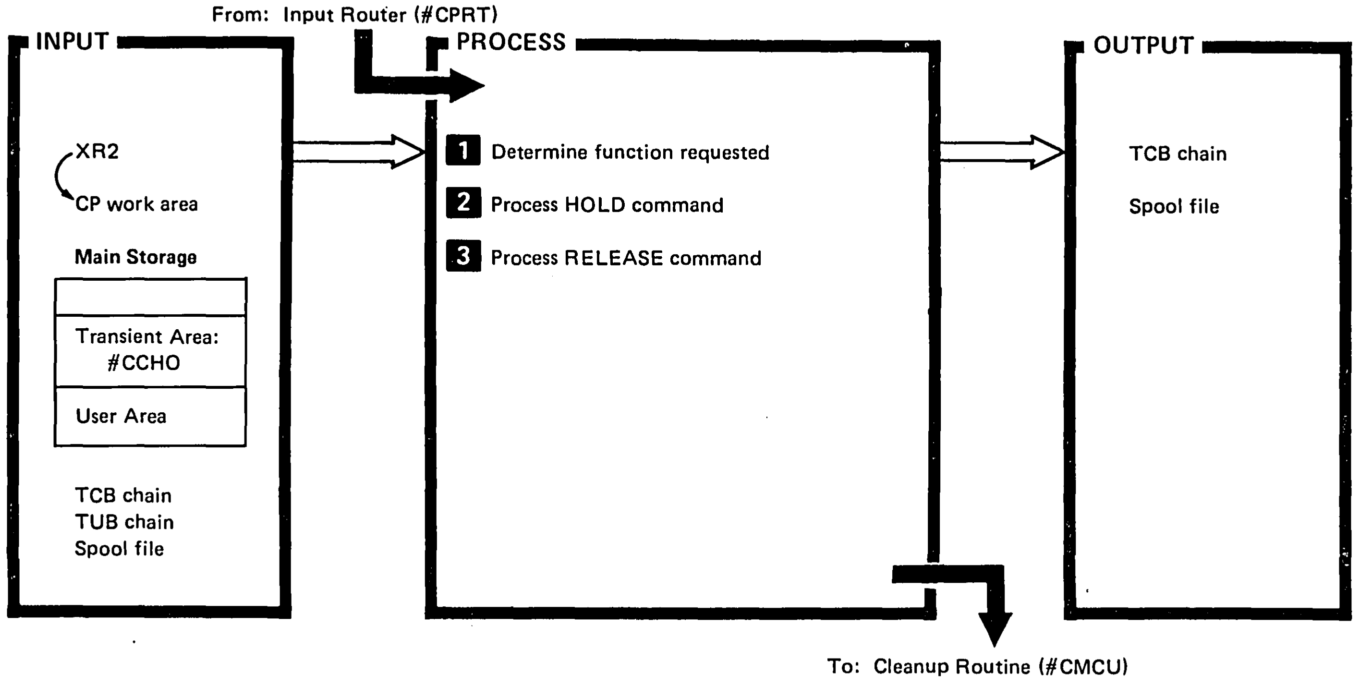
To: Cleanup Routine (#CMCU)

DESCRIPTION	MODULE/ ROUTINE
1 Locate command processor task control block (TCB) and master console terminal unit block (TUB).	#CCCO
If security active on system (SCAMSEC), ensure command user security status is system operator or higher (TUBOPSTS).	
If inquiry active at terminal entering command (TUBINQ1 and TUBINQ2), issue error message (MIC 5635).	
If work station not defined alternate console at configuration time (TUBACN) issue error message (MIC 5636).	
If current console does not have error indicator (TUB$ERR), issue error message (MIC 5638).	
2 Assign space to retrieve screen image from command processor task work area (TWA).	
Blank out assign/free area.	
Build work station data management parameter list.	#CCCO
Display blank console format.	#DWDM
Process system console image matrix: ● Find next image matrix entry requiring response. ● Determine disk sector containing screen data line. ● Read sector from TWA on disk. ● Build work station data management parameter list.	#CCCO
● Display appropriate 80-byte entry on new console screen.	#DWDM
● Repeat steps until all system console image matrix entries displayed.	#CCCO
3 Set off master console indication (TUBMCN) for old console TUBs.	
Set on master console indication (TUBMCN) for new console.	
Indicate new console in console mode (TUBMCNSL).	

Diagram 2.7 (Part 1 of 2). Perform CONSOLE Request Processing

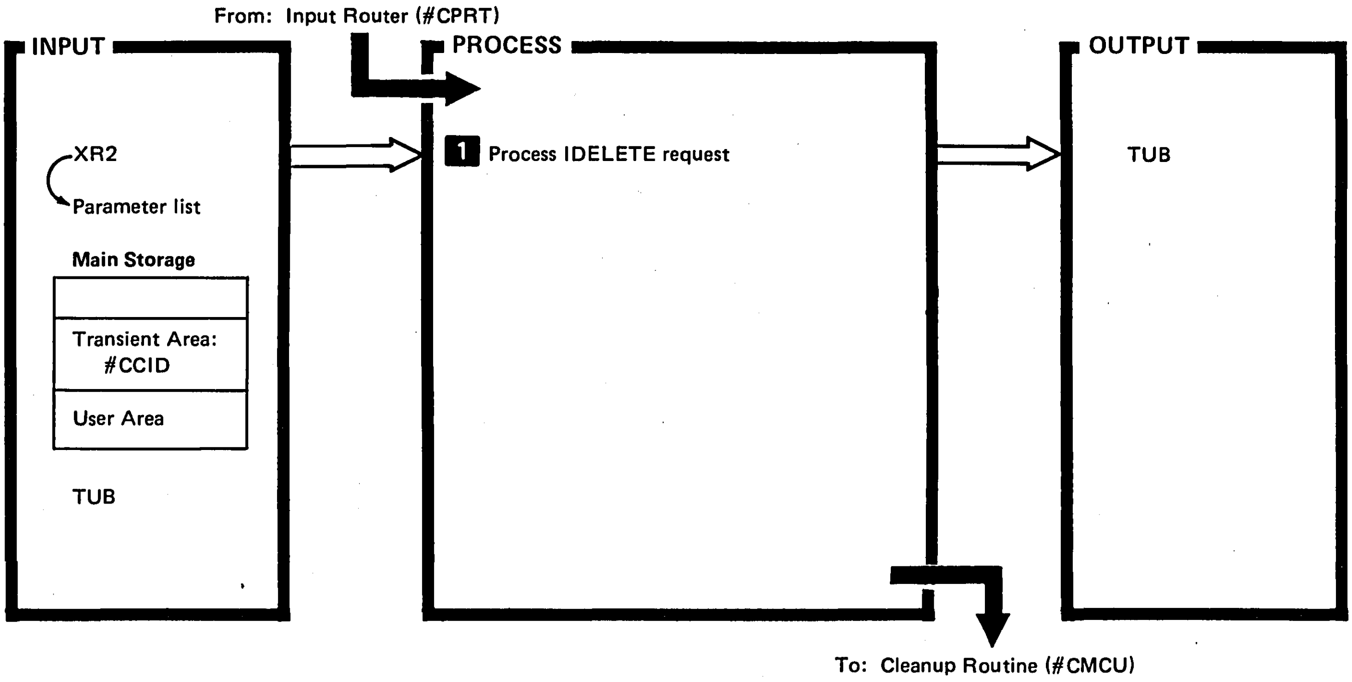
DESCRIPTION	MODULE/ ROUTINE
Turn off console check light.	#CCCO
Create command TUB and chain to console TUB.	
Set up console TUB address in SCA (SCADMTUB).	
Lock on to the TUB chain and free up the assign free space of the original console TUB.	
Rechain any TUBs above the old console TUB to point at horizontal TUB.	
Call error recovery transient to free up any resources associated with a job that was active at the original system console.	#SVERP
Load and pass control to cleanup routine (#CMCU).	#CCCO
Note: If errors occur, call #CMCU to issue error message.	

Diagram 2.7 (Part 2 of 2). Perform CONSOLE Request Processing



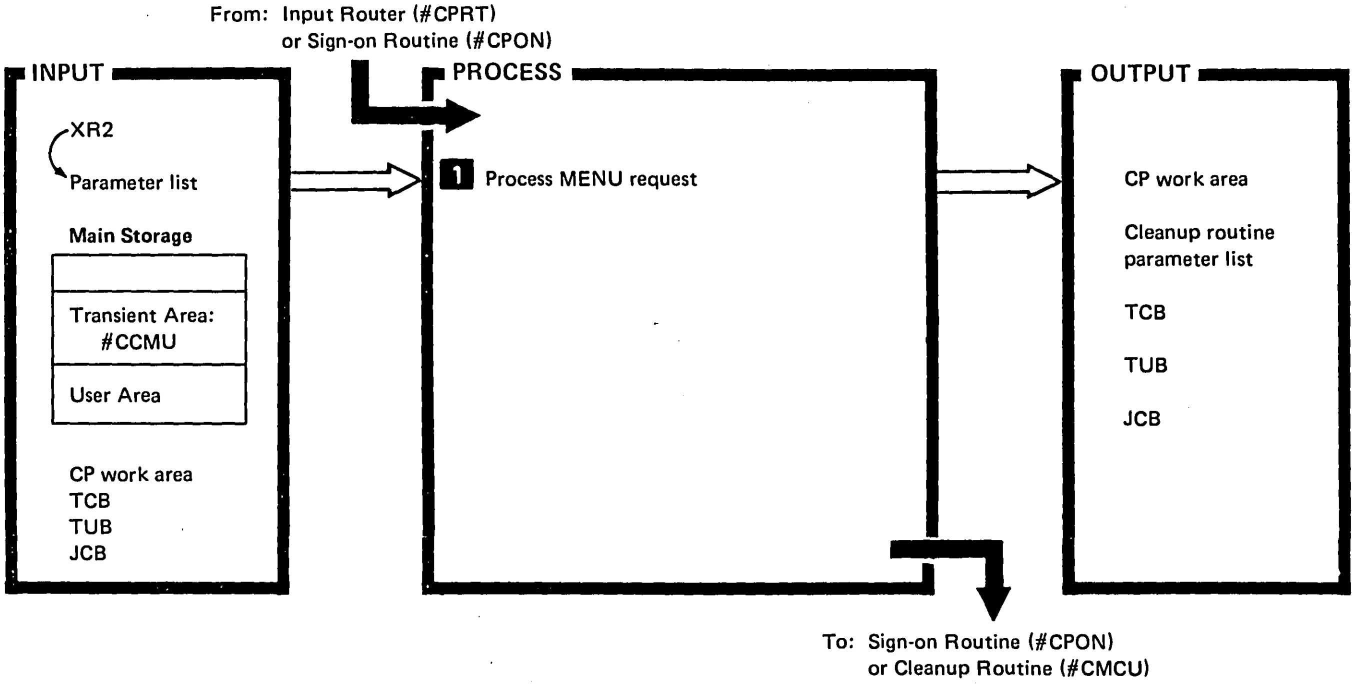
DESCRIPTION	MODULE/ ROUTINE
1 Search command routing code for requested function: ● If HOLD, go to **2**. ● If RELEASE, go to **3**.	#CCHO
2 Check for jobname specified in command.	
If jobname not specified:	
● Set Q-held indicator in master index.	Disk IOS
● Call cleanup routine (#CMCU) to issue message (HOLD COMMAND SUCCESSFUL) and exit.	#CCHO
If jobname specified:	
● Set job ISIN held bit in spool file index to indicate specified jobname held.	Disk IOS
● Call #CMCU to issue message (HOLD COMMAND SUCCESSFUL) and exit.	#CCHO
3 Check for jobname specified in command.	
If jobname not specified:	
● Set off Q-held indicator in master index.	Disk IOS
● Call #CMCU to issue message (RELEASE COMMAND SUCCESSFUL) and exit.	#CCHO
If jobname specified:	
● Set off job is in held bit in spool file index.	Disk IOS
● Call #CMCU to issue message (RELEASE COMMAND SUCCESSFUL) and exit.	#CCHO
If ALLH specified:	
● Set off job is in held bit in spool file index of all jobs currently held.	Disk IOS
● Call #CMCU to issue message (RELEASE COMMAND SUCCESSFUL) and exit.	#CCHO
Note: If error occurs, call #CMCU to issue error message.	

Diagram 2.8. Perform HOLD and RELEASE Command Processing



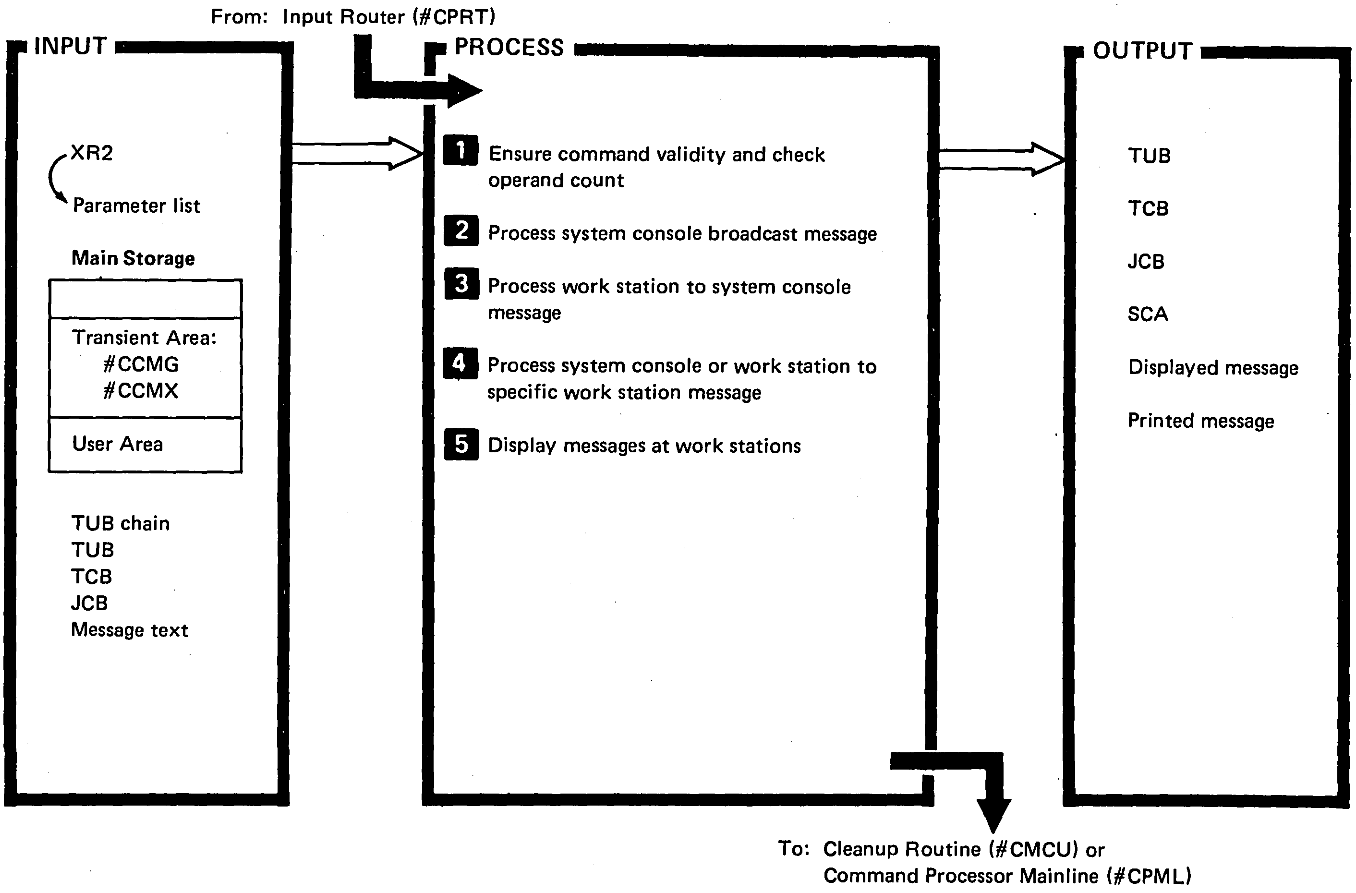
DESCRIPTION	MODULE/ ROUTINE
1 Check parameter 1 for ON or OFF entered.	#CCID
If parameter 1 not ON, OFF, or null, issue error message (MIC 5720).	#CMCU
Find terminal unit block (TUB) address.	#CCID
If OFF entered, indicate IDELETE off in TUB (TUBIDEL).	
If ON or nothing entered, indicate IDELETE on in TUB (TUBIDEL).	
Set up IDELETE successful message (MIC 5721).	
Build cleanup parameter list.	
Load and pass control to cleanup (#CMCU).	

Diagram 2.9. Perform IDELETE Command Processing



DESCRIPTION	MODULE/ ROUTINE
1 Determine command entered.	#CCMU
If cancel menu request (option 0):	
● Build work station data management parameter list.	
● Display command screen.	#DWDM
● Set off menu active bit (TUB MENUA) in TUB.	#CCMU
● Zero disk address in JCB.	
● If menu from #LIBRARY, subtract 1 from user count.	
● Exit.	
Find menu members:	#MASFN
● Menu format member (ensure format valid SFGR format).	
● Menu message member.	
● Verify from same library.	
▶ Retrieve message member:	#CCMU ◀
● Set up job control block (JCB).	
● Set up IOB.	
● Read format member from disk.	Disk IOS
● Indicate menu active (TUBMENUA).	#CCMU
● If menu was active, end previous menu.	
If called from sign-on function, return control to sign-on routine (#CPON).	
If status active (TUBSTATA), pass control to status display (#CCSM) to end status.	
If called by MENU command:	
● Build work station data management parameter list.	
● Display requested menu.	#DWDM
● Pass control to #CMCU.	#CCMU
Note: If errors occur, call #CMCU to issue error message.	

Diagram 2.10. Perform MENU Command Processing


From: Input Router (#CPRT)
INPUT
XR2
Parameter list
Main Storage
Transient Area:
#CCMG
#CCMX
User Area
TUB chain
TUB
TCB
JCB
Message text
PROCESS
1 Ensure command validity and check operand count
2 Process system console broadcast message
3 Process work station to system console message
4 Process system console or work station to specific work station message
5 Display messages at work stations
OUTPUT
TUB
TCB
JCB
SCA
Displayed message
Printed message
To: Cleanup Routine (#CMCU) or
Command Processor Mainline (#CPML)

DESCRIPTION	MODULE/ ROUTINE
1 Check message command entered and issue error messages as necessary: ● If command invalid in console mode, issue message identification code (MIC) 5633. ● If no messages to send, issue MIC 5634. ● If message text missing, issue MIC 5628. ● If message text longer than 60 characters, issue MIC 5627. ● If message is being sent to a device other than a work station, issue MIC 5626. ● If ALL invalid, issue MIC 5625. ● If not enough assign/free space, issue MIC 5702. ● If work station is offline, issue MIC 5623. If error detected: ● Build cleanup routine (#CMCU) parameter list.	#CCMG
● Display error message.	#CMCU
If operand count zero, call message command show routine (#CCMX) to issue messages and go to **5**.	#CCMG
2 Ensure room exists in message queue. Build message in work area. Locate users message queue sector. (TWA SVC 51). Place message in users queue sector.	

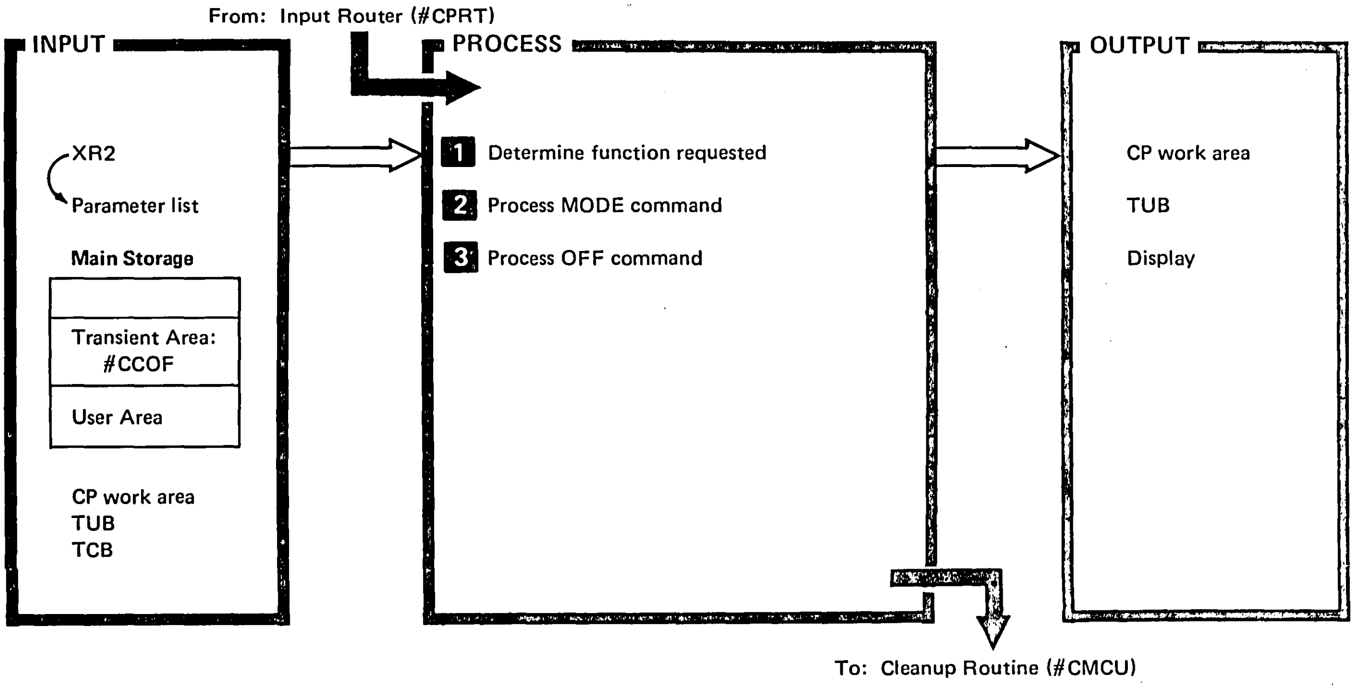
Diagram 2.11 (Part 1 of 3). Perform MSG Command Processing

DESCRIPTION	MODULE/ ROUTINE
Write sector back to disk. (TWA access SVC 51).	#CCMG
Return.	
If spool inactive, system in single program mode, and log printer active, put message to printer.	#CMCU
If unable to find space in user's message queue, issue error message — MIC 5639 and exit to #CMCU.	#CCMG
Sound alarm at each display station online.	WSIOCH
Set up MESSAGE COMMAND SUCCESSFUL message — MIC 5632.	#CCMG
Build cleanup routine (#CMCU) parameter list.	
Display message MIC 5632.	#CMCU
3 Ensure room exists in message queuê.	#CCMG
If unable to find space in console message queue, issue MIC 5629 and exit to #CMCU.	
Build message in work area.	
Locate message queue sector.	
Place message in queue sector.	
Update and write sector back to disk.	
Increment message queue count in SCA.	
Post console SYSLOG (#CMCI).	
Build cleanup routine (#CMCU) parameter list.	
Display message — MIC 5632.	#CMCU
4 Search TUB chain for work station ID matching work station ID entered (WS-ID).	#CCMG
Search TUB chain for user ID if user-ID given.	
If no work station ID or user ID found to match, issue MIC 5630 and exit to #CMCU.	
Save TUB address.	
Initialize message queue sectors associated with specified work station if message count in TUB=zero.	
Build message in work area.	
Locate message queue sector. (TWA request SVC 51)	
If no queue space available, issue MIC 5630 and exit to #CMCU.	
Place message in queue sector.	
Write queue sector back to disk. (TWA request SVC 51)	
Increment message queue count.	
If display station is online, sound alarm at display station.	WSIOCH

Diagram 2.11 (Part 2 of 3). Perform MSG Command Processing

DESCRIPTION	MODULE/ ROUTINE
Build cleanup routine (#CMCU) parameter list.	#CCMG
Display message MIC 5632.	#CCMU
5 Indicate message screen active (TUBMSGA).	#CCMX
Reset broadcast message failure.	
Determine TUB to use.	
▶ Turn off light at display station.	#DWDM ◀
Set on initial call switch.	#CCMX
Build work station data management parameter list and insert work station ID.	
Put screen to proper terminal.	#DWDM
Reset initial call switch.	#CCMX
Check user message queue and if not empty:	
● Find message queue sector. (TWA request SVC 51)	
● Read message from disk.	
● Decrement use count byte in message sector.	
● Write updated message sector back to disk. (TWA request SVC 51)	
● Increment message count on screen.	
● Decrement message Q-count.	
● Build work station data management parameter list.	
● Output messages to proper terminal.	#DWDM
● If spool inactive, system in single program mode, and log printer active: put message to printer.	#CMCU
● Repeat until user message queue empty.	#CCMX
If user message queue empty:	
● Set indicator to restore command screen (TUBRST2) when enter key pressed.	
● Indicate message screen active and more messages to be displayed (TUBMSGA).	
● Load and pass control to CP mainline (#CPML).	

Diagram 2.11 (Part 3 of 3). Perform MSG Command Processing

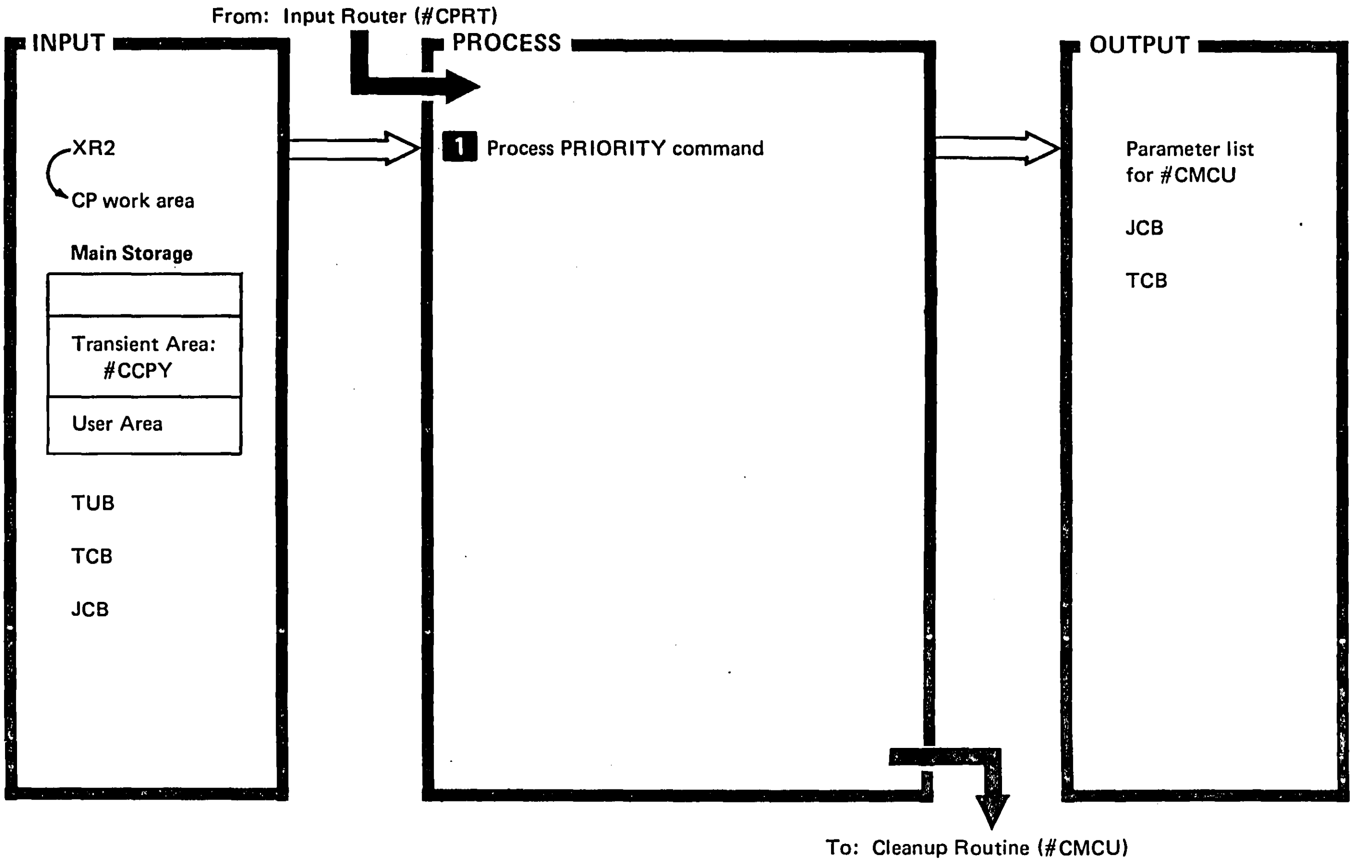


DESCRIPTION	MODULE/ ROUTINE
1 Ensure terminal not in inquiry mode (TUBATTR4).	#CCOF
If menu active and menu from system library, subtract 1 from format 1 user count.	
Search command routing code for requested function:	
● If MODE command, go to **2**.	
● If OFF command or if ERROR OFF command code, go to **3**.	
2 If work station in command mode (TUBATTR2):	
● If status active, call #CCSM to end status.	
● Indicate terminal in standby mode (TUBATTR2).	
● Set up work station data management parameter list.	
● Display standby screen.	#DWDM
● Load and pass control to cleanup routine (#CMCU).	#CCOF
If work station in standby mode:	
● Indicate terminal in command mode (TUBATTR2).	
● Set up work station data management parameter list.	
● Display command screen	#DWDM
● Load and pass control to #CMCU.	#CCOF
3 Indicate terminal not signed on (TUBATTR1).	
If OFF command from command work station:	
● If status active, call #CCSM to end status.	
● Initialize terminal unit block (TUB).	
● If job control block (JCB) pointed to by TUB:	
— If format 1 user count is 1, free format 1 for each file specification block (FSB).	
— If user count greater than 1, decrement user count by 1 for each FSB.	
— Free each FSB.	

Diagram 2.12 (Part 1 of 2). Perform MODE and OFF Command Processing

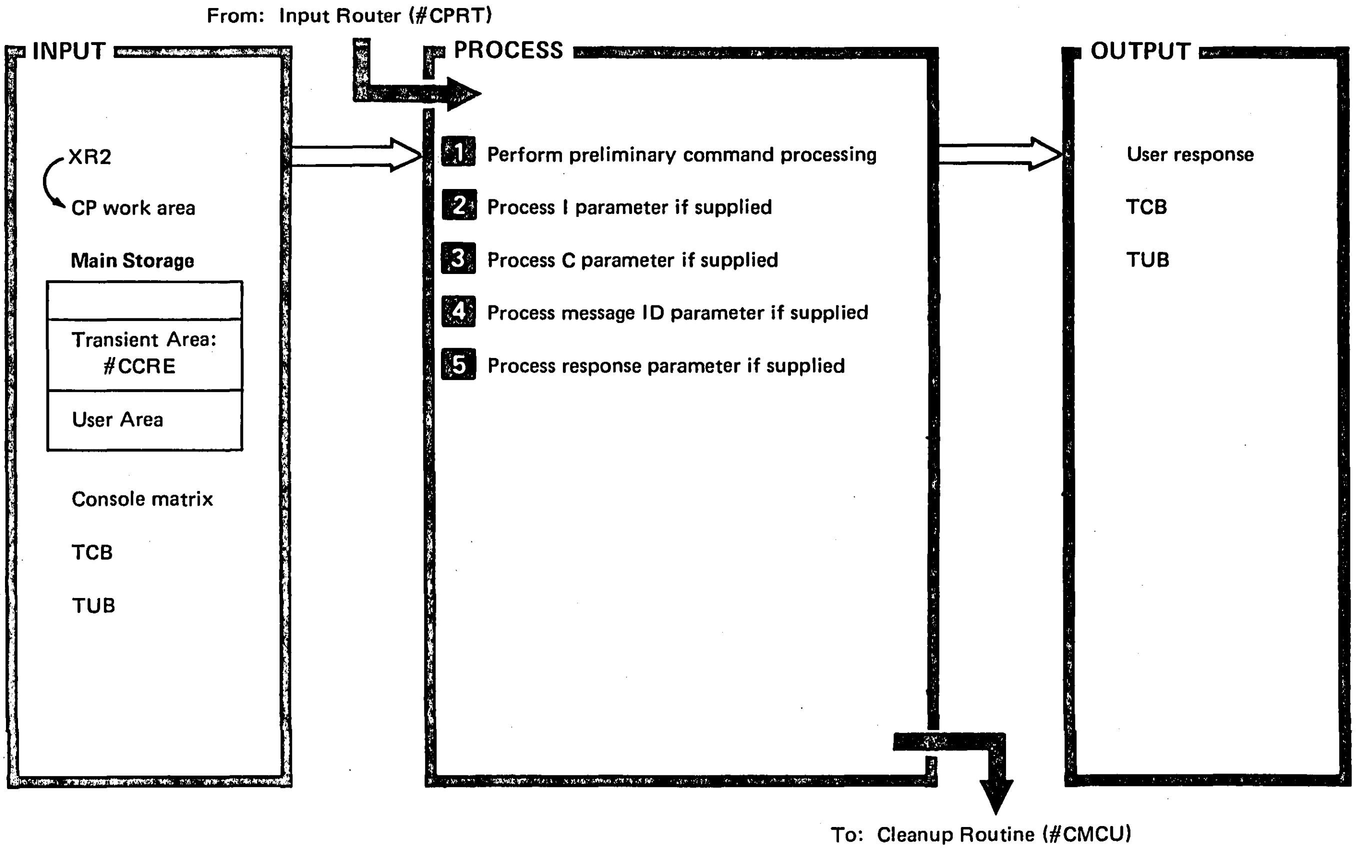
DESCRIPTION	MODULE/ ROUTINE
● Free job control block (JCB).	#CCOF
● Initialize local area to blanks.	
● Read work station configuration record to obatin default library name.	Disk IOS
● If ERROR OFF, clear ERB and exit.	#CCOF
● Insert library name in work station data management parameter list.	#DWDM
● Turn off the message waiting light.	
● Display command work station sign-on display.	
● If test request call, set off test request and exit.	
● Load and pass control to #CMCU.	#CCOF
If OFF command from data work station:	
● Initialize TUB.	
● Build work station data management parameter list.	
● Display data work station sign-on display.	#DWDM
● Load and pass control to #CMCU.	#CCOF
Note: If errors occur, call #CMCU to issue error message.	

Diagram 2.12 (Part 2 of 2). Perform MODE and OFF Command Processing



DESCRIPTION	MODULE/ ROUTINE
1 Determine if command entered from console or work station (TUBATTR2). If command entered from work station: ● Set priority indicator in job control block (JCBDSCH2). ● Load and pass control to cleanup routine (#CMCU) to display message (PRIORITY ACCEPTED). If command entered from console: ● Ensure valid jobname supplied. ● Find JCB for specified job. ● Verify ON/OFF operand if supplied. ● If ON or not specified, set priority indicator in JCB (JCBDSCH2) and set TCB (TCBPRIOR) priority to high. ● If OFF requested, set off priority indicator in JCB and set TCB priority to low. *Note:* If errors occur, call #CMCU to issue error message.	#CCPY

Diagram 2.13. Perform PRTY Command Processing



From: Input Router (#CPRT)

INPUT

XR2

CP work area

Main Storage

Transient Area: #CCRE

User Area

Console matrix

TCB

TUB

PROCESS

1 Perform preliminary command processing

2 Process I parameter if supplied

3 Process C parameter if supplied

4 Process message ID parameter if supplied

5 Process response parameter if supplied

OUTPUT

User response

TCB

TUB

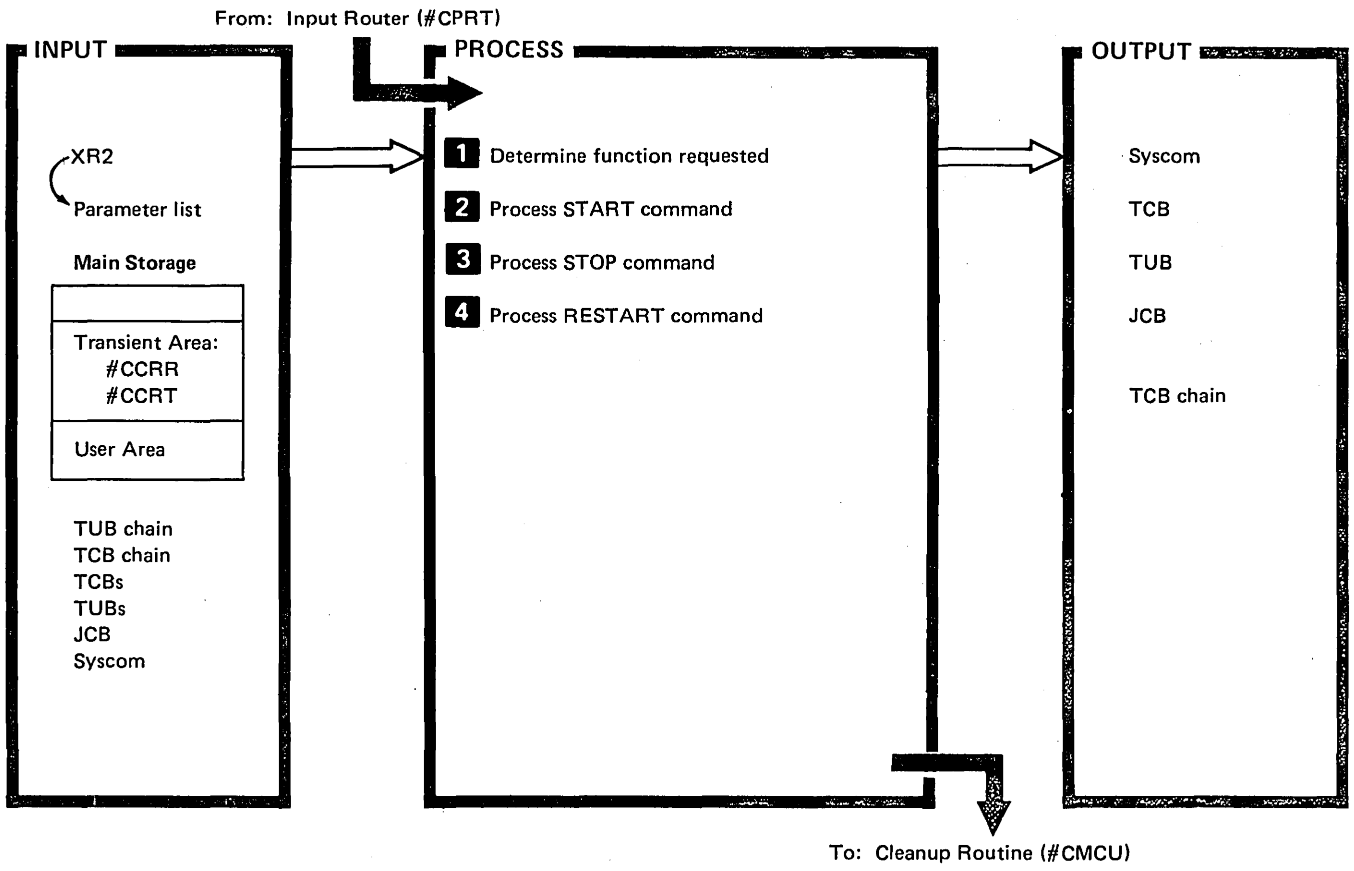
To: Cleanup Routine (#CMCU)

DESCRIPTION	MODULE/ ROUTINE
1 Determine if status display on display screen (TUBATTR2).	#CCRE
Issue error message if REPLY command and status display present (MIC 5616).	#CMCU
Scan system console image matrix for message response from end of job (EOJ) (CMCISWH).	#CCRE
If response from EOJ, put ** over message IDs.	#CMEJ
If no message ID entered (first parameter null) or message ID greater than two characters, issue error message (MIC 5617).	#CMCU
2 Determine if I parameter entered. If entered: ○ Scan image matrix for throw-response-away bit (CMCITHR). Ⓐ ○ Indicate response given (CMCIREP).	#CCRE
○ Put ** over message ID.	#CMEJ
○ Display successful message (MIC 5615).	#CMCU
○ If number of available lines is five or more, issue messages still pending. ○ Pass control to cleanup routine (#CMCU).	#CMCI
3 If C parameter entered:	#CCRE
○ Rearrange matrix so all messages still in need of a response are at the end of the matrix.	#CMEJ
○ Roll down screen and clear rolled lines.	#DWDM
○ Display successful message (MIC 5615).	#CMCU
○ If number of available lines is five or more, issue messages still pending. ○ Pass control to cleanup routine (#CMCU).	#CMCI

Diagram 2.14 (Part 1 of 2). Perform REPLY Command Processing

DESCRIPTION	MODULE/ ROUTINE
4 Syntax check message ID.	#CCRE
Issue syntax error message (MIC 5617) if necessary.	#CMCU
Search system console image matrix for message ID match.	#CCRE
Issue error message (MIC 5618) if no match found. If match found is throw-away-response, go to **2** **A**.	#CMCU
5 Blank out response data area.	#CCRE
If nonresponse entered for operand 2 and,	#CMCU
if SYSLOG halt, call #CCRS to handle second level message:	#CCRE
● Read console image sector. ● Extract MIC number from image.	#CCRS
▶ ● Retrieve second level message from user message member.	#MGRET ◀
● Display reply successful message.	#CMCU
● Save console screen. ○ Display second level message.	#DWDM
▶ ● Indicate restore console interface on next keystroke. ● Pass control to cleanup routine (#CMCU). If operand 2 is a response: ● Verify operand 2 length less than or equal to caller requested option length.	#CCRE ◀
● Issue invalid length message (MIC 5619) if necessary.	#CMCU
● If SYSLOG halt: — Verify option entered is permitted.	#CCRE
— Issue option response error message (MIC 5620) if necessary.	#CMCU
— If response on, go to **2** **A**. Handle response not previously processed: ● Move response to task work area. ● Pass response to user. ● Post task complete. ● Pass control to #CMCU. *Note:* If errors occur, call #CMCU to issue error message.	#CCRE

Diagram 2.14 (Part 2 of 2). Perform REPLY Command Processing



DESCRIPTION	MODULE/ ROUTINE
1 Search for command in command routing code: ● If START command, go to **2**. ● If STOP command, go to **3**. ● If RESTART command, go to **4**.	#CCRT
2 Search for operand 1 in start command table.	
If operand 1 invalid, issue error message.	#CMCU
If operand 1 is PRT or P, load spool transient (#CCRR):	#CCRT
● If spool not supported or writer active, issue error message.	#CMCU
● If writer not active: – Set forms number in class/page number area (BPCLSS) if form number given. – Create job control block (JCB) for writer.	#CCRR
– Attach spool writer. – Issue start successful message (MIC 5664).	#SVAT
If operand 1 is WORKSTN or W:	#CCRT
● If operand 2 is missing or invalid, issue error message (MIC 5667).	#CMCU
▶ ● If operand 2 is ALL: – Set start initiation flag in terminal unit block (TUB) for all work stations.	#CCRT ◀
– Issue start successful message (MIC 5664).	#CMCU

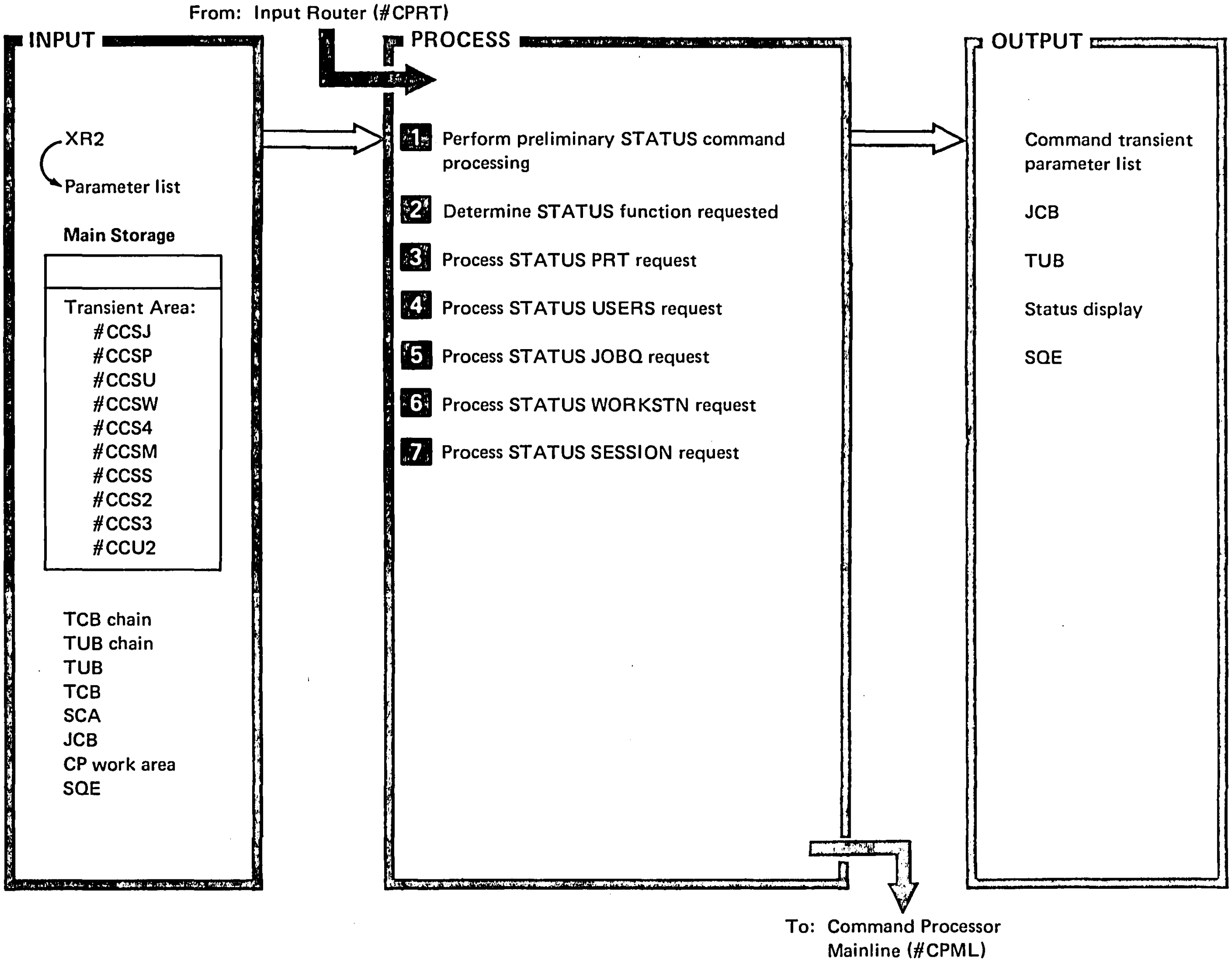
Diagram 2.15 (Part 1 of 3). Perform START, STOP, or RESTART Command Processing

DESCRIPTION	MODULE/ ROUTINE
● If operand 2 is work station ID: — Find specified work station on terminal unit block (TUB) chain. — Set start initiation flag for that specific work station.	#CCRT
— Issue start successful message (MIC 5664).	#CMCU
If operand 1 is JOBQ or J, load job queue transient (#CCJS):	#CCRT
● Ensure dedicated program not running or Job queue not empty.	#CCJS
● Issue error message if necessary.	#CMCU
	#CCJS
● Take first entry of job queue.	Disk IOS
● Indicate program to be loaded.	#CCJS
● Find user library if given.	#MAFLB
● Attach job.	#SVAT
● If unable to attach job because resources not available:	#CCJS
— Put entry back on top of queue.	Disk IOS
— Return.	#CCJS
— Issue error message (MIC 5691). ● If start successful, issue message (MIC 5685).	#CMCU
If operand 1 is SYSTEM or S:	#CCRT
● If there are two operands, issue error message (MIC 5541).	#CMCU
● Verify that a start system can now occur. ● Set flag in SCA (SCAMALL) to indicate all initiation started. ● Restart spool writer and job queue if they were active at the time stop system was issued. ● Start work stations.	#CCRT
If operand 1 is JOB:	
● If operand 2 is missing or invalid, issue error message (MIC 5567).	#CMCU
● If operand 2 is ALL: — Find task control block (TCB) chain. — Ensure task suspended by system operator. — Set off suspended flag (TCBSDTSO). — Resume user tasks.	#CCRT
— Issue start successful message (MIC 5664).	#CMCU
● If operand 2 is jobname: — Ensure valid jobname. — Find task control block (TCB) for specified jobname. — Ensure task suspended by system operator. — Set off suspended flag (TCBSDTSO). — Resume user task.	#CCRT
— Issue start successful message (MIC 5664).	#CMCU
3 Search for operand 1 in stop command operand table.	#CCRT
If operand 1 invalid, issue error message.	#CMCU
If operand 1 is PRT or P, load spool transient (#CCRR):	#CCRT
● If spool not supported or writer previously stopped, issue error message.	#CMCU
● Set stop writer flag (BPFLAG).	#CCRR
● Issue message Spool Writer Stopped (MIC 5663).	#CMCU
If operand 1 is WORKSTN or W:	#CCRT
● If operand 2 is missing or invalid, issue error message (MIC 5667).	#CMCU
● If operand 2 is ALL: — Set off initiation flag in TUB for all work stations.	#CCRT
— Issue stop successful message (MIC 5663).	#CMCU
● If operand 2 is work station ID: — Find specified work station on TUB chain. — Set off start initiation flag in TUB for specific work station.	#CCRT
— Issue stop successful message (MIC 5663).	#CMCU

Diagram 2.15 (Part 2 of 3). Perform START, STOP, or RESTART Command Processing

DESCRIPTION	MODULE/ ROUTINE
If operand 1 is JOBQ or J, load job queue transient (#CCJS):	#CCRT
● Ensure dedicated program not running. ● Indicate job queue stopped in SCA.	#CCJS
▶ ● Issue stop successful message.	#CMCU ◀
If operand 1 is SYSTEM or S: ● Ensure stop all indication flag (SCAMALL) not already set. ● Set stop all indication flag (no new JOBQ or spool tasks will be started). ● Post for input op-end, all never ending programs that have an invite count of zero for shut down inquiry. ● Stop work stations.	#CCRT
● If operand 2 is SORT, or not given, perform keysort on all files that have the sort or merge bits on in the F1 (F1AMSORT, F1AMMRGE).	#DDKAA
● Issue stop successful message (MIC 5663).	#CMCU
If operand 1 is JOB:	#CCRT
● If operand 2 is missing or invalid, issue error message (MIC 5667).	#CMCU
● If operand 2 is ALL: – Find TCB chain. – Set task suspended indicators (TCBSDTSO). – Suspend user tasks.	#CCRT
– Issue stop successful message (MIC 5663).	#CMCU
● If operand 2 is jobname: – Ensure valid jobname. – Find TCB for specified jobname. – Set task suspended indicator (TCBSDTSO). – Suspend user task.	#CCRT
– Issue stop successful message (MIC 5663).	#CMCU
4 Load and pass control to spool transient (#CCRR).	#CCRT
Ensure spool supported. If spool writer active: ● Set page number to value given in operand 2 or set to zero if no operand supplied. ● Set flag (BPFLAG) to restart writer.	#CCRR
● Issue spool writer restarted message (MIC 5703).	#CMCU
If spool writer not active: ● Set page number to value given in operand 2 or set to zero if operand 2 not supplied. ● Set flag (PBFLAG) to restart writer.	#CCRR
● Attach spool writer.	#SVAT
● Issue restart successful message (MIC 5703).	#CMCU
Note: If errors occur, call #CMCU to issue error message.	

Diagram 2.15 (Part 3 of 3). Perform START, STOP, or RESTART Command Processing



From: Input Router (#CPRT)

INPUT

XR2

Parameter list

Main Storage

Transient Area:
#CCSJ
#CCSP
#CCSU
#CCSW
#CCS4
#CCSM
#CCSS
#CCS2
#CCS3
#CCU2

TCB chain
TUB chain
TUB
TCB
SCA
JCB
CP work area
SQE

PROCESS

1 Perform preliminary STATUS command processing

2 Determine STATUS function requested

3 Process STATUS PRT request

4 Process STATUS USERS request

5 Process STATUS JOBQ request

6 Process STATUS WORKSTN request

7 Process STATUS SESSION request

OUTPUT

Command transient parameter list

JCB

TUB

Status display

SQE

To: Command Processor Mainline (#CPML)

DESCRIPTION	MODULE/ ROUTINE
1 Save portion of command processor work area in appropriate request block (RB). Examine command routing code (CPCODE) for end status request. If end status request: ● Find associated status queue element (SQE). ● If SQE not found, exit program. ● Free and dequeue SQE and exit. If status not active, go to **2**. If status active: ● If SQE not found, abnormally terminate status. ● If E page control character entered: – Restore previously saved display if work station in console mode.	#CCSM
– Call console log processor if work station in console mode.	#CMCI
– Return.	#CCSM

Diagram 2.16 (Part 1 of 6). Perform STATUS Command Processing

DESCRIPTION	MODULE/ ROUTINE
– Display menu if menu active and work station not in console mode. – Display command display if menu not active and work station not in console mode.	#DWDM
– End status (TUBATTR2). – Dequeue and free SQE. – Exit program. ● If I page control character entered: – Pass input information tp #CMCU for logging to history file. – Go to 2 Ⓐ. ● If other page control character entered: – Determine which status was active in SQE (STATOPC): – If P, go to 3 Ⓐ. – If U, go to 4 Ⓐ. – If J, go to 5. – If W, go to 6. – If S and page 1 or 2 to be displayed, pass control to #CCSS 7. – If S and page 3 or 4 to be displayed, pass control tp #CCS3 7 Ⓑ. – If none of the above, abnormally terminate status. 2 Assign and queue an SQE.	#CCSM
Log input to history file.	#CMCU
	#CCSM
If work station in console mode, save console display.	#DWDM
Ⓐ If no operand 1 given, go to 7. Search for operand 1 in operand table and check number of operands: ● If P or PRT, go to 3. ● If U or USERS, go to 4. ● If J or JOBQ, go to 5. ● If W or WORKSTN, go to 6. ● If S or SESSION, go to 7. 3 Check if in console mode. Ⓐ Restore CPWRK. Route control to #CCSP.	#CCSM
Ensure spool supported. Save portion of work area. Assign space for IOB, FDT, and text sectors of format and execution time data area. Free all but execution time data area. Check if first call or F, R, or U page control character.	#CCSP
Read spool file master index. Read entries from spool file.	Disk IOS
Locate entries to display.	#CCSP

Diagram 2.16 (Part 2 of 6). Perform STATUS Command Processing

DESCRIPTION	MODULE/ ROUTINE
Format entries to display.	#CCSP
Set up work station parameter list to display entries.	
Display spool print queue entries.	#DWDM
Free output area.	#CCSP
Restore work area.	
Load and pass control to command processor mainline (#CPML).	
4 Check if in console mode.	#CCSM
A Restore CPWRK.	
Route control to #CCSU.	
Save portion of CPWRK area.	#CCSU
Assign space for IOB, Format FDT sectors, text sectors, and execution time data areas; free all but execution time data areas.	
If operand 2 given, start function with job specified by operand 2.	
If R option, start with first TCB on TCB chain.	
If U option, start with TCBTSKID specified in SQE (IDCURRENT).	
If F or other option, start with TCBTSKID specified in SQE (IDFORW).	
If no users active:	
● If status users was active, display user's display with no jobs.	#DWDM
● If status users was not active, call #CMCU to display error message.	#CCSU
Restore CPWRK.	
Route control to #CCU2.	#CCU2
Call message retrieve to retrieve translatable constants (if retrieve fails, use English version).	#MGRET
Save part of CPWRK.	#CCU2
If JOBQ TCB exists and is requested, or if display is for page 1: ● If operand 2 points to this task, display this TCB only. ● If no operand 2, create this line.	
Create lines until display is full or end of TCBs is reached beginning with TCB pointed to by #CCSU.	
If end of TCBs is reached and no lines have been built, restart from first TCB and build lines.	
If end of TCBs is reached and there are lines to be displayed, display them.	#DWDM
Display first half of display (if this is automatic status update, display format that will not clear input areas).	
Display second half of display.	
Point SQE at next TCB.	#CCU2
Free output area.	

Diagram 2.16 (Part 3 of 6). Perform STATUS Command Processing

DESCRIPTION	MODULE/ ROUTINE
Restore CPWRK.	#CCU2
Exit program.	
5 Restore CPWRK.	#CCSM
Route control to #CCSJ.	
Ensure job queue supported.	#CCSJ
Save portion of work area.	#CCSM
Assign space for IOB, FDT, AND text sectors of format and execution time data area.	
Free all but execution time data area.	
Check if first call and if forward (F), reset (R), or update (U) page control character entered: ● Ensure job queue supported.	#CCSJ
● Read job queue file index sector.	Disk IOS
● If status not active or R entered, set first entry on job queue as start of display.	#CCSJ
● If no page control character entered, assume forward (F) character and chain through queue to find entry by position number.	Disk IOS
Ensure queue not empty.	
If job name given, find entry requested.	#CCSJ
Read entry into buffer.	Disk IOS
If command from work station, ensure user ID and work station ID match.	#CCSJ
Format requested display in output buffer: ● If work station request, work station ID and user ID must match. ● If system console request, format all input job queue entries.	
Build work station data management parameter list.	
Display job queue entries.	#DWDM
Free output area and restore work area.	#CCSJ
Load and pass control to #CPML.	
6 Restore CPWRK.	#CCSM
Route control to #CCSW.	
Save portion of work area.	#CCSW
Assign space for IOB, FDT and text sectors of format, and execution time data area.	
Call message retrieve to retrieve translatable constants.	
Free all but execution time data area.	
If operand 2 given, start function with work station given in operand 2.	
If R control character, start with first TUB.	
If U control character, start with current work station in SQE.	

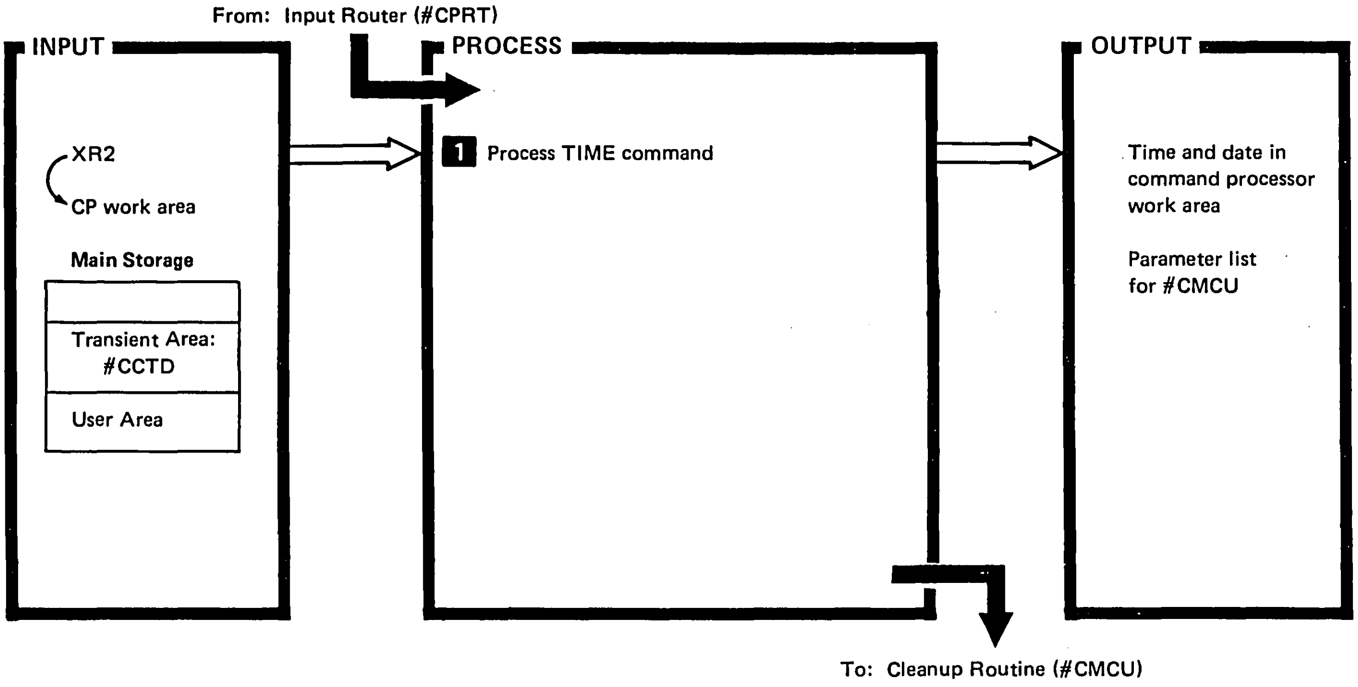
Diagram 2.16 (Part 4 of 6). Perform STATUS Command Processing

DESCRIPTION	MODULE/ ROUTINE
If F or other control character, start with forward work station in SQE. Build line of output for each device.	#CCSW
At end of page, display page (if automatic status, display format that does not clear input).	#DWDM
Free output area. Restore work area. Exit program.	#CCSW
7 Ensure not in console mode. Restore CPWRK. Route control to #CCSS.	#CCSM
Save portion of work area. If I option, start page 1. If R option, start page 1. If U option, start from current page pointer in SQE. If F option, start from forward page pointer in SQE. If start page = 2: ● Restore CPWRK. ● Route control to #CCS2 7 A. Assign space for IOB, format text and FDT sectors, and execution time data area.	#CCSS
Call message retrieve to retrieve translatable fields and place in transient area (if retrieve fails, use English version).	#MGRET
Free all but execution time data area. Build page 1 information. Set up SQE for current page = 1 and forward = 2.	#CCSS
Call work station data management to display page 1 (if automatic status, display format that will not clear input lines).	#DWDM
Free execution time data area. Restore CPWRK. Exit program.	#CCSS
A Save 30 bytes of CPWRK. Assign enough space for format FDT and text sectors, IOB, and execution time data area. Call message retrieve to retrieve translatable fields and place in work area (if message retrieve fails, use English version). Free all but execution time data area.	#CCS2

Diagram 2.16 (Part 5 of 6). Perform STATUS Command Processing

DESCRIPTION	MODULE/ ROUTINE
Build page 2.	#CCS2
Set up SQE current = 2, forward = 3.	
Call work station data management to display page 2 (if automatic status, display format that will not clear input lines).	#DWDM
Restore CPWRK.	#CCS2
Exit program.	
B Save 30 bytes of work area.	#CCS3
If U option, begin function with page in SQE.	
If F option, begin with forward page in SQE.	
If page 4, go to 7 C.	
Assign space for format FDT and text sectors, IOB, and execution time data areas.	
Call message retrieve to retrieve translatable fields (if retrieve fails, use English verison).	#MGRET
Free all but execution time fields.	#CCS3
Build page 3 data.	
Indicate in SQE forward page = 4, current = 3.	
Call work station data management to display page 3 (if automatic status update, do not clear input lines).	#DWDM
Restore CPWRK.	#CCS3
Exit program.	
C Assign space for format FDT and text sectors, IOB, and execution time data areas.	
Store assign area address in WDRECA into CPWRK.	
Call #CCS4 to build execution time data.	
Build execution time data for page 4.	#CCS4
Free all but execution time data area.	#CCS3
Point SQE forward to 1, current = 4.	
Call work station data management to display format (if automatic status update, do not clear input lines).	#DWDM
Exit program.	#CCS3
Notes: 1. If errors occur, call #CMCU to issue error message. 2. If SQE is new, dequeue and free SQE.	

Diagram 2.16 (Part 6 of 6). Perform STATUS Command Processing



DESCRIPTION	MODULE/ ROUTINE
1 Initialize timer request block. Get time of day and date and place in command processor work area. Build parameter list for #CMCU. Pass control to #CMCU to display time and data. *Note:* If errors occur, call #CMCU to issue error message.	#CCTD

Diagram 2.17. Perform TIME Command Processing

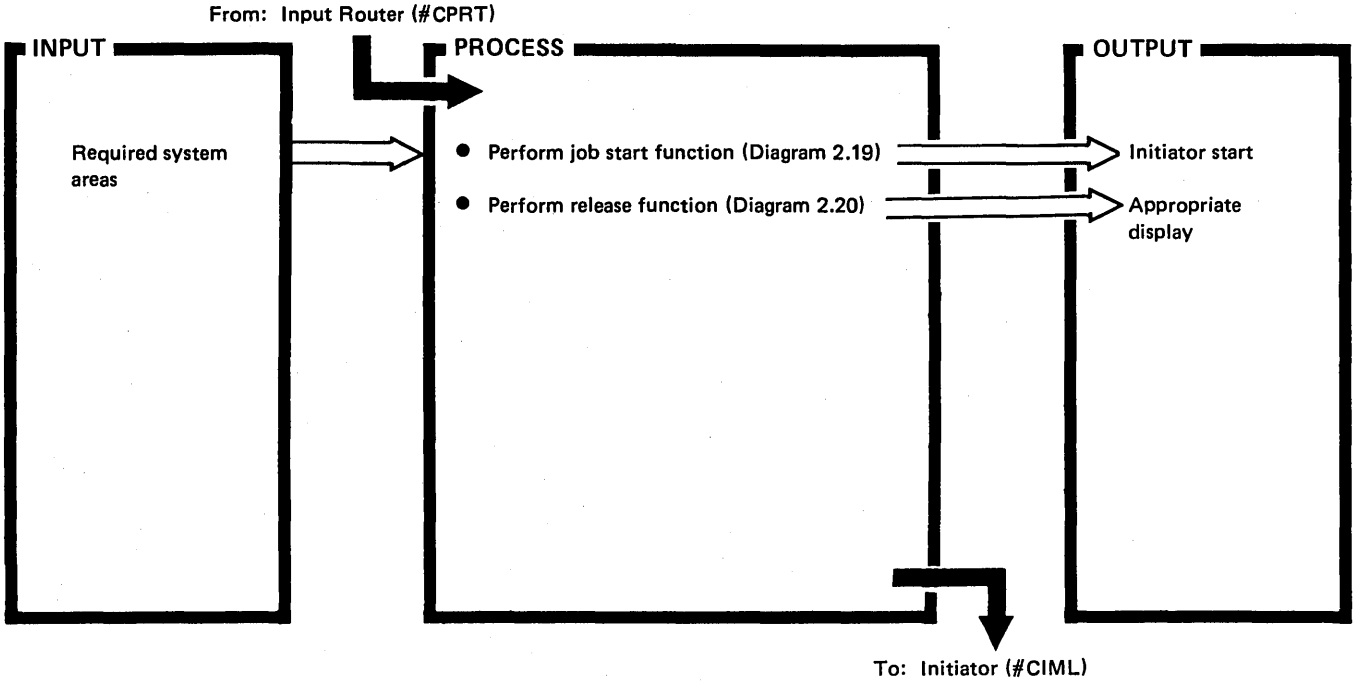
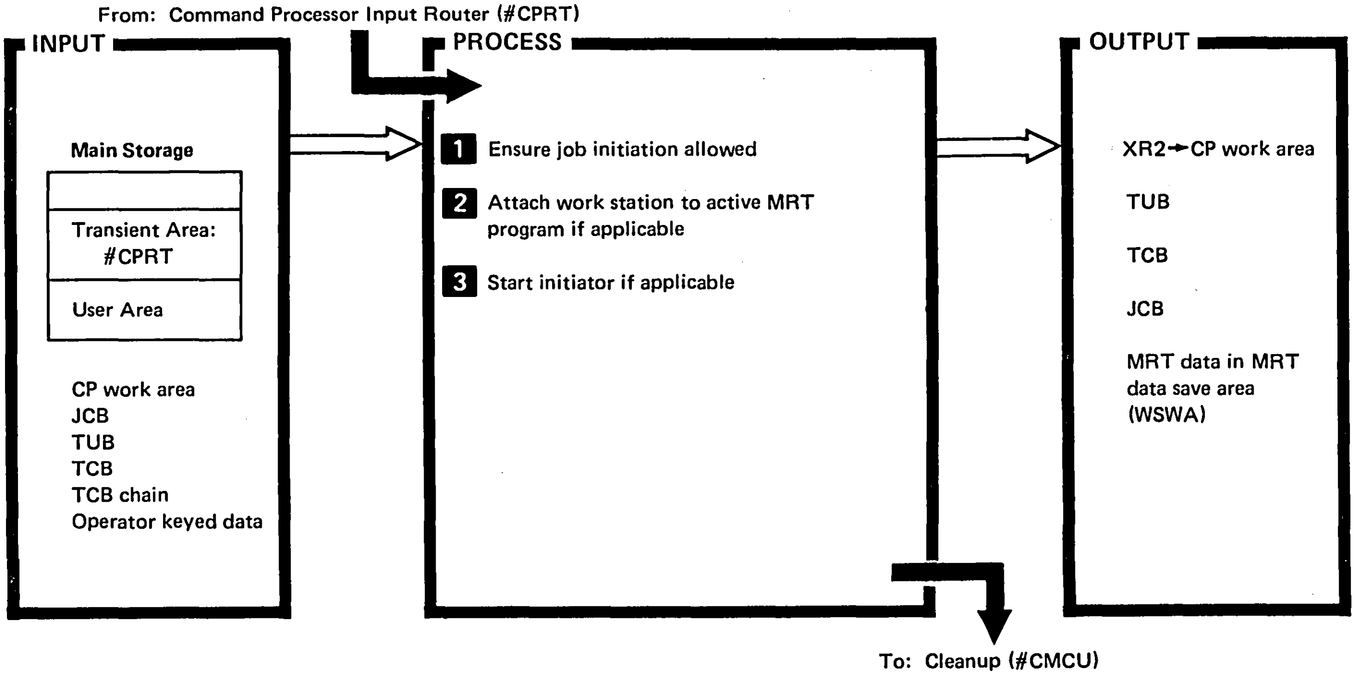

From: Input Router (#CPRT)
INPUT
Required system areas
PROCESS
● Perform job start function (Diagram 2.19)
● Perform release function (Diagram 2.20)
OUTPUT
Initiator start
Appropriate display
To: Initiator (#CIML)

Diagram 2.18. Overview of Job Initiation and Work Station Release

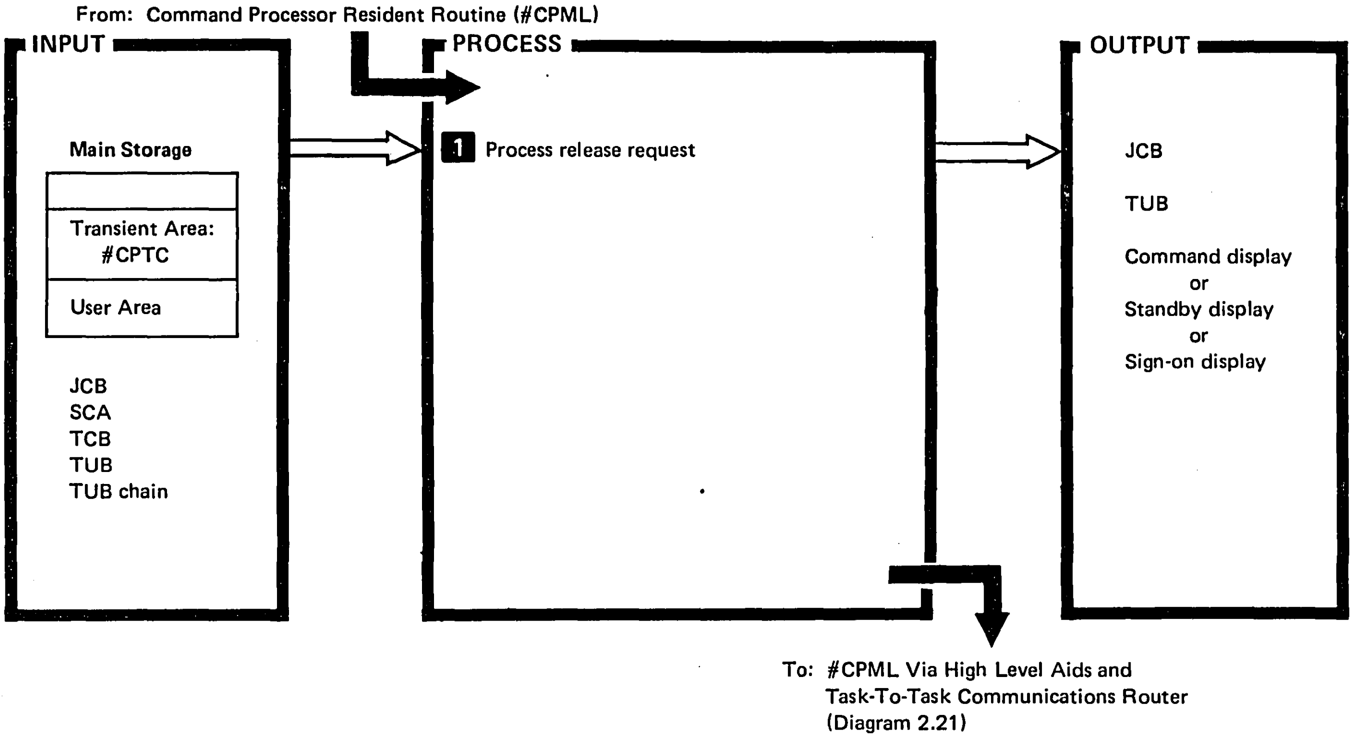

From: Command Processor Input Router (#CPRT)
INPUT
Main Storage
Transient Area: #CPRT
User Area
CP work area
JCB
TUB
TCB
TCB chain
Operator keyed data
PROCESS
1 Ensure job initiation allowed
2 Attach work station to active MRT program if applicable
3 Start initiator if applicable
OUTPUT
XR2→CP work area
TUB
TCB
JCB
MRT data in MRT data save area (WSWA)
To: Cleanup (#CMCU)

DESCRIPTION	MODULE/ ROUTINE
1 Call cleanup routine (#CMCU) to issue message: ● If initiation stopped (TUBSTPJ), issue job initiation stopped by system operator error message (MIC 5534). ● If terminal in console mode (TUBMCNSL), or if terminal is data terminal (TUBCMDT), issue command not allowed error message (MIC 5539).	#CPRT #CMCU
2 Check procedure to see if it is for active multiple requester terminal (MRT) program. If not for MRT, go to **3**. Ensure request valid: ● Inquiry requester not attempting to attach to a MRT waiting for resources. ● Inquiry requester not attaching to same MRT. Calculate data input length and place in MRT. Write input data to MRT data save area (WSWA). Update MRT task control block (TCB) fields for active requester count, allocated work station count and task invite count. If MRT task can not handle another requester (TCBMRTMX), set TUBECM skip flag (TUB$SKIP) off. Set MRT TCB address in requester TUB to designate ownership. Build action control element (ACE). Activate data mode in TUB.	#CPRT

Diagram 2.19 (Part 1 of 2). Perform Job Start Function

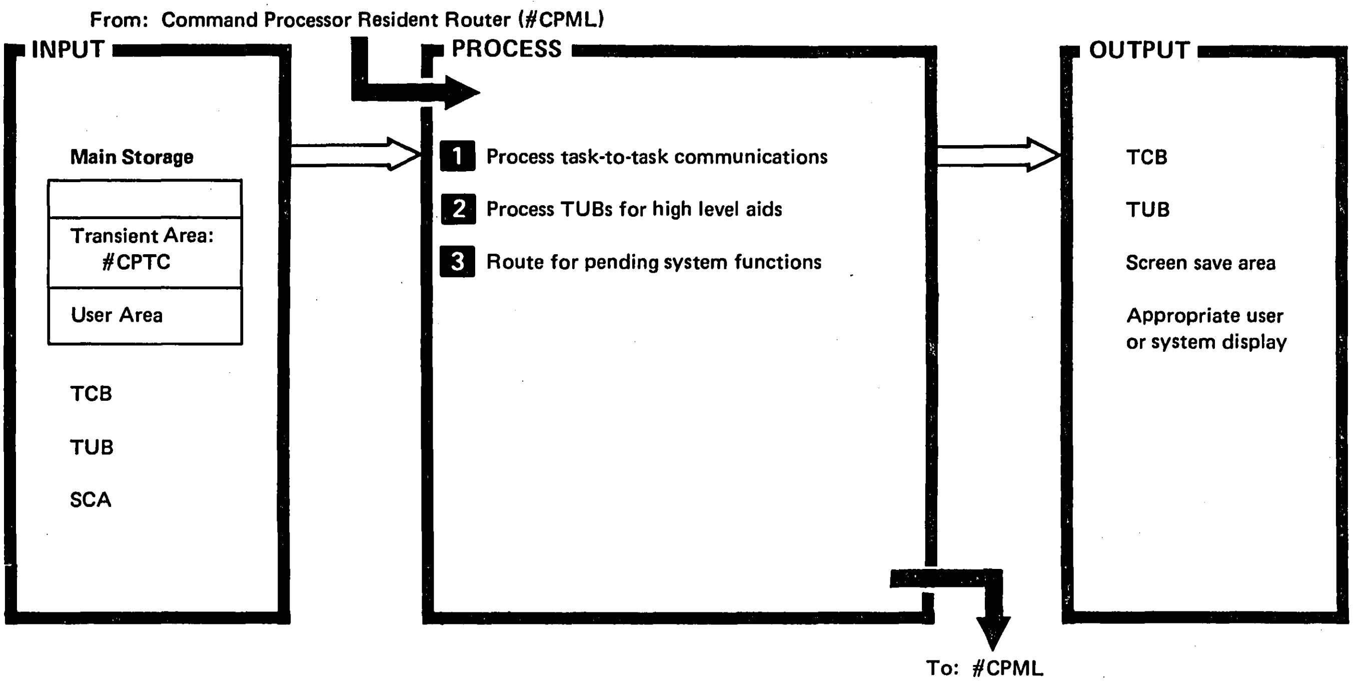
DESCRIPTION	MODULE/ ROUTINE
Post MRT task to accept input from new requester.	#CPRT
Pass control to cleanup routine (#CMCU) to log input.	
3 Build initiator attach parameter list.	
Activate data mode in TUB.	
Pass control to supervisor task attach transient (#SVAT) to start initiator (#CIML).	#SVAT
If attach fails, display error message.	

Diagram 2.19 (Part 2 of 2). Perform Job Start Function


From: Command Processor Resident Routine (#CPML)
INPUT
Main Storage
Transient Area: #CPTC
User Area
JCB
SCA
TCB
TUB
TUB chain
PROCESS
1 Process release request
OUTPUT
JCB
TUB
Command display
or
Standby display
or
Sign-on display
To: #CPML Via High Level Aids and Task-To-Task Communications Router (Diagram 2.21)

DESCRIPTION	MODULE/ ROUTINE
1 If termination cleanup required prior to release, call initiator with termination hook. If test request, call #CCOF.	#CPTC
If no more job steps:	#CPIO
● Initialize TUB fields.	#CPTC
● If restore not required invite display station for Enter key.	#CPIO
● If any messages to be displayed, call #CCMX. ● Display command interface. ● Post tasks which may want this display station. ● Return. If more job steps: ● If end of outermost procedure, setup hook for initiator to call termination (TUBABTRM). ● Start initiator.	#CPTC
● If initiator attach fails, set timer interrupt to retry in 2 seconds. ● Return.	#CPTC

Diagram 2.20. Perform RELEASE Function



From: Command Processor Resident Router (#CPML)

INPUT

Main Storage

Transient Area: #CPTC

User Area

TCB

TUB

SCA

PROCESS

1 Process task-to-task communications

2 Process TUBs for high level aids

3 Route for pending system functions

OUTPUT

TCB

TUB

Screen save area

Appropriate user or system display

To: #CPML

DESCRIPTION	MODULE/ ROUTINE
1 If I/O error, process it.	#CPTC #SVERP
If display station release, process it. (See *Perform Release Function*, Diagram 2.20.)	#CPTC
2 If system request, process request. (See *Process System Request Event*, Diagram 2.30.) If Attn key: ● If work station is: – Not signed on. – In test request mode. – Not in data mode. – Inquiry disabled. – In command reject mode. – Already in data mode escape. – Running a program which is in termination, then ignore Attn key. ● If the TCB address of the TUB is zero, pend the inquiry and retry in 2 seconds. (Use control storage routine, NUASGN.) ● If the vertical hold or save/restore interlock is on, pend the inquiry and retry in 2 seconds (NUASGN). ● If the interrupted task is a SRT: – If 1 option is pending, and if SQB count = 0, and if posted because of Attn key, then erase 1 option pending and process inquiry. – If SQB count = 4, post tasks waiting on disk enqueue. – If task owns interlocks, retry Attn key in 2 seconds (NUASGN); otherwise, suspend task. – Build an inquiry TUB and work station work area. – If no resources are available, retry inquiry and go to **1** Ⓐ.	

Diagram 2.21 (Part 1 of 2). Perform High Level Aids and Task-to-Task Communications Router Functions

DESCRIPTION	MODULE/ ROUTINE
● If the Interrupted task is a MRT: – If display station not invited, retry inquiry. – Indicate TUB to be skipped. – Build inquiry TUB. – If no resources available, retry inquiry. – If implicit invite to TUB or if PRUF (put for read under format) invite, decrease requester and work station count.	#CPTC
Ⓐ Swap the new TUB and the inquiry TUB: ● Find the previous TUB on the TUB chain. ● Lock the TUB chain. ● Rechain the new TUB to the TUB chain. ● Unlock the TUB chain. ● Post other TUB users. ● If a JCB exists: – Free up JCB and FSBs chained to it. – Free up WSWA. – Free up TUB.	#CPIQ
Save the work station display.	#CPIO
If inquiry option 1 was pending,	#CPTC
▶process 1 option. Otherwise,	#CPIQ
put up inquiry display.	#CPIO
Return to caller.	#CPTC
If Help key, process.	#WDDH
3 If console SYSLOG is pending,	#CPTC
process it.	#CMCI
If stop pending for any task, and if interlocks are now freed, suspend the task.	#CPTC
If stop system is pending, and stop is complete,	#CCRT
process the completed stop.	#CPTC
If JOBQ detach is pending, process next JOBQ task.	#CCJS

Diagram 2.21 (Part 2 of 2). Perform High Level Aids and Task-to-Task Communications Router Functions

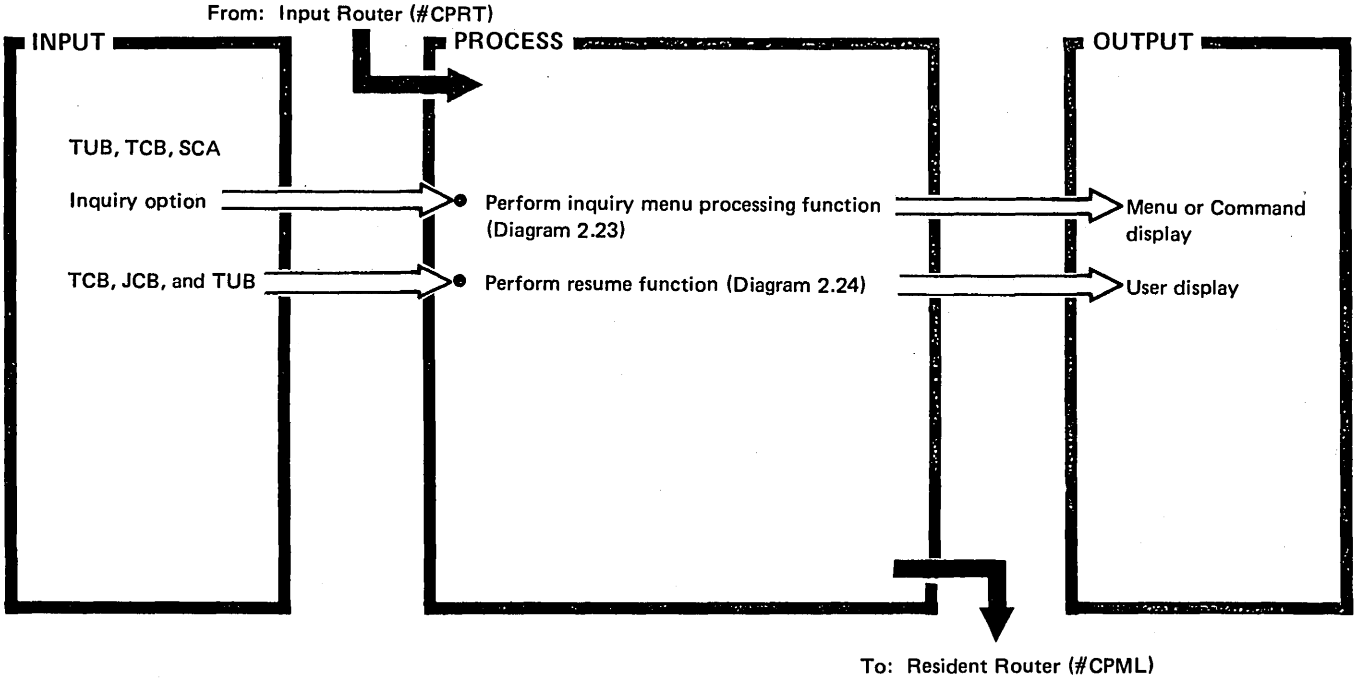
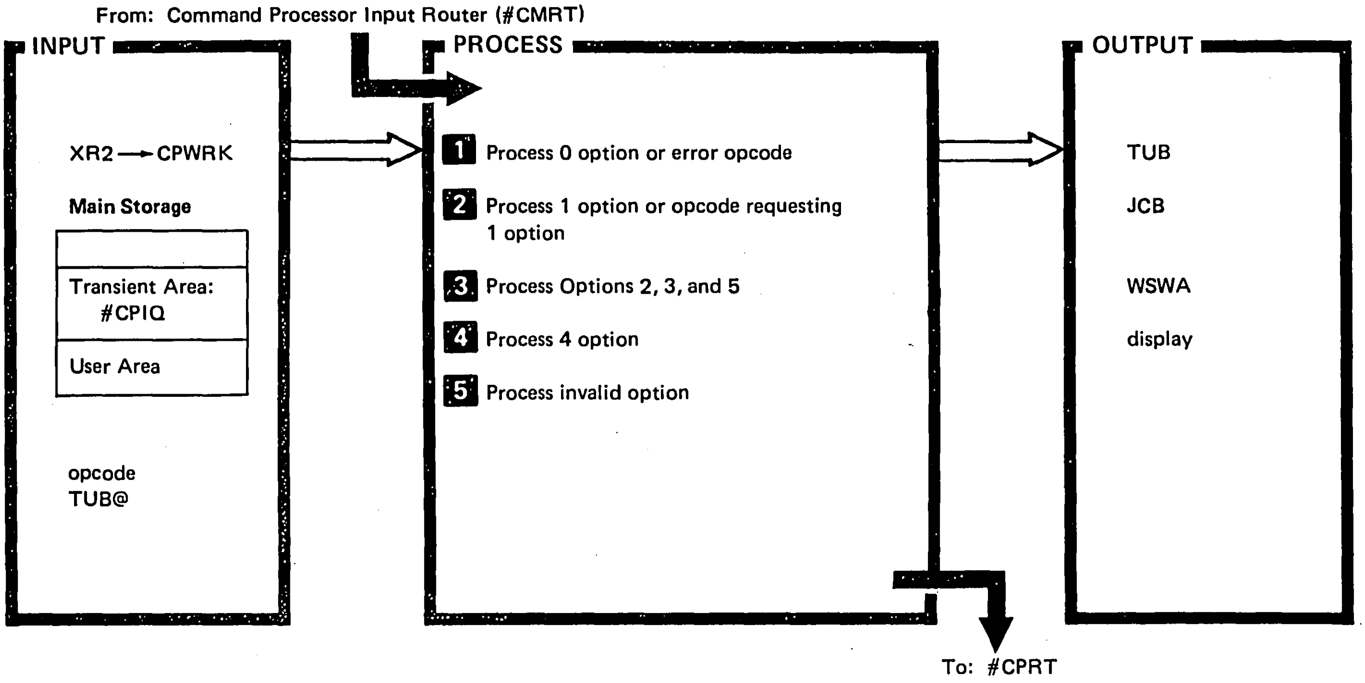

From: Input Router (#CPRT)
INPUT
PROCESS
OUTPUT
TUB, TCB, SCA
Inquiry option
TCB, JCB, and TUB
Perform inquiry menu processing function (Diagram 2.23)
Perform resume function (Diagram 2.24)
Menu or Command display
User display
To: Resident Router (#CPML)

Diagram 2.22. Overview of Inquiry Menu Option Processor and Resume



From: Command Processor Input Router (#CMRT)

INPUT

XR2 ──► CPWRK

Main Storage

Transient Area: #CPIQ
User Area

opcode
TUB@

PROCESS

1 Process 0 option or error opcode

2 Process 1 option or opcode requesting 1 option

3 Process Options 2, 3, and 5

4 Process 4 option

5 Process invalid option

OUTPUT

TUB

JCB

WSWA

display

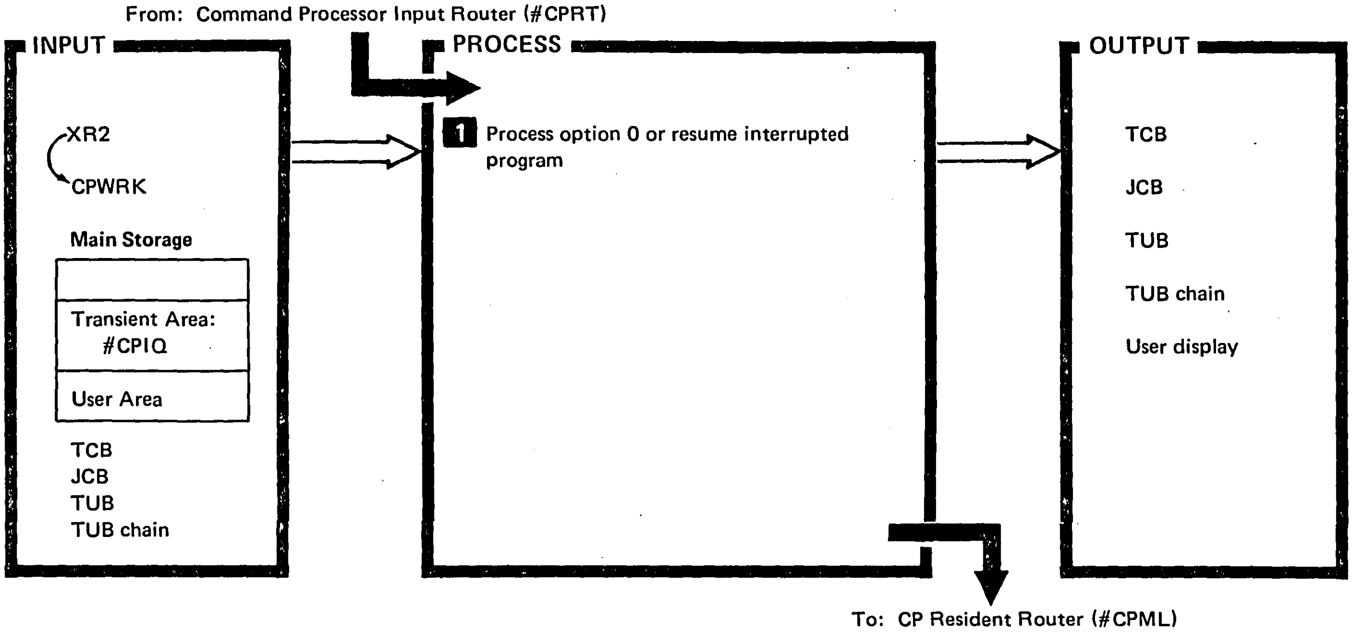
To: #CPRT

DESCRIPTION	MODULE/ ROUTINE
1 Dequeue inquiry TUB and free. Resume interrupted program. (See *Resume Function,* Diagram 2.25.)	#CPIQ
2 If second inquiry, issue diagnostic.	#CMCU
If task does not allow inquiry, issue diagnostic. If task is not a MRT, and if the sector queue count is not 0, ● Pend the inquiry. ● Resume the task. Otherwise, ● If the JCB and WSWA are already assigned, – Go to 2 A. ● Otherwise, – Assign JCB and allocate WSWA.	#CPIQ
– If no space, issue a diagnostic.	#CMCU
● If menu active and menu from #LIBRARY, increment format 1 use count.	#CPIQ
● Initialize JCB and WSWA.	#FDIOS
● Call RFINDLIB to	#CPIQ
find each library in the FSBs.	$MAFLB
A Put up the command display.	#CPIO
3 If TUB and TCB = 0, the TCB is a MRT; go to 3 A. Indicate to restore command display at termination, and disable inquiry.	#CPIQ

Diagram 2.23 (Part 1 of 2). Perform Inquiry Menu Function

DESCRIPTION	MODULE/ ROUTINE
If job is cancellable and not in termination, swap TUBs to remove inquiry: ● Find the previous TUB on the TUB chain. ● Lock the TUB chain. ● Rechain the new TUB to the TUB chain. ● Unlock the TUB chain. ● Post other TUB users. ● If a JCB exists: – Free up JCB and FSBs chained to it. – Free up WSWA. – Free up TUB. ● Set up parameter list and pass control to #CCCM. If job is in termination and a 2 option was taken, pend the cancel. If job is not cancellable: ● Indicate 2 or 3 cancel in TCB. ● Process RESUME option (Diagram 2.25). A Swap TUBs to free inquiry TUB: ● Find the previous TUB on the TUB chain. ● Lock the TUB chain. ● Rechain the new TUB to the TUB chain. ● Unlock the TUB chain. ● Post other TUB users. ● If a JCB exists: – Free up JCB and FSBs chained to it. – Free up WSWA. – Free up TUB. Indicate user display is not active. If a 2 option was taken: ● If task is not in termination and if the MIC in the JCB is zero, move MIC 3721 to JCB. If a 3 option was taken, indicate abnormal release in TUB.	#CPIQ
If the program has released the display station, process the release.	#CPTC
Set the no skip flag in TUB. If an ACE exists on the complete queue, ● Requeue it, last-in-first-out, to the complete queue. ● Dispatch task. Otherwise, ● Post TUB complete. ● Put release aid in TUB. ● Return to caller. 4 If MRT or released work station, issue error. Otherwise, set inquiry latch in JCB. Process as option 0. (See Diagram 2.25.) 5 Build parameter list. Pass control to #CMCU to issue error message.	#CPIQ

Diagram 2.23 (Part 2 of 2). Perform Inquiry Menu Function



From: Command Processor Input Router (#CPRT)

INPUT

XR2

CPWRK

Main Storage

Transient Area: #CPIQ

User Area

TCB
JCB
TUB
TUB chain

PROCESS

1 Process option 0 or resume interrupted program

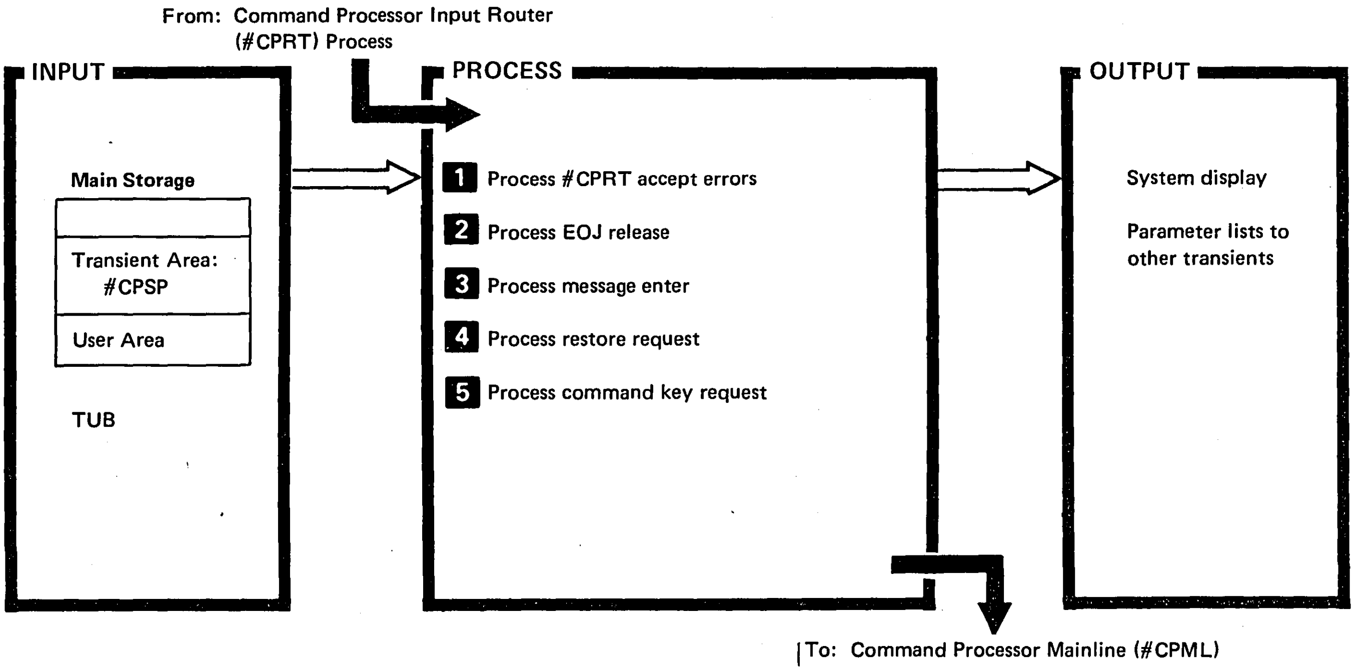
OUTPUT

TCB
JCB
TUB
TUB chain
User display

To: CP Resident Router (#CPML)

DESCRIPTION	MODULE/ ROUTINE
1 If status active, call #CCSM.	#CPIQ
End status.	#CCSM
If menu active and menu is from #LIBRARY, decrement user count on disk.	#CPIQ
If not error resume:	#CPIO
● Restore display. ● Swap TUBs to free inquiry TUB: – Find the previous TUB on the TUB chain. – Lock the TUB chain. – Rechain the new TUB to the TUB chain. – Unlock the TUB chain. – Post other TUB users. – If a JCB exists: a. Free up JCB and FSBs chained to it. b. Free up WSWA. c. Free up TUB. If display station has been released, call #CPTC to release. If MRT: ● If implicit invite to TUB, or invite due to put for read under format: – Add 1 to MRT REQ count. – Add 1 to work station count. ● If requester count less than MRTMAX, set on no-skip bit. ● Have task dispatched. ● Return to caller. If SRT: ● If printer readjustment for forms or image is required, assign an RB that causes the resumed task to call #CSIM. ● Set off task suspend bit in TCB. ● If suspended due to stop command or error suspend, bypass post; otherwise, post the task out of suspend.	#CPIQ

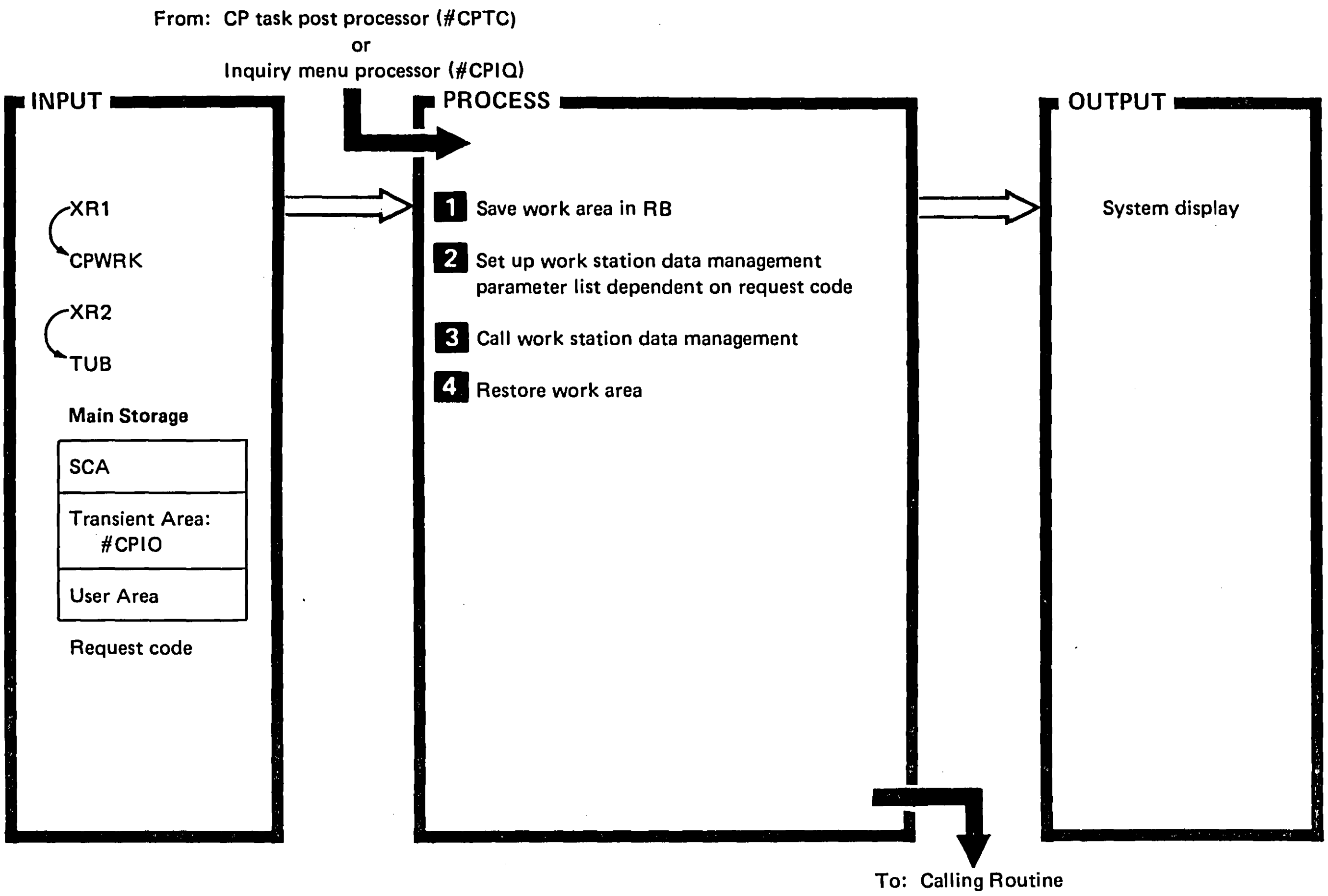
Diagram 2.24. Perform Resume Function


From: Command Processor Input Router (#CPRT) Process

|To: Command Processor Mainline (#CPML)

DESCRIPTION	MODULE/ ROUTINE
1 If #CPRT accept error: ● If sys request or inquiry in high level aid: – Set sys request and inquiry to zero. – Indicate that TUB is invited.	#CPSP
– Retry accept.	#CPRT
2 If restore yes indicator on in TUB:	#CPSP
● If messages pending, display them. Otherwise:	#CCMX
put up command display.	#CPIO or #CPIQ
3 If messages active, process them.	#CCMX
4 If restore indicated in TUB: ● If console restore, restore console.	#CPSP
Handle messages:	#CMCI
● If work station:	#CPIQ
– If status active, update status display. Otherwise:	#CCSM
put up command display and post tasks requesting display station.	#CPIO
Return to caller.	#CPSP
5 If command key 1 and if resume pending:	#CPSP
● Handle 0 option.	#CPIQ
If resume not pending:	#CPSP
● Indicate key not valid.	#WDDH
If test request key:	#CPSP
● If signed on, indicate key not valid. Otherwise,	#WDDH
attach test request.	#CPON

Diagram 2.25. Perform Special Command Processing Function


From: CP task post processor (#CPTC)
or
Inquiry menu processor (#CPIQ)
INPUT
XR1
CPWRK
XR2
TUB
Main Storage
SCA
Transient Area:
#CPIO
User Area
Request code
PROCESS
1 Save work area in RB
2 Set up work station data management parameter list dependent on request code
3 Call work station data management
4 Restore work area
OUTPUT
System display
To: Calling Routine

DESCRIPTION	MODULE/ ROUTINE
1 Save 30 bytes of work area (starting at IOCODE) and use this area for work space.	#CPIO
2 If clear request, indicate clear opcode.	
If invite request, indicate invite opcode.	
If stop invite request, indicate stop invite opcode.	
If save request, indicate save opcode.	
If restore request, indicate restore opcode.	
If none of the above, indicate put then invite request.	
If not invite, turn off invite opcode.	
Move work station ID to output area.	
If console mode, set up console index.	
If standby mode, set up standby index.	

Diagram 2.26 (Part 1 of 2). Perform Command Processor/Work Station Data Management Interface Function

DESCRIPTION	MODULE/ ROUTINE
If inquiry mode: ● Create jobname and work station ID. ● If MRT, set up MRT inquiry index and go to **3**. Otherwise, point to SRT inquiry index. If IQB count is not zero, indicate delayed option 1. If noncancellable, indicate delayed option 2, 3, or 5. If in termination, indicate a delayed option 2. **3** Pass control to #WDSM to display data. If I/O error, indicate hex FF in return I/O code.	# CPIO
If restore command reject, issue clear screen,	#DWDM
and try restore again. **4** Restore CPWRK from RB.	#CPIO

Diagram 2.26 (Part 2 of 2). Perform Command Processor/Work Station Data Management Interface Function

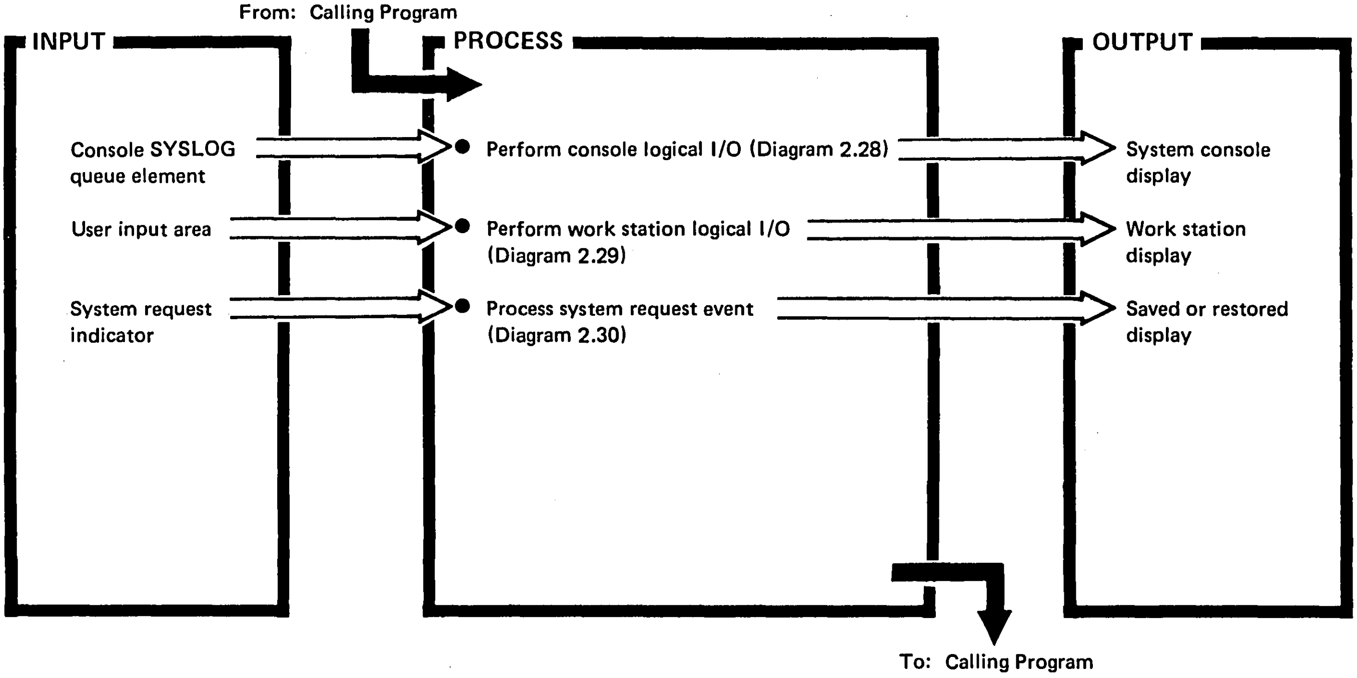
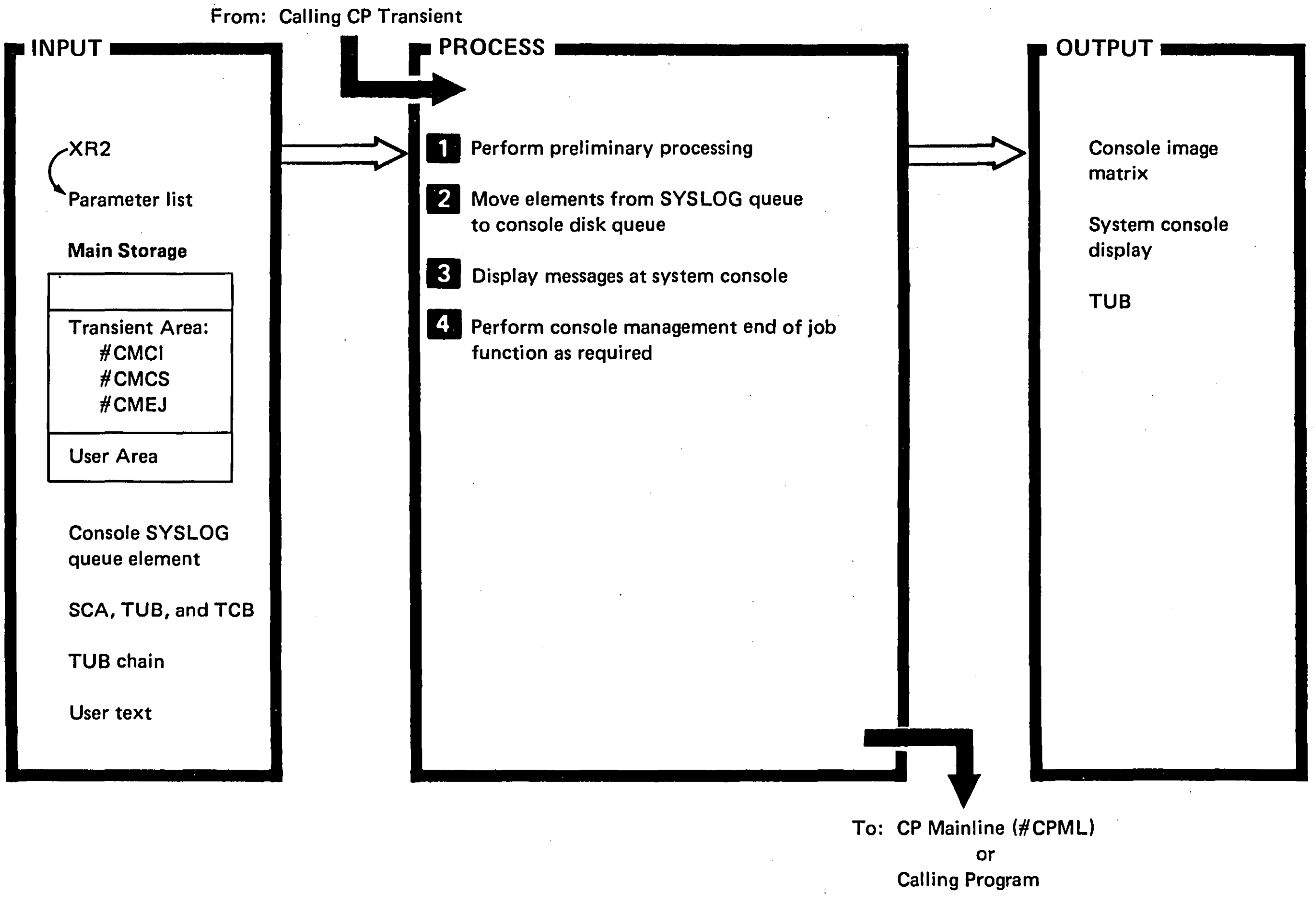

From: Calling Program
INPUT
PROCESS
OUTPUT
Console SYSLOG queue element
User input area
System request indicator
Perform console logical I/O (Diagram 2.28)
Perform work station logical I/O (Diagram 2.29)
Process system request event (Diagram 2.30)
System console display
Work station display
Saved or restored display
To: Calling Program

Diagram 2.27. Overview of Console Management



From: Calling CP Transient

INPUT

XR2

Parameter list

Main Storage

Transient Area:
#CMCI
#CMCS
#CMEJ

User Area

Console SYSLOG queue element

SCA, TUB, and TCB

TUB chain

User text

PROCESS

1 Perform preliminary processing

2 Move elements from SYSLOG queue to console disk queue

3 Display messages at system console

4 Perform console management end of job function as required

OUTPUT

Console image matrix

System console display

TUB

To: CP Mainline (#CPML)
or
Calling Program

DESCRIPTION	MODULE/ ROUTINE
1 Check system communication area (SCA) error recovery procedure (ERP) indicator (SCAMERP) and if on, call control storage error handler (#SVER).	#CMCI #SVERP
Assign task work area space.	
Find terminal unit block (TUB) for system console.	
Enqueue the TUB on the console queue.	#CMCI
▶ If there are no more messages to be displayed, turn off light.	WSIOCH ◀
2 Determine if any elements on SYSLOG queue waiting to be moved from user SYSLOG sector to console SYSLOG disk queue.	#CMCI
Find space on SYSLOG disk queue if possible.	
Move element from user SYSLOG sector to console SYSLOG disk queue.	Disk IOS
Free up SYSLOG queue sector for reuse.	#CMCI

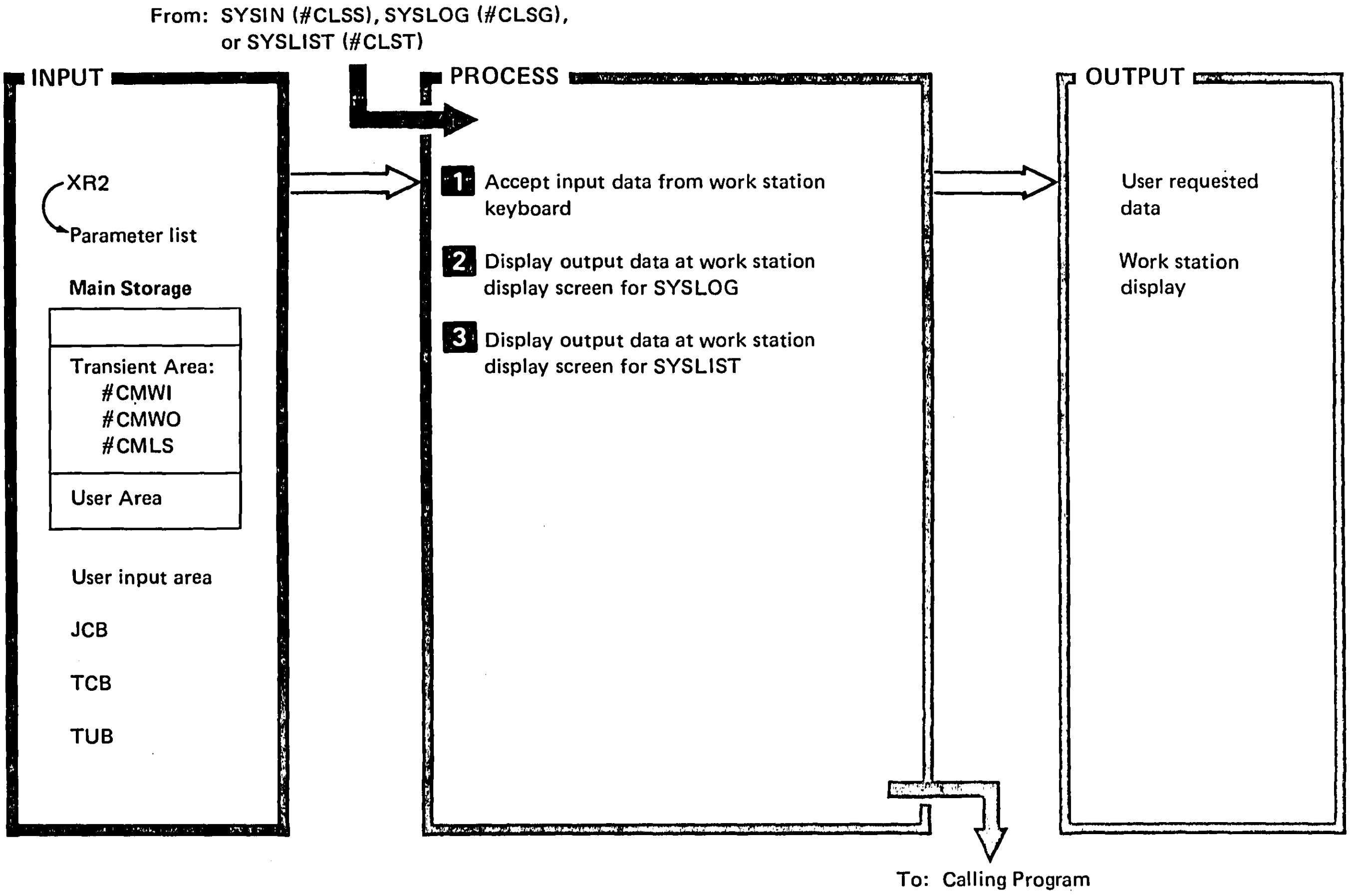
Diagram 2.28 (Part 1 of 3). Perform Console Logical I/O Function

DESCRIPTION	MODULE/ ROUTINE
If queue full: ● Find unprocessed queue element with no wait indicator (CMCINOW) on. ● Dequeue element with no wait indication. ● Free queue sector for resue. ● Post calling program. If IDELETE active at system console: ● Search for write to operator without reply (WTO) element in queue. ● If WTO element found and space needed: – Deallocate SYSLOG queue sector. – Dequeue and free disk sector (FIFO). If no WTOs to free and queue full, caller must wait for space. 3 Find first queue element address. Ensure enough lines available or can be rolled off screen to display new data. If not enough screen lines or if no more elements on queue, and if console in console mode: ● Dequeue console queue resource. ● If EOJ response indication (CMCIEJR) on in console matrix, go to 4; otherwise, exit to command processor mainline (#CPML). If this is a new console SYSLOG post: ● Build work station data management parameter list.	# CMCI
● Sound audible alarm at console.	WSIOCH
If console not in console mode, or if screen not available, exit to #CPML. Load and pass control to console management second half (#CMCS).	#CMCI
Build work station data management parameter list to consolidate lines at bottom of screen that may be overlaid.	#CMCS
Roll screen.	#DWDM
If message command queue element (CMCIMSG): ● Find proper message queue sector. ● Read in message sector. ● Update message sector to indicator sector available for reuse. If user text sector: ● Assign message number. ● Find user sector text. ● Read in text sector. Generate message ID for operator reply. Increment message ID control counter byte (SCADMID#). Build printer IOB if spool not active, system in single program mode, and log printer active.	#CMCS
Call work station input output control handler (WSIOCH) to print output data.	#CMCU
Set up console matrix entries associated with line on screen. Save line to be displayed on screen in command processor task work area (TWA) line save area. Build work station data management parameter list.	#CMCS

Diagram 2.28 (Part 2 of 3). Perform Console Logical I/O Function

DESCRIPTION	MODULE/ ROUTINE
Put display to console display screen.	#DWDM
Dequeue and free queue element just processed. Continue processing elements until queue is empty, screen is full, or element not yet placed in disk queue.	#CMCS
If queue empty, turn off message light.	WSIOCH
Exit: ● Free up any assign/free area still owned. ● If EOJ response indication (CMCIEJR) is on in console matrix, go to **4**; otherwise dequeue on console SYSLOG queue, and exit to command processor mainline (CPML).	#CMCS
4 Find command processor task work area (TWA). Point at in-core matrix. Loop through matrix setting off EOJ bits (CMCIEJR). Determine message ID fields to be overlayed by '**'. Build work station data management parameter list.	#CMEJ
Put '**' over message ID on console display screen.	#DWDM
Exit to calling program.	#CMEJ

Diagram 2.28 (Part 3 of 3). Perform Console Logical I/O Function


From: SYSIN (#CLSS), SYSLOG (#CLSG),
or SYSLIST (#CLST)
INPUT
XR2
Parameter list
Main Storage
Transient Area:
#CMWI
#CMWO
#CMLS
User Area
User input area
JCB
TCB
TUB
PROCESS
1 Accept input data from work station keyboard
2 Display output data at work station display screen for SYSLOG
3 Display output data at work station display screen for SYSLIST
OUTPUT
User requested data
Work station display
To: Calling Program

DESCRIPTION	MODULE/ ROUTINE
1 Obtain task work area (TWA) space.	#CMWI
Find requesting work station's terminal unit block (TUB).	
If user invite outstanding, wait for invite end.	
If user display on screen:	
● Build work station data management parameter list to save user display .	
▶ ● Save current format on screen.	#DWDM ◁
Reset invite and SYSLIST screen active indication.	#CMWI
If initial call for logical I/O interface:	
● Retrieve user's initial record entered on command display (OCL or procedure statement).	Disk IOS
● Return.	#CMWI
● Output user's initial record to screen.	#DWDM
If not initial call for logical I/O interface:	
● Build work station data management parameter list to output logical I/O display.	#CMWI
● Display logical I/O interface and wait for user's response.	#DWDM

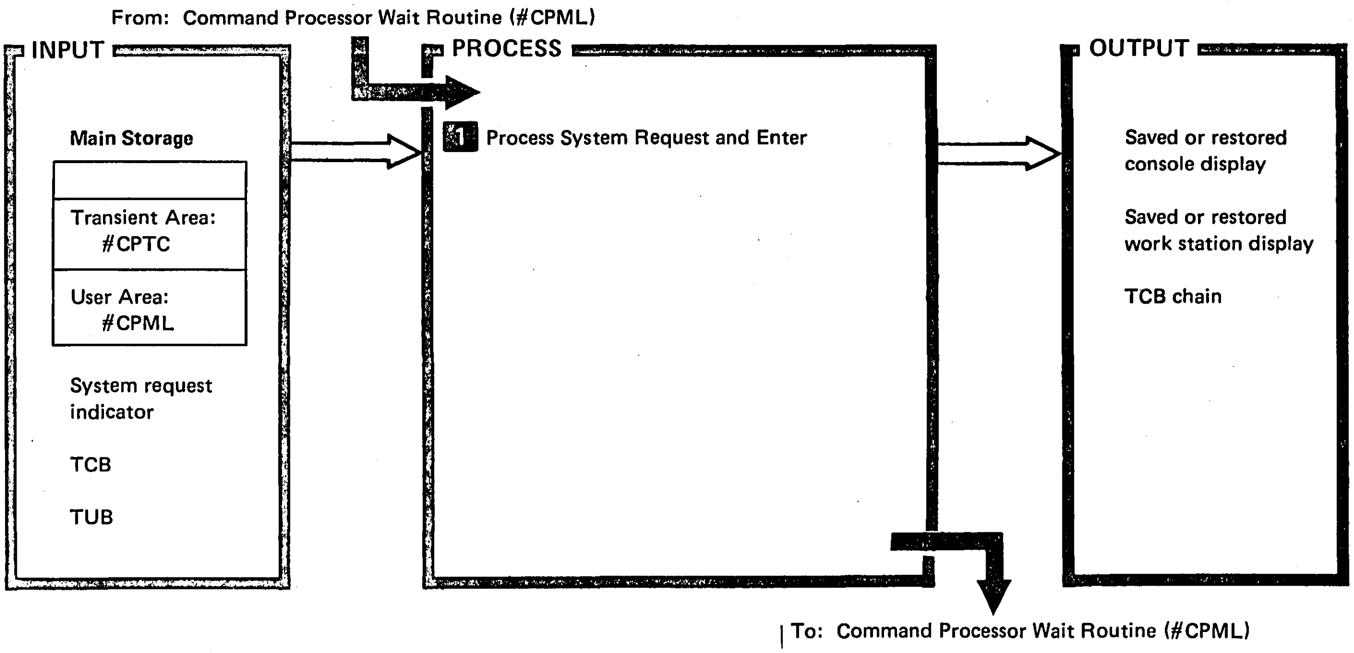
Diagram 2.29 (Part 1 of 3). Perform Work Station Logical I/O Function

DESCRIPTION	MODULE/ ROUTINE
Process user entered response data:	#CMWI
● If null data response, blank out input data area.	
● Move user response bytes into work station logical I/O data management input record area and then to user's area.	
Roll user response lines off screen input area:	
● Build work station data management parameter list with information to clear screen input area.	
● Roll user entered data out of input area.	#DWDM
Return control to SYSIN (#CLSS).	#CMWI
2 If entry is from SYSLIST, go to 3.	#CMWO
If write to operator without reply request, IDELETE active, and logical I/O display interface not active:	
● Throw away informational message.	
● Return to SYSLOG (#CLSG).	
Wait for user I/O activity to finish.	#CMWO
If user display active on screen (TUBUSUP):	
● Save work station invite status.	
● Build work station data management parameter list to save user's display.	
● Save user's display.	#DWDM
● Indicate user screen saved (TUBATTR6).	#CMWO
If logical I/O interface not active on screen (TUBINPA):	
● If logical I/O interface previously displayed (TUBLIOS):	
– Build work station data management parameter list to restore logical I/O display.	
– Restore logical I/O screen.	#DWDM
● If logical I/O interface never active for this sign-on (TUBLIOS):	#CMWO
– Retrieve user's initial OCL or procedure statement.	Disk IOS
– Build work station data management parameter list for initial logical I/O screen.	#CMWO
– Display screen with first keyed record.	#DWDM
If second level message to display (CMW02ND):	#CMWO
● Build work station data management parameter list.	
● Display second level message.	#DWDM
● Blank out response input area.	#CMWO
● Read user's option response.	#DWDM
● Ensure response length valid.	#CMWO
● If response length error:	
– Retrieve error message text.	#MGRET
– Build work station data management parameter list.	#CMWO
– Display error message.	#DWDM
– Read user's retry response.	
● Move good response to user area.	#CMWO
● Verify user response valid:	
– If write to operator without reply (WTO), indicate no screen restore.	
– If other valid response, indicate to restore screen.	
● If restore requested:	
– Build work station data management parameter list to restore user screen.	
– Restore screen.	#DWDM
– Return to calling program.	#CMWO
A If call by WTO:	
● Put out blank logical I/O screen.	
● Build work station data management parameter list to display WTO message.	#CMWO
● Display message.	#DWDM
● Return to SYSLOG (#CLSG).	#CMWO

Diagram 2.29 (Part 2 of 3). Perform Work Station Logical I/O Function

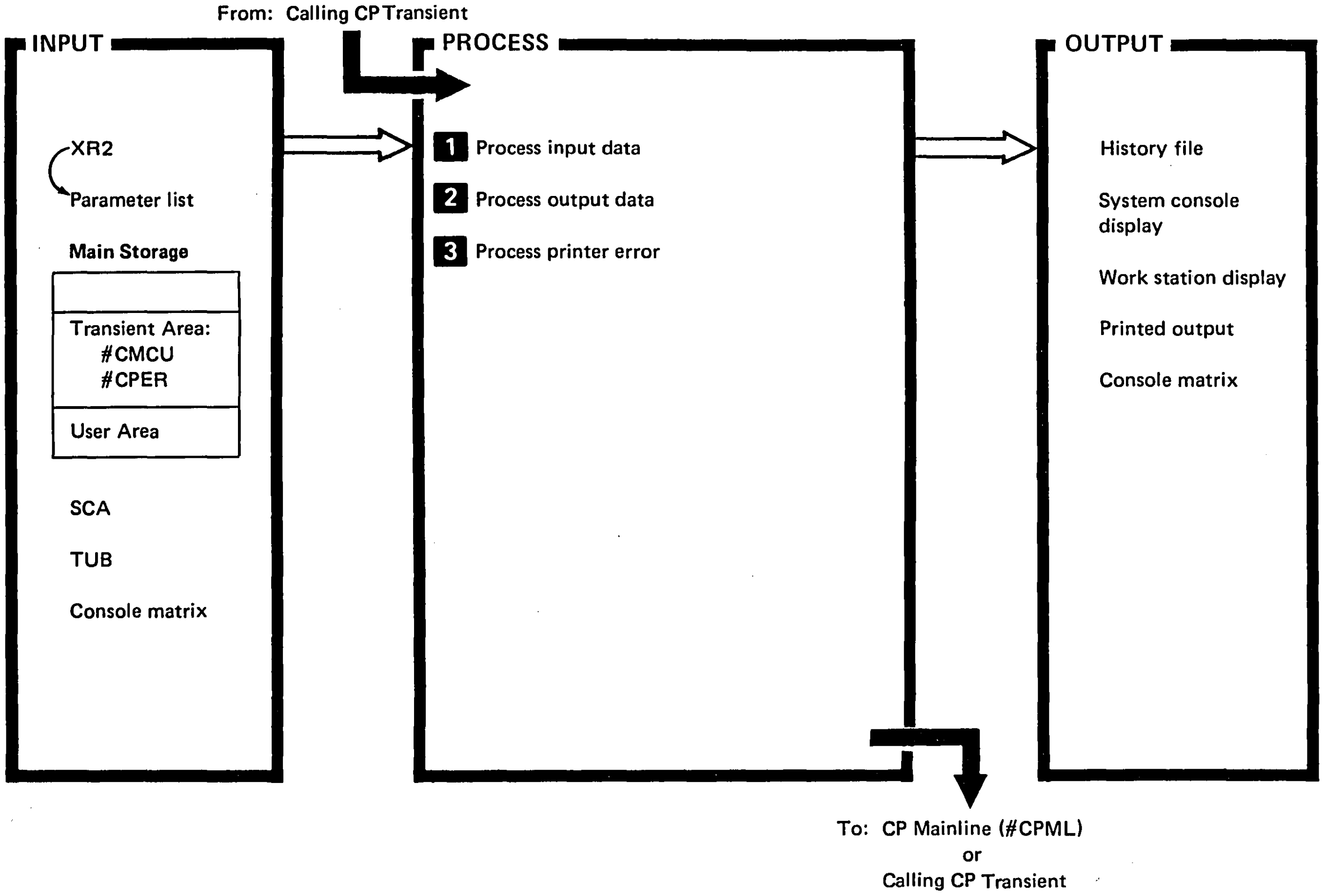
DESCRIPTION	MODULE/ ROUTINE
If call to display SYSLOG message, write to operator with reply (WTOR):	# CMWO
● If SYSLOG halt and not WTO/WTOR:	
– Build work station data management parameter list.	
– Save screen.	
– Sound audible alarm at work station.	
● Build work station data management parameter list to display message.	
● Display halt/WTOR message.	#DWDM
● If WTO request, return to caller.	#CMWO
● Read user's response.	#DWDM
● Ensure response length valid.	#CMWO
● If response length error:	
– Retrieve error message text.	#MGRET
– Display error messgge	#DWDM
– Read user's retry response.	
● Move good response to user area.	#CMWO
● Verify user response valid.	
● Roll up WTOR response data.	#DWDM
● If restore indicated:	#CMWO
– Build work station data management parameter list to restore screen saved.	
– Restore user's screen.	#DWDM
– Return to calling program.	#CMWO
3 If SYSLIST active on screen (TUBSYLST), go to 3 A.	#CMLS
Wait for user I/O activity to finish.	
If user display active on screen (TUBUSUP):	
● Save work station invite status.	
● Build work station data management parameter list to save user's display.	
● Save user's display	#DWDM
● Reset invite status (TUBATTR3).	#CMLS
● Indicate user screen saved (TUBATTR6).	
A If call by SYSLIST put:	
● If SYSLIST screen inactive (TUBSYLST):	
– Save logical I/O screen if active.	#DWDM
– Put out blank logical I/O screen.	
● Build work station data management parameter list to display SYSLIST message.	#CMLS
● Display message.	#DWDM
● Return to SYSLIST (#CLST)	#CMLS
If call by SYSLIST get:	
● Build work station data management parameter list to display message.	
● Display WTOR message.	#DWDM
● Read user's response.	
● Ensure response length valid.	#CMLS
● If response length error:	
– Retrieved error message text.	#MGRET
– Display error message.	#DWDM
– Read user's retry response.	
● Move good response to user area.	#CMLS
● Verify user response valid.	
● Blank user response area.	#DWDM
● Return to calling program.	#CMLS

Diagram 2.29 (Part 3 of 3). Perform Work Station Logical I/O Function


From: Command Processor Wait Routine (#CPML)
INPUT
Main Storage
Transient Area: #CPTC
User Area: #CPML
System request indicator
TCB
TUB
PROCESS
1 Process System Request and Enter
OUTPUT
Saved or restored console display
Saved or restored work station display
TCB chain
To: Command Processor Wait Routine (#CPML)

DESCRIPTION	MODULE/ ROUTINE
1 If IPL is in progress:	#CPTC
○ If master console or alternative console, and not already signed on, call #CPTS. Display sign on.	#CPTS
○ Return to caller.	#CPTC
If IPL not in progress:	
○ If not signed on, call #CPTS. Display sign on.	#CPTS
○ If console in error, call #CPER for error recovery.	
If work station in console mode:	
○ If error recovery busy, ignore system request.	
○ Issue stop invite to system console.	#CPIO
○ If stop invite failed, ignore system request.	#CPTC
○ Issue save for console display.	#CPIO
○ If work station display was not previously saved:	#CPTC
– Swap the TUBs.	#CPIO
– Display the proper system display.	#CPTC
– Return to caller.	
○ If console mode was forced, issue clear unit I/O operation;	#CPIO
otherwise, restore the saved display.	#CPTC
○ Swap the TUBs.	
○ If the work station was released, exit to #CPIQ.	
○ If a task was active, have it dispatched.	
○ If inquiry menu pending, process the inquiry.	
If work station to console:	
○ If vertical hold or save/restore interlock, pend the system request.	
○ If TUB not owned by command processor, set TUB to be skipped.	
○ If TUB owned by command processor, issue stop invite.	#CPIO
○ Swap TUBs.	#CPTC
○ Save work station display.	#CPIO
○ If restore is required, restore console display; otherwise, put up console display.	#CPTC
○ Indicate to call #CMCI.	
○ Return to caller.	

Diagram 2.30. Process System Request Event

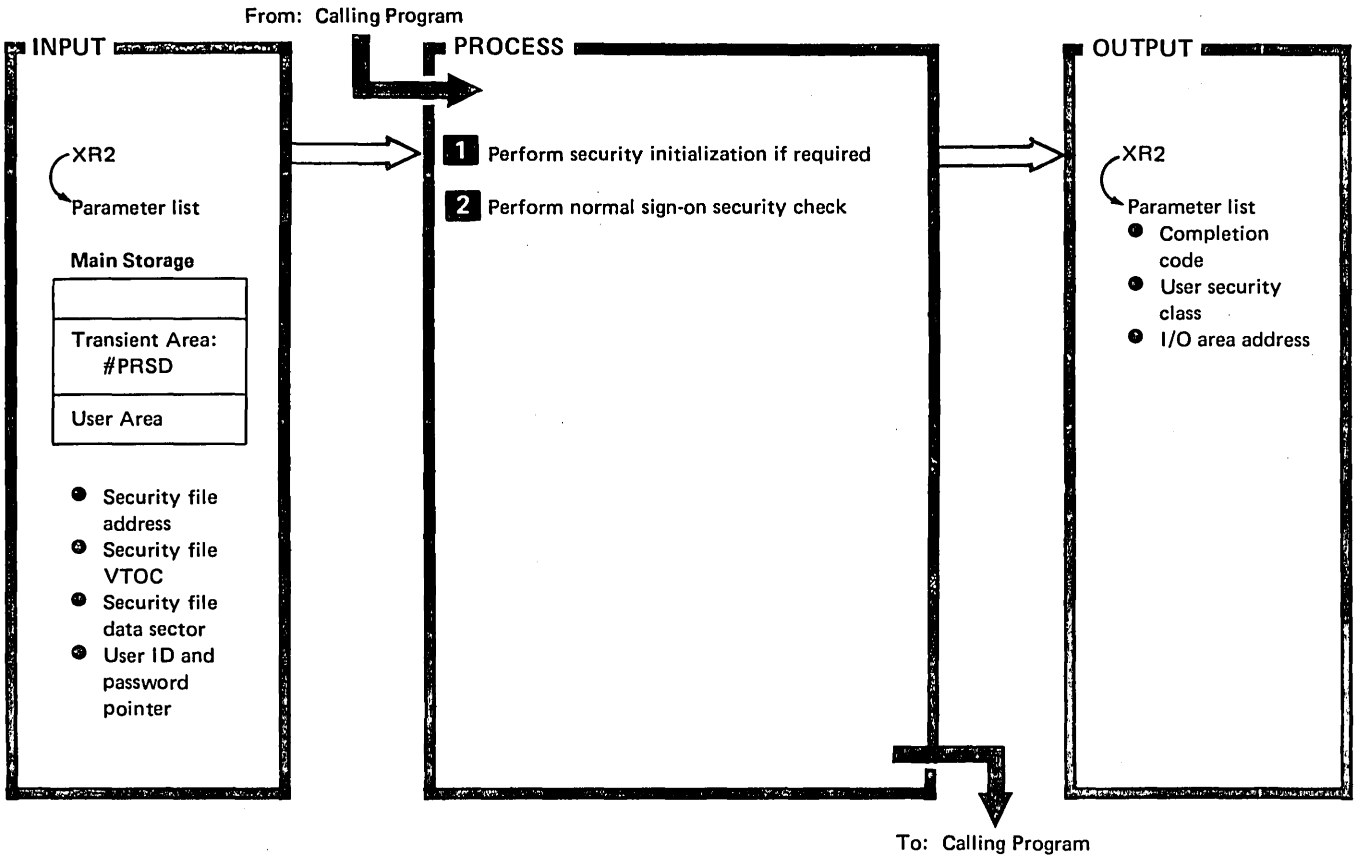


DESCRIPTION	MODULE/ ROUTINE
1 If no input data and no output data to process, return to calling program.	#CMCU
If no input data (CMCUINP = OFF), go to **2**.	
If log input requested (CMCULOG):	
● Build history file put (#HFPUT) parameter list.	
● Place input data in history file.	#HFPUT
● Build printer IOB if spool not active, system in single program mode, and log printer active.	#CMCU
● Call work station input output control handler (WSIOCH) to print input data.	WSIOCH
● If printer error, go to **3**.	#CMCU
If roll screen request (CMCUROL):	
● Find terminal unit block (TUB) for terminal requesting roll.	
● If screen format is menu standby, command, inquiry, sign-on, or status (TUBATTR1):	
– Build work station data management (WSDM) parameter list.	
– Reinvite screen.	#DWDM
– Return to calling command processor transient.	#CMCU
If system console and in console mode:	
● Build WSDM parameter list to consolidate lines at bottom of screen that will be overlaid.	
● Roll screen.	#DWDM

Diagram 2.31 (Part 1 of 2). Perform Command Processor Cleanup Function

DESCRIPTION	MODULE/ ROUTINE
● Build WSDM parameter list for system console.	#CMCU
● Modify system console display.	#DWDM
● Return to calling command processor transient.	#CMCU
2 If message identification code (MIC) number provided (CMCUMIC):	
● Build message retrieve parameter list.	
▶ ● Retrieve message.	#MGRET ◀
If execution time data to be substituted in message (CMCUMSUB):	#CMCU
● Scan message looking for # signs.	
● Substitute appropriate characters for # signs.	
If in-core message to be displayed (output switch on and MIC number field=zero), move message to command processor work area.	
If log output request (CMCULOG):	
● Build history file put (#HFPUT) parameter list.	
● Place input data in history file.	#HFPUT
Find output terminal TUB address.	#CMCU
If screen format is standby, command, Inquiry, menu, sign-on, or status (TUBATTR1 and TUBATTR2):	#CMCU
● Build WSDM parameter list.	
● Display messages.	#DWDM
Build printer IOB if spool not active, system in single program mode, and log printer active.	#CMCU
Call WSIOCH to print output data.	WSIOCH
If printer error, or if console only bits, go to **3**.	#CPER
If request to show to console also (CMCUSWS):	#CMCU
● Find space in disk queue for message if possible.	
● Place message in disk queue sector.	
● Build console queue element.	
● Place queue element on console SYSLOG queue.	
● Post console SYSLOG.	
If system console and in console mode (TUBATTR2):	
● Build WSDM parameter list to consolidate lines at bottom of screen that will be overlaid.	
● Roll screen.	#DWDM
● Build WSDM parameter list for system console.	#CMCU
● Put appropriate screen to system console.	#DWDM
● Build printer IOB if spool not active, system in single program mode, and log printer active.	#CMCU
● Call WSIOCH to print data if IOB built.	
● If printer error, go to **3**.	#CPER
Return to calling command processor transient module.	#CMCU
3 If printer error:	#CPER
● Find space in disk queue for message if possible.	
● Place message in disk queue sector.	
● Build console queue element.	
● Place queue element on console SYSLOG queue.	
● Post console SYSLOG.	
● Issue reset operation to the printer through WSIOCH.	
● Reset any error condition indicators in printer TUB.	
Return to calling program.	

Diagram 2.31 (Part 2 of 2). Perform Command Processor Cleanup Function

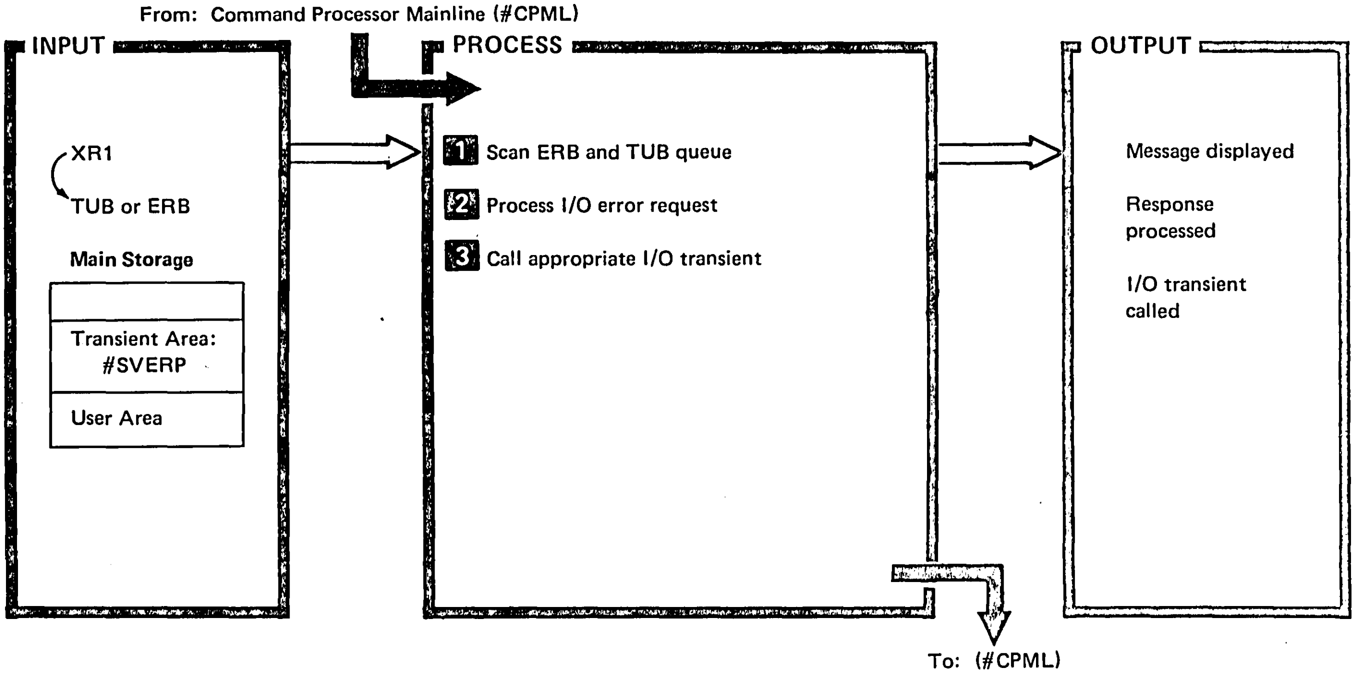


DESCRIPTION	MODULE/ ROUTINE
1 Check security initialization flag (SCAMSFLG); if not on, go to **2**. If user ID and password are special values, and if the user is at the system console: ● Permit sign-on. ● Set normal return code (PRSFNML). ● Make current user master security officer. ● Return to calling program. Build VTOC read parameter list.	#PRSD
▶ Read security file format 1 from VTOC.	@CSVF
If no security profile exists: ● Set security profile lost return code (PRSFLST). ● Return to calling program. Initialize security data in system communication area (SCA). **2** Initialize search buffer: ● Clear buffer to X'FF'. ● Put user ID in first 8 bytes of each 64-byte record in search buffer. Build disk IOB for security file disk scan operation.	#PRSD
Search security file for user ID.	Disk IOS

Diagram 2.32 (Part 1 of 2). Perform Password Security Function

DESCRIPTION	MODULE/ ROUTINE
If disk scan not successful or successful but desired record not in scan buffer: ● Set not found completion code (PRSFRNF). ● Return to calling program. If search successful-scan hit ($FDSCEQ) and record found in scan buffer: ● If incorrect password given: – Set bad password completion code (PRSFBPW). – Return to calling program. ● If correct password given: – Set normal completion code (PRSFNML). – Move security class to output area. – Return to calling program. If I/O error occurs on disk scan: ● Set I/O error return code (PRSFIOR). ● Return to calling program.	#PRSD

Diagram 2.32 (Part 2 of 2). Perform Password Security Function

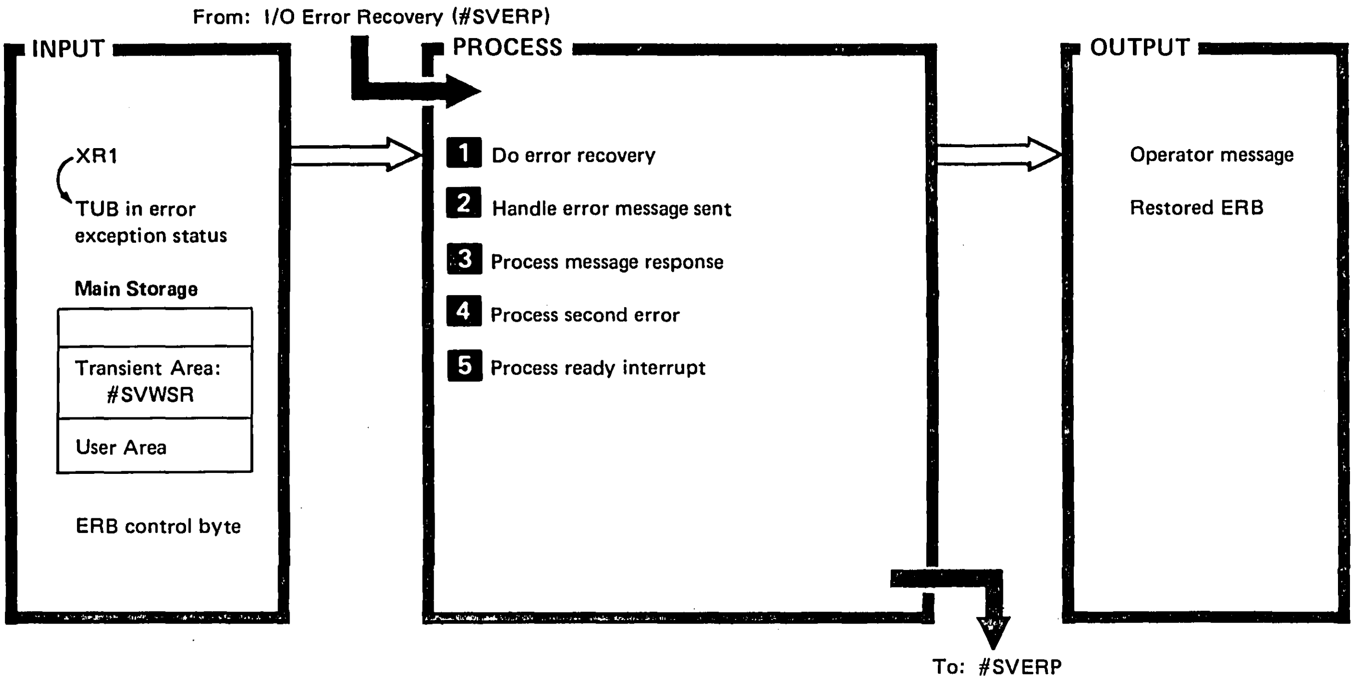


DESCRIPTION	MODULE/ ROUTINE
1 Scan the ERB and TUB chains to check for error conditions: ● ERB busy on. ● ERB done off. ● I/O error flag on. If not error conditions found, return to caller. **2** If request is for message at system console:	#SVERP
● Prepare space for message in console SYSLOG area on disk. ● Build message in the command processor work area: – Push current work area to disk.	#SVWER
– Get format line. – Get no message found message. – Get message for MIC#.	#MGRE
● Write message buffer to history file and to console SYSLOG sector on disk.	#HFPUT
● Place message on console SYSLOG queue. ● Restore command processor work area. ● Return to #SVERP.	#SVWER
● Go to **3**. If request is wait for response: ● If work station error is being processed, pass control to #SVWSR (Diagram 2.35). ● If 3 option taken, point task IAR at the EOJ SVC in system communications area. ● If D option taken, point task IAR at the Dump SVC in the system communications area. ● Go to **3**. If request is purge message: ● Search the console SYSLOG queue for any I/O messages that have been built but not displayed; also search display station matrix. ● Remove message, if found.	#SVERP

Diagram 2.33 (Part 1 of 2). I/O Error Recovery

DESCRIPTION	MODULE/ RESPONSE
3 If request is for I/O error recovery: ● Mark ERB as done. ● If work station error is being processed, pass control to #SVWSR (Diagram 2.35). ● If the ERB request was for printer error recovery:	#SVERP
– If the no response required flag (ERBMNRSP) in the printer TUB ERB is on,	#SVPRE
clear the error by issuing a clear command to the printer.	Ptr IOCH
– If the no response required flag is off, and if the printer TUB is for a display station printer:	#SVPRE
a. Set forms length. b. Set graphic error action. c. Set forms width.	Ptr IOCH
● If other printer error, or other I/O device error is being processed, call applicable control storage error recovery transient. ● Go to **1**.	#SVERP

Diagram 2.33 (Part 2 of 2). I/O Error Recovery



From: I/O Error Recovery (#SVERP)

INPUT

XR1

TUB in error exception status

Main Storage

Transient Area: #SVWSR

User Area

ERB control byte

PROCESS

1 Do error recovery

2 Handle error message sent

3 Process message response

4 Process second error

5 Process ready interrupt

OUTPUT

Operator message

Restored ERB

To: #SVERP

DESCRIPTION	MODULE/ ROUTINE
1 If called to do error recovery, determine class of error:	#SVWSR
● If resources temporarily unavailable:	
– If work station is not ready, exit to display station error recovery for device not ready (Diagram 2.36).	
– If error is read to unlocked keyboard by command processor:	
a. Free TP buffer.	
b. Post command processor.	
c. Clear ERB.	
d. Reset/invite keyboard.	
e. Return to caller.	
– If device is in error mode or powered off:	
a. Set hardware error.	
b. Go to 1 D.	
– If device is operational:	
a. Set programming error.	
b. Go to 1 A.	
● If programming error:	
– Free TP buffer.	
– If command processor caused error, reclassify error as hardware and go to 1 D.	
A – If user program caused error:	
a. If TUB in error is being released, pend error until release has completed.	
b. Suspend user task or pend suspension.	
c. Recycle error if not already done.	
d. If programming error on master console, force console mode.	# CPTC
e. Set 2 and 3 options allowed.	#SVWSR
B f. Set up ERB to send message.	
g. Return to caller.	

Diagram 2.34 (Part 1 of 4). Perform Display Station Error Recovery

DESCRIPTION	MODULE/ ROUTINE
○ If hardware error: – Free TP buffer. – If error occurred while processing inquiry: a. Recycle (save) error, if not already done. b. Go to 3 Ⓐ. – If TUB owned by user: a. If format has been put to TUB, go to 1 Ⓐ. b. Otherwise: ○ Clear ERB (ignore error). ○ Return to caller. – If TUB owned by command processor: a. Recycle error if not already done. b. If TUB is not signed on: ○ Set informational message flag. ○ If TUB is the system console: – Set on console check. – Clear ERB. – Return to caller. ○ Otherwise: – Set up ERB to send message. – Return to caller. c. If TUB in error is not the system console: ○ Set up ERB to send 1, 3 halt message. ○ If TUB is an inquiry TUB, set up ERB to send 1, 2, halt message. ○ Set up ERB fields to send message. ○ Return to caller. Ⓒ d. If console error is due to a recovery attempt, go to 1 Ⓔ. e. Otherwise: ○ Wait 7 seconds unless entered due to SYS/REQ sequence. ○ Reset ERB and attempt to clear screen. ○ If clear fails, go to 1 Ⓔ. ○ Otherwise: – If error occurred during IPL: a. Set off console check. b. Clear ERB. c. Return to caller. – If STATUS display was active at time of error:	#SVWSR
a. Update or end STATUS.	#CCSM
– b. If update was performed, go to 1 Ⓓ. – Determine console mode at time of error: a. If in forced console mode (error occurred while running user program): ○ If console screen was previously saved, restore console interface. ○ Otherwise, put up console display and go to 1 Ⓓ. b. If in console mode:	#SVWSR
○ Rebuild console display	#CCCO
○ Go to 1 Ⓓ. c. If in command or other mode: ○ Put up last system display. ○ Go to 1 Ⓓ. Ⓓ – If recovery succeeded: a. Set off console check. b. If console mode was not forced, set up ERB to send message. c. Otherwise, clear ERB. d. Return to caller. Ⓔ – If recovery failed: a. Set up ERB to allow SYS/REQ. b. Set on console check. c. Free TP buffer. d. Return to caller.	#SVWSR

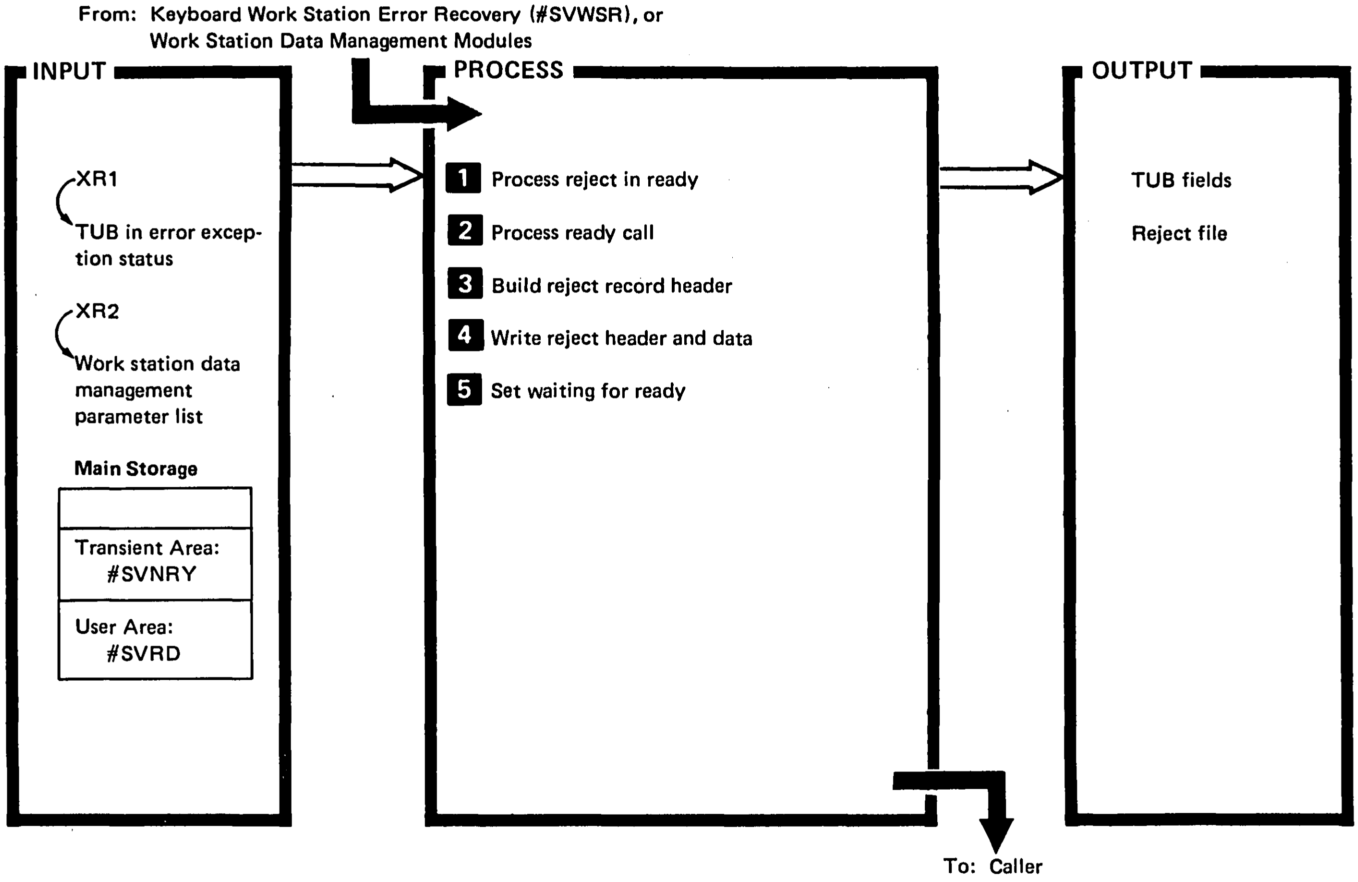
Diagram 2.34 (Part 2 of 4). Perform Display Station Error Recovery

DESCRIPTION	MODULE/ ROUTINE
2 If message sent does not require a response: ● Initialize ERB. ● Return to caller. Otherwise: ● If message was not due to a hardware error on the system console: – Set waiting for response in TUBERPCT. – Clear error aid (TUBERAID = 0). – Return to caller. ● If message was sent due to a hardware error on the system console: – Put console TUB on vertical tubchain. – Mark TUB as console mode forced. – Go to **1** Ⓒ. **3** If message response for a TUB owned by a user: ● Post TUB complete with error. ● If 2 option taken, unsuspend user task. ● If 3 or D option taken,	# SVWSR
ABTRM the task and clear ERB.	#CCCM
● Return to caller. If message response for a TUB owned by the command processor: ● If option 1 taken: – Clear ERB. – Attempt to clear screen in error. – If clear screen failed, return to caller. Otherwise, try to put up last system display or, if status was active,	#SVWSR
refresh STATUS display.	#CCSM
● If option 2 taken: Ⓐ – Copy ERB of inquiry TUB to next TUB on horizontal chain.	#SVWSR
– Perform pseudo resume of interrupted task. – Rechain interrupted TUB. – Free inquiry JCB, WSWA, and TUB.	#CPIQ
– Go to **1** Ⓐ. ● If option 3 taken: – Sign off work station.	#SVWSR
– Clear ERB.	#CCOF
– Return to caller. **4** Set up ERB to get MIC. Call control storage transient to get message ID. Go to **1**. **5** If ready is pending, go to **5** Ⓐ. If ready function is in progress (reject and ready occurred while processing a previous reject): ● Post TUB complete in error. ● Clear ERB. ● Return to caller. Otherwise: ● Set ready in TUB. ● If ready task not needed (TUBTCB not 0): – PIQ ACE. – Post TUB complete. – Return to caller.	#SVWSR

Diagram 2.34 (Part 3 of 4). Perform Display Station Error Recovery

DESCRIPTION	MODULE/ ROUTINE
● Otherwise: – If command processor owns TUB: a. If error occurred during Sys/req – Enter sequence: ● Set off reject/ready indicators. ● Set command processor aid to SYS/REQ. ● Clear ERB. ● Return to caller. b. Otherwise: ● If a read command was rejected: – Reset TUB to invite status. – Clear ERB. – PIQ ACE. – Post TUB invite complete. – Return to caller. A Build attach parameter list.	#SVWSR
Attach ready task.	#SVAT
If attach was successful, clear ERB. Otherwise, set ready pending. Return to caller.	#SVWSR

Diagram 2.34 (Part 4 of 4). Perform Display Station Error Recovery


From: Keyboard Work Station Error Recovery (#SVWSR), or
Work Station Data Management Modules
INPUT
XR1
TUB in error exception status
XR2
Work station data management parameter list
Main Storage
Transient Area: #SVNRY
User Area: #SVRD
PROCESS
1 Process reject in ready
2 Process ready call
3 Build reject record header
4 Write reject header and data
5 Set waiting for ready
OUTPUT
TUB fields
Reject file
To: Caller

DESCRIPTION	MODULE/ ROUTINE
1 If processing a reject in ready function:	#SVNRY
● Zero TUB completion code.	
● Set up ERB to wait for ready.	
● Free TP buffer if required.	
● Return to caller.	
2 If processing a ready call:	
● Initialize ERB.	
● Perform ready function:	#SVUR
– Push user and load #SVRD.	
– Save invite status.	#SVRD
– Move all reject records for TUB to display screen via #DWDM.	
– Restore invite status and call #SVUR.	
– Pull user into main storage.	#SVUR

Diagram 2.35 (Part 1 of 2). Perform Display Station Error Recovery for Device Not Ready Function

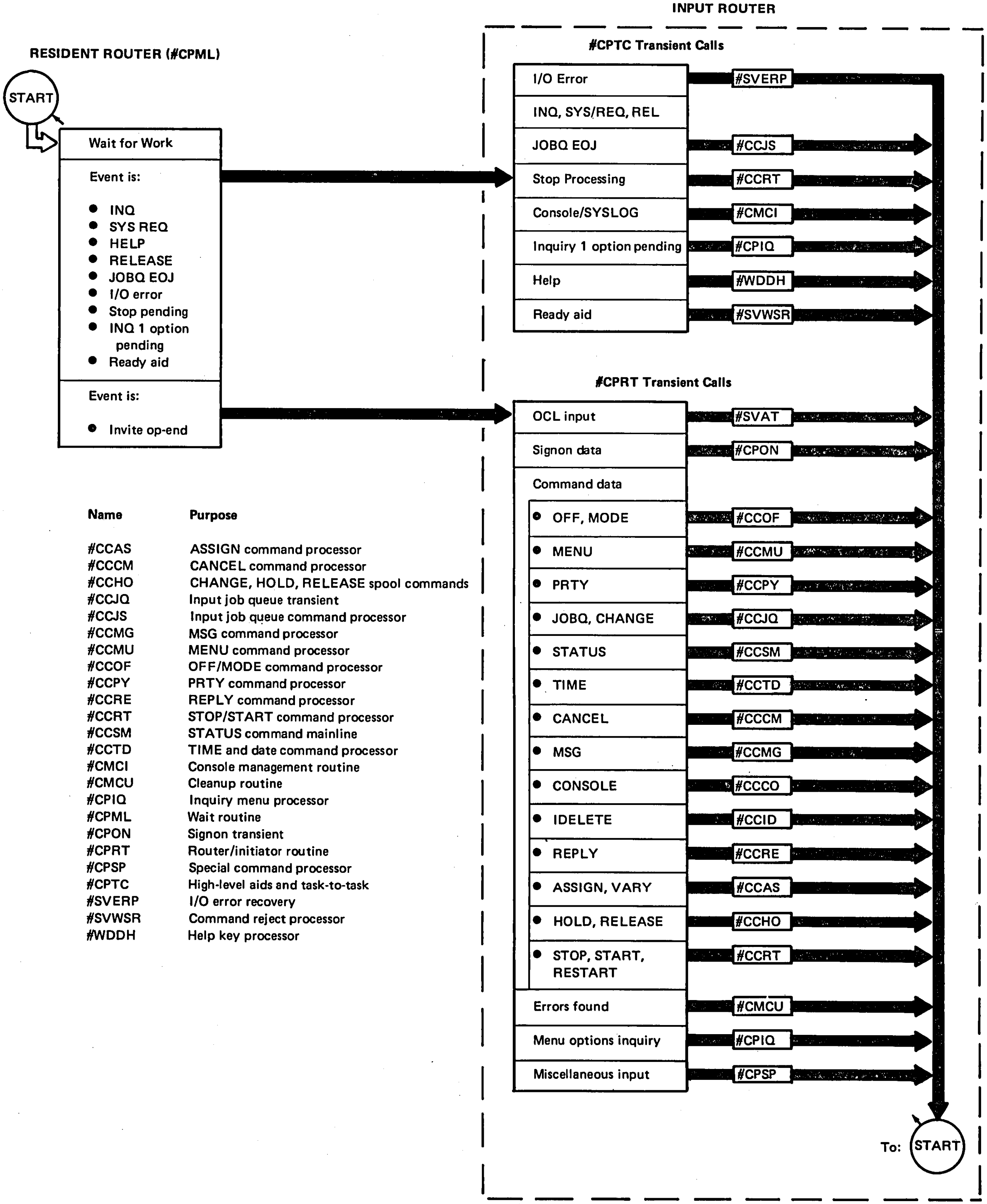
DESCRIPTION	MODULE/ ROUTINE
3 If command reject: ● Build command reject record header. – Save the following fields which may be part of the reject header: a. TUBCMND – command code. b. TUBCMOD – command modifier. c. TUBCOUNT – data transfer count. d. Work station data management parameter list. – Determine type of reject record: a. If rejected command is invite/cancel: ● Set header type as invite/cancel. ● Set record length equal to header length. ● Go to 4. b. If rejected command is read input or read screen go to 4. c. If rejected command is save table, save screen, restore table, or restore screen: ● If required, recycle error. ● Set header type to save/restore. ● Set record length equal to header length. ● Save SS address of save restore area. ● Go to 4. d. If rejected command is output and not a restore operation: ● Recycle error if required. ● If rejected command is to be saved as an output type 1: – Save TUBCTSAV (saved input length). – Set header type to output type 1. – Set length of record equal to header length plus length of data (TUBCOUNT). – Go to 4. ● If rejected command is to be saved as an output type 2: – Set header type to output type 2. – If indicators present, save indicators in reject header. – Set length of record equal to header length plus length of output data (WDOUTL). 4 Determine if reject record is to be written: ● If reject is being recycled: – Free up TP buffer if required. – Return to caller. ● If reject command is read type: – Zero TUB completion code. – Go to 5. ● If this is first command reject, initialize reject file current sector and relative displacement. ● Calculate space remaining in reject file. ● If reject record will fit in file: – Update reject file current sector and relative displacement.	#SVNRY
– Write out reject record header and any data.	Disk IOS
– Post TUB complete. 5 Set up ERB to wait for ready: ● If first command reject, set up ERB to wait for ready. ● Free TP buffer if required. ● Return to caller.	#SVNRY

Diagram 2.35 (Part 2 of 2). Perform Display Station Error Recovery for Device Not Ready Function

Program Organization

Figures 2-3 through 2-32 show the control flow for the command processor functions. They are:

- Router
- Sign-on
- Command processing
- Job initiator
- High level aids and task-to-task communications
- Inquiry and resume
- Special command processor
- Command processor/work station data management interface
- Console management
- Cleanup
- Password security
- Work station error recovery



Name	Purpose
#CCAS	ASSIGN command processor
#CCCM	CANCEL command processor
#CCHO	CHANGE, HOLD, RELEASE spool commands
#CCJQ	Input job queue transient
#CCJS	Input job queue command processor
#CCMG	MSG command processor
#CCMU	MENU command processor
#CCOF	OFF/MODE command processor
#CCPY	PRTY command processor
#CCRE	REPLY command processor
#CCRT	STOP/START command processor
#CCSM	STATUS command mainline
#CCTD	TIME and date command processor
#CMCI	Console management routine
#CMCU	Cleanup routine
#CPIQ	Inquiry menu processor
#CPML	Wait routine
#CPON	Signon transient
#CPRT	Router/initiator routine
#CPSP	Special command processor
#CPTC	High-level aids and task-to-task
#SVERP	I/O error recovery
#SVWSR	Command reject processor
#WDDH	Help key processor

Figure 2-3. Router Control Flow

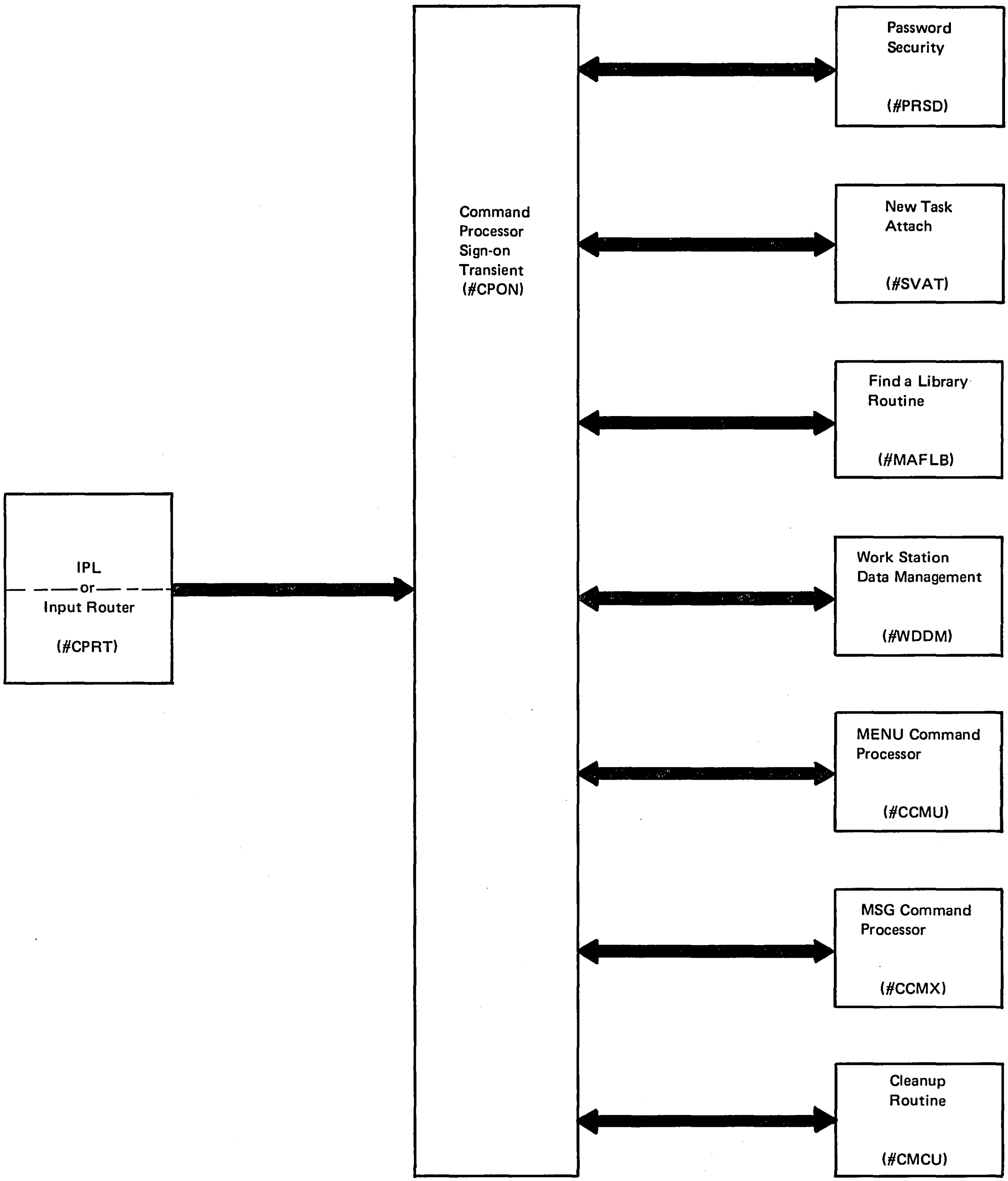

Password Security (#PRSD)
New Task Attach (#SVAT)
Find a Library Routine (#MAFLB)
Work Station Data Management (#WDDM)
MENU Command Processor (#CCMU)
MSG Command Processor (#CCMX)
Cleanup Routine (#CMCU)
Command Processor Sign-on Transient (#CPON)
IPL
or
Input Router
(#CPRT)

Figure 2-4. Command Processor Sign-on Control Flow

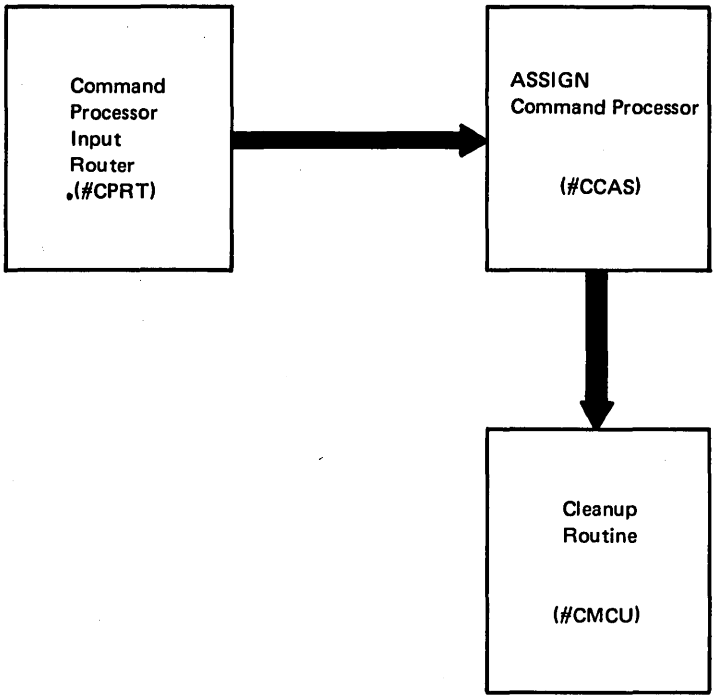


Figure 2-5. ASSIGN and VARY Command Processing Control Flow

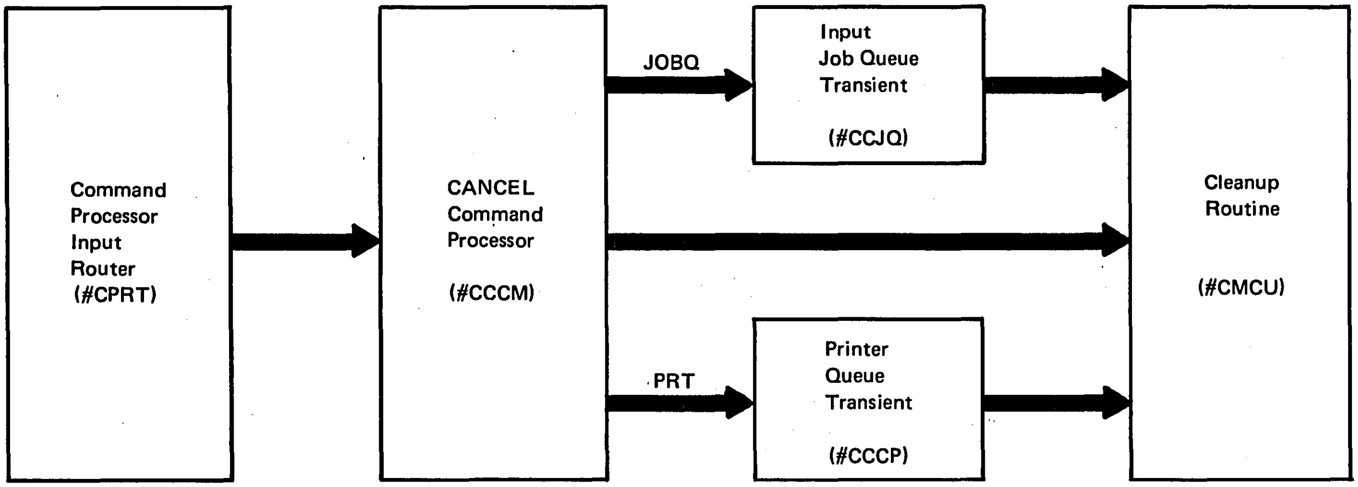


Figure 2-6. CANCEL Command Processing Control Flow

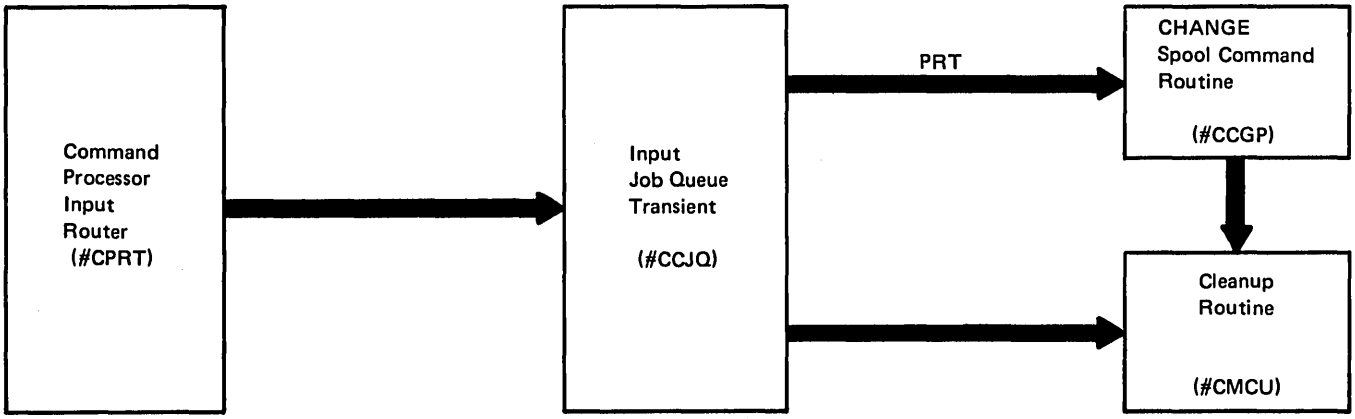


Figure 2-7. CHANGE and JOBQ Command Processing Control Flow

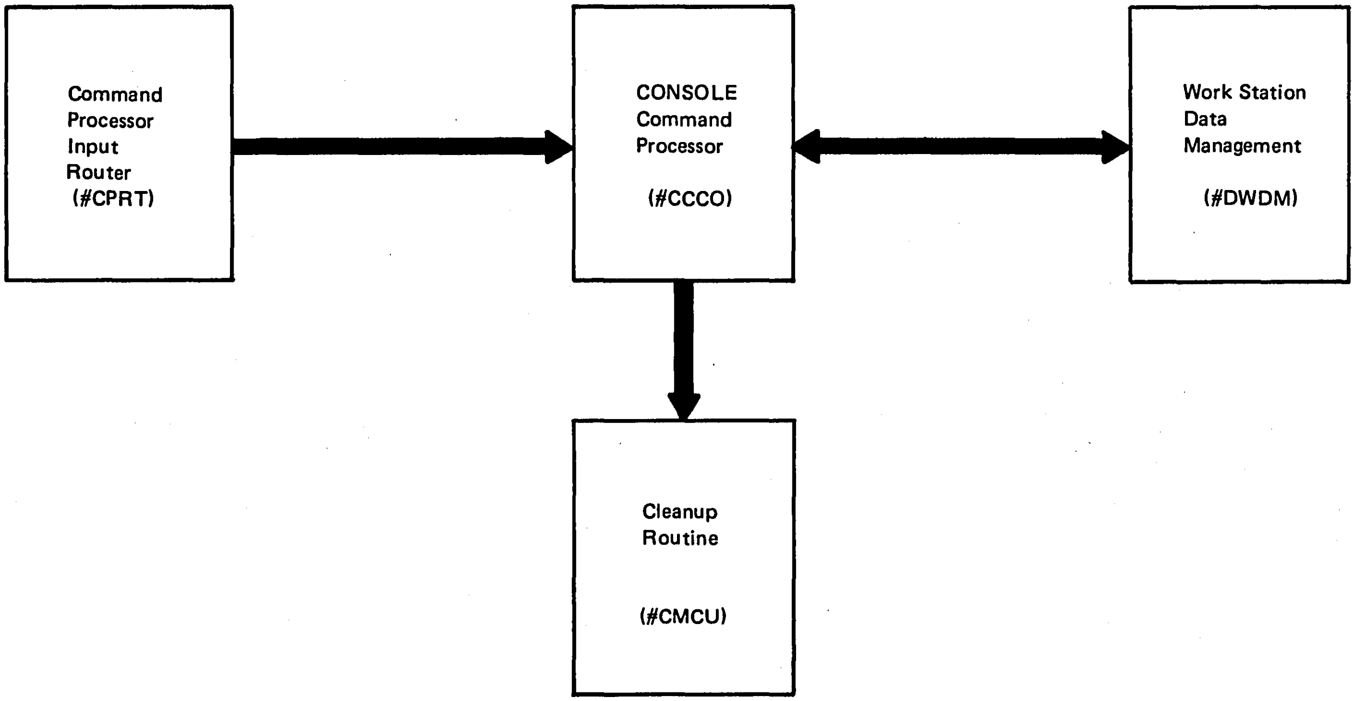


Figure 2-8. CONSOLE Command Processing Control Flow

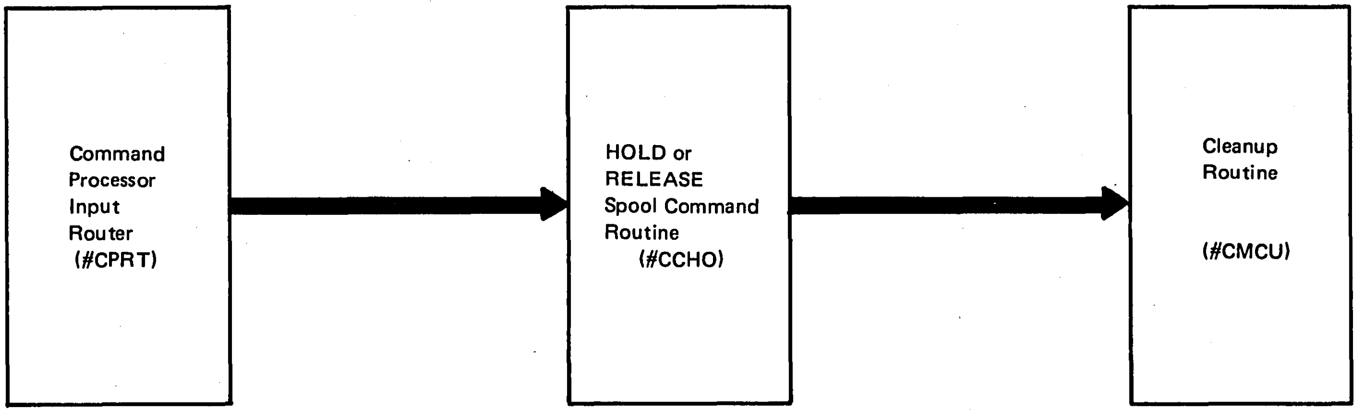


Figure 2-9. HOLD and RELEASE Command Processing Control Flow

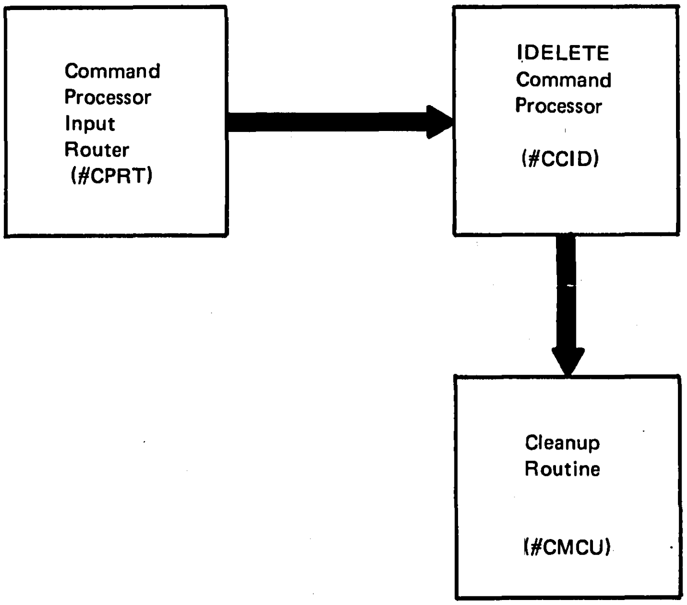


Figure 2-10. IDELETE Command Processing Control Flow

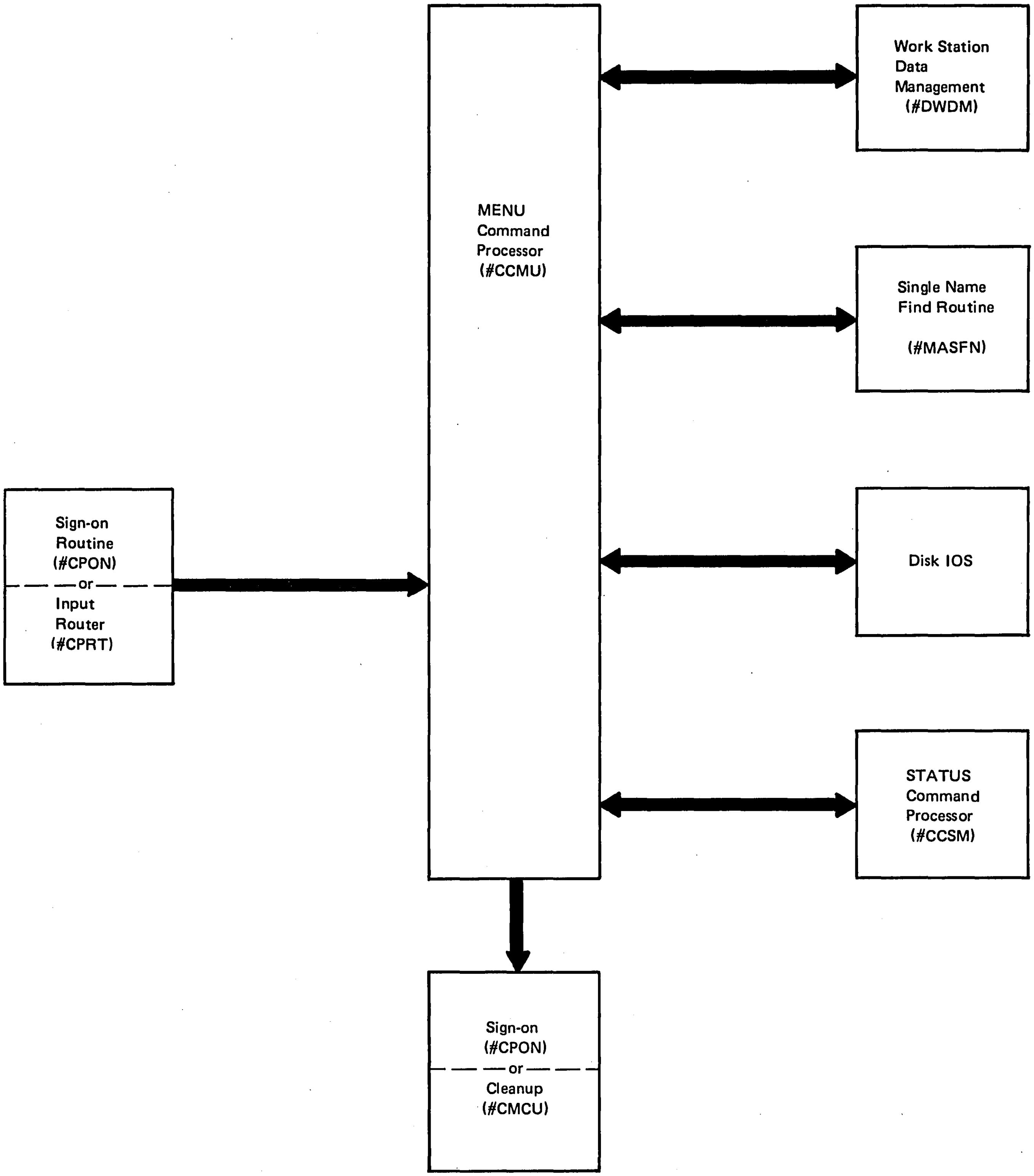

Work Station Data Management (#DWDM)
MENU Command Processor (#CCMU)
Single Name Find Routine (#MASFN)
Sign-on Routine (#CPON)
or
Input Router (#CPRT)
Disk IOS
STATUS Command Processor (#CCSM)
Sign-on (#CPON)
or
Cleanup (#CMCU)

Figure 2-11. MENU Command Processing Control Flow

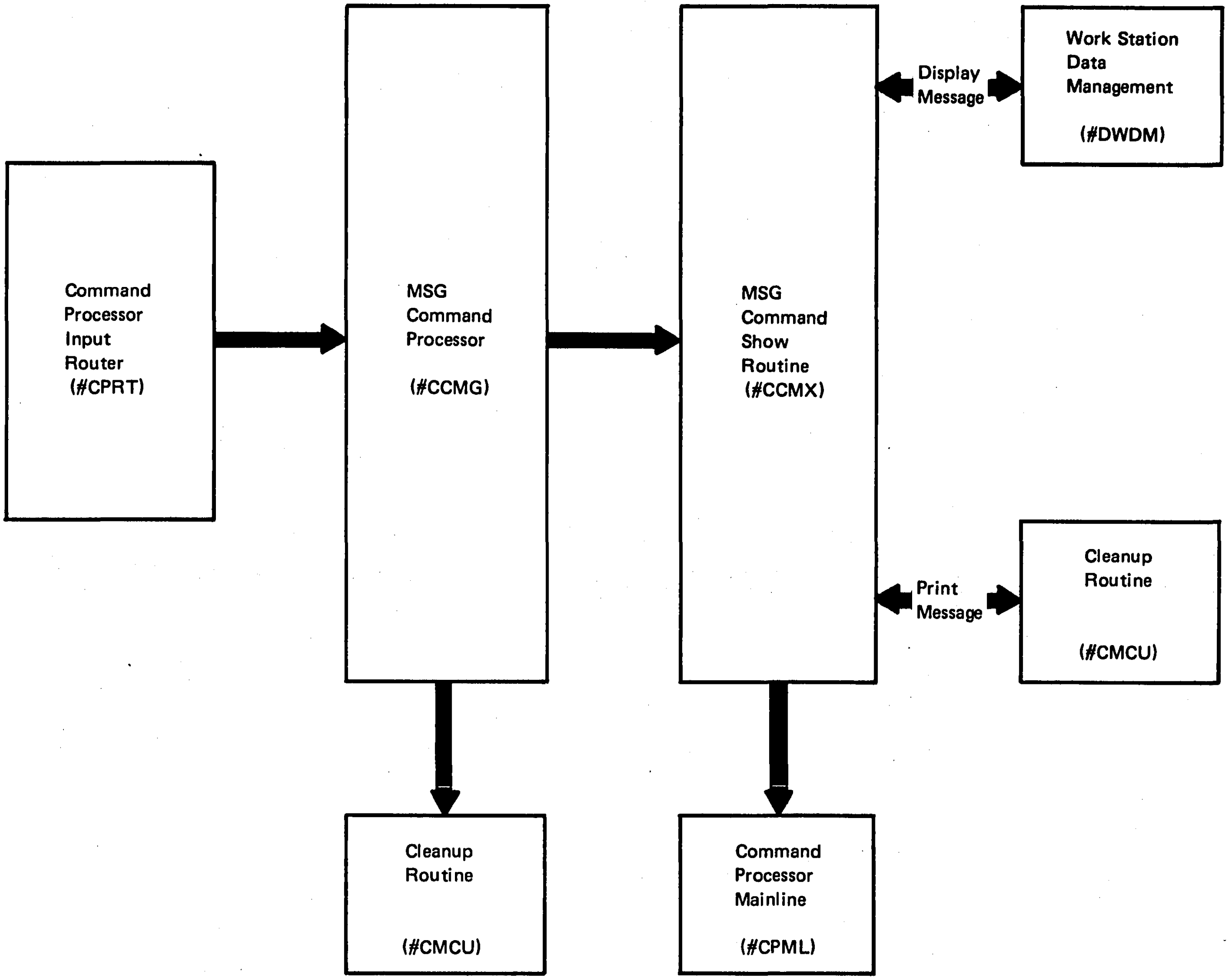

Command Processor Input Router (#CPRT)
MSG Command Processor (#CCMG)
MSG Command Show Routine (#CCMX)
Display Message
Work Station Data Management (#DWDM)
Print Message
Cleanup Routine (#CMCU)
Cleanup Routine (#CMCU)
Command Processor Mainline (#CPML)

Figure 2-12. MSG Command Processing Control Flow

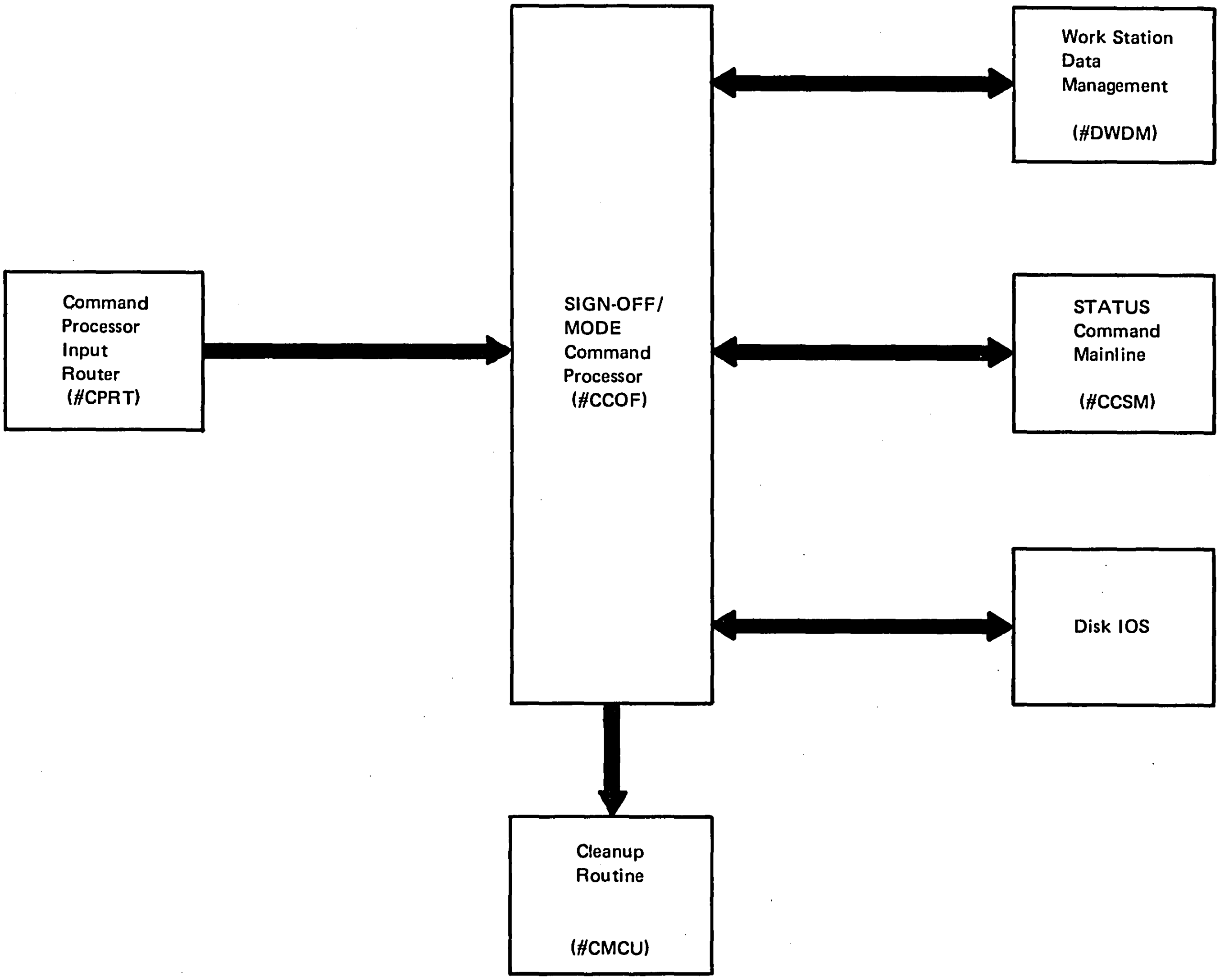


Figure 2-13. MODE and OFF Command Processing Control Flow

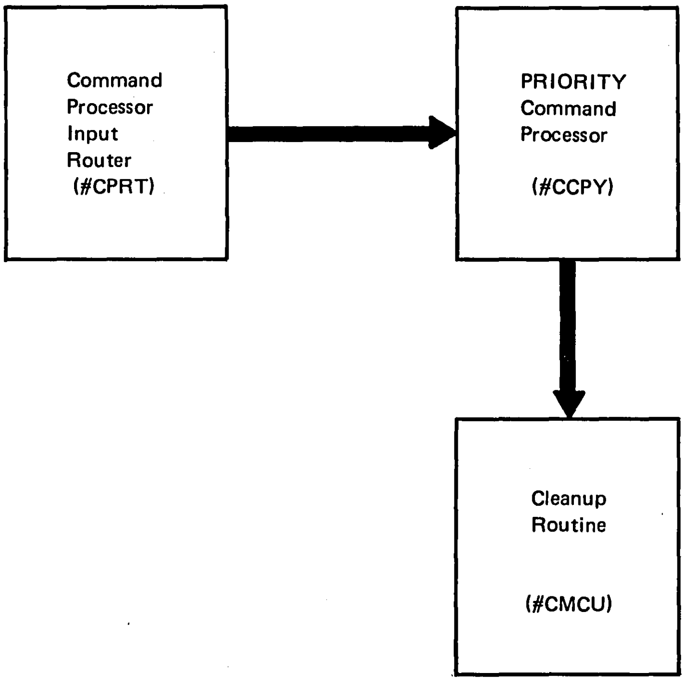


Figure 2-14. PRIORITY Command Processing Control Flow

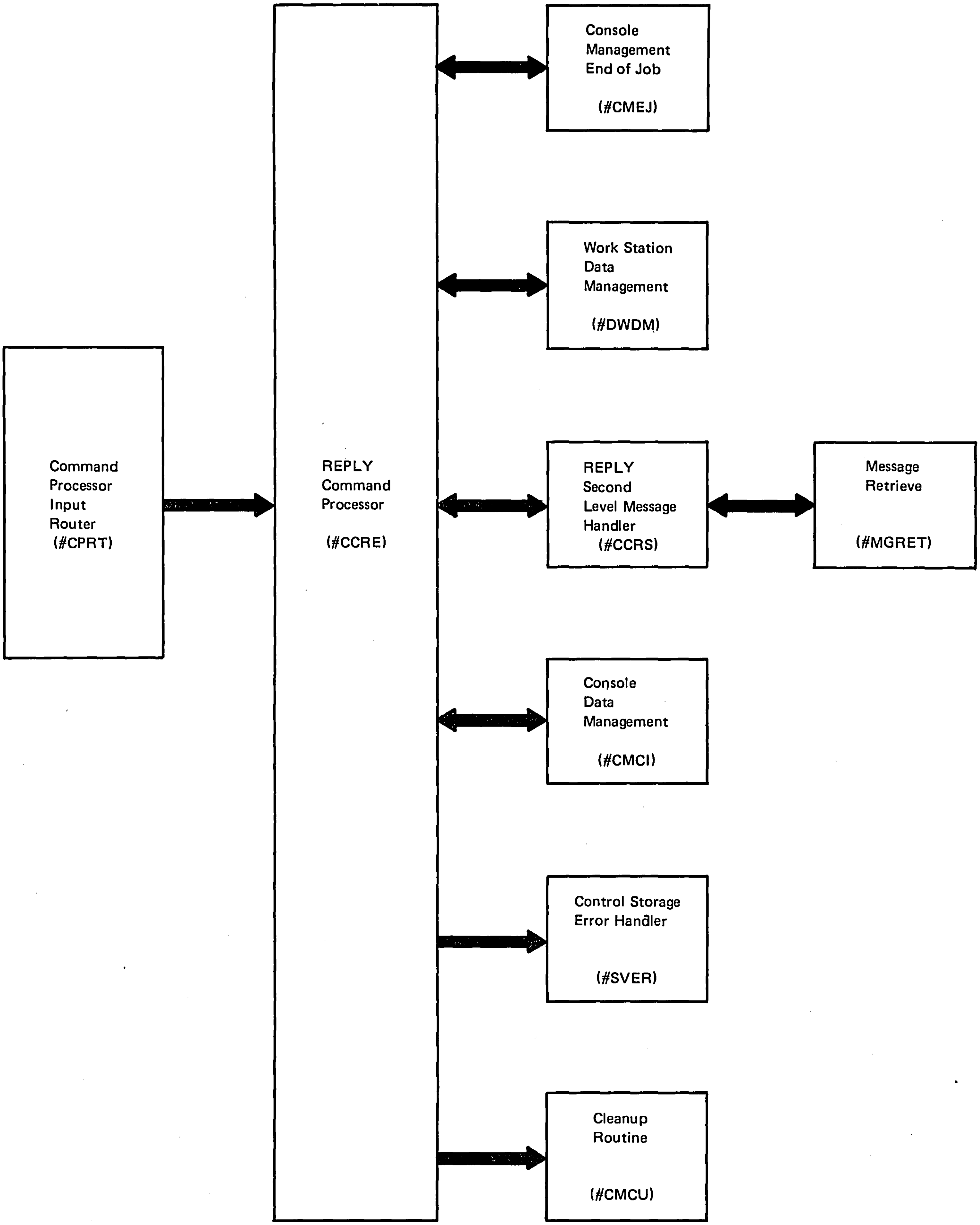
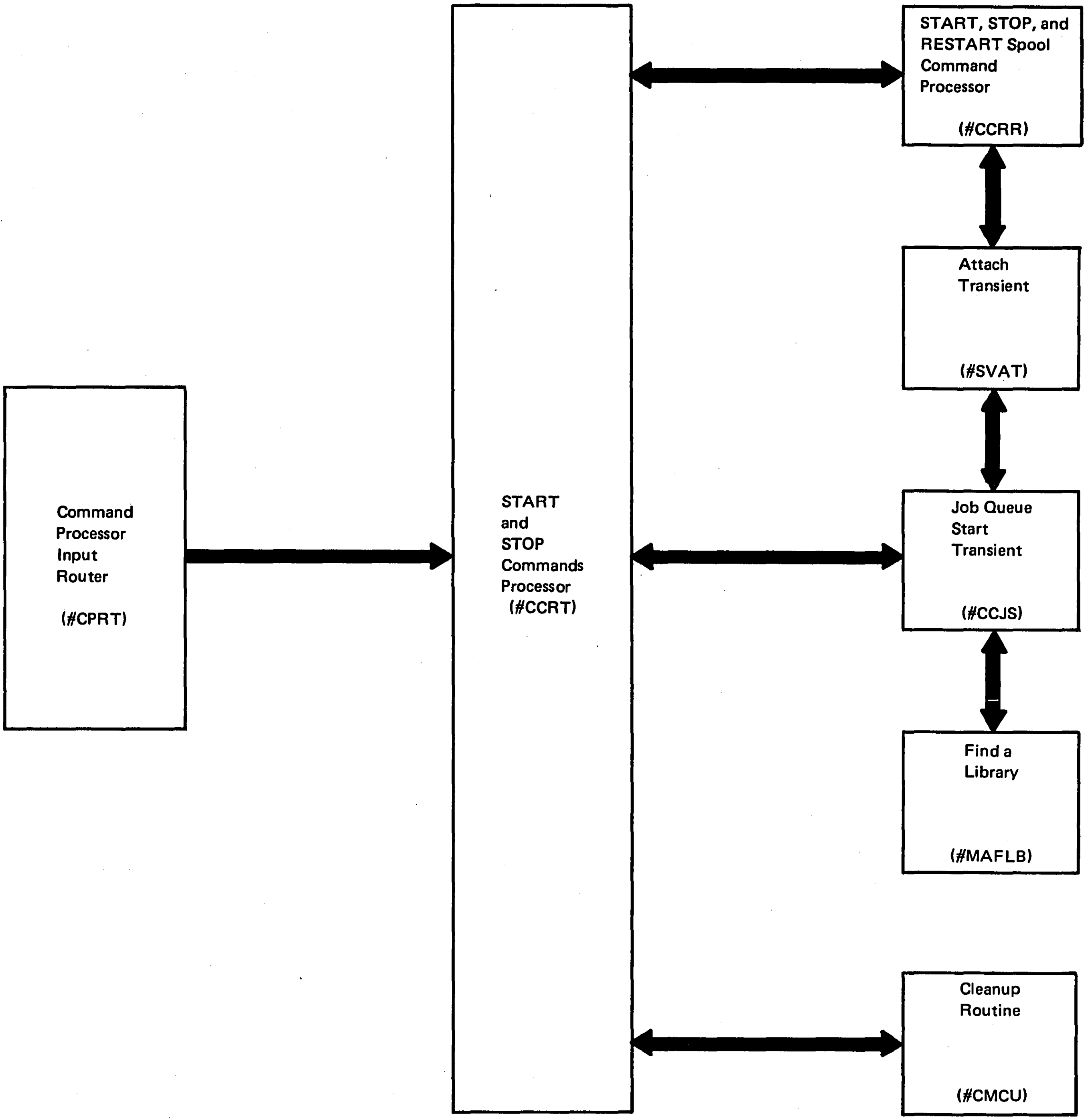


Figure 2-15. REPLY Command Processing Control Flow



START, STOP, and RESTART Spool Command Processor
(#CCRR)
Attach Transient
(#SVAT)
Command Processor Input Router
(#CPRT)
START and STOP Commands Processor
(#CCRT)
Job Queue Start Transient
(#CCJS)
Find a Library
(#MAFLB)
Cleanup Routine
(#CMCU)

Figure 2-16. START, STOP, and RESTART Command Processing Control Flow

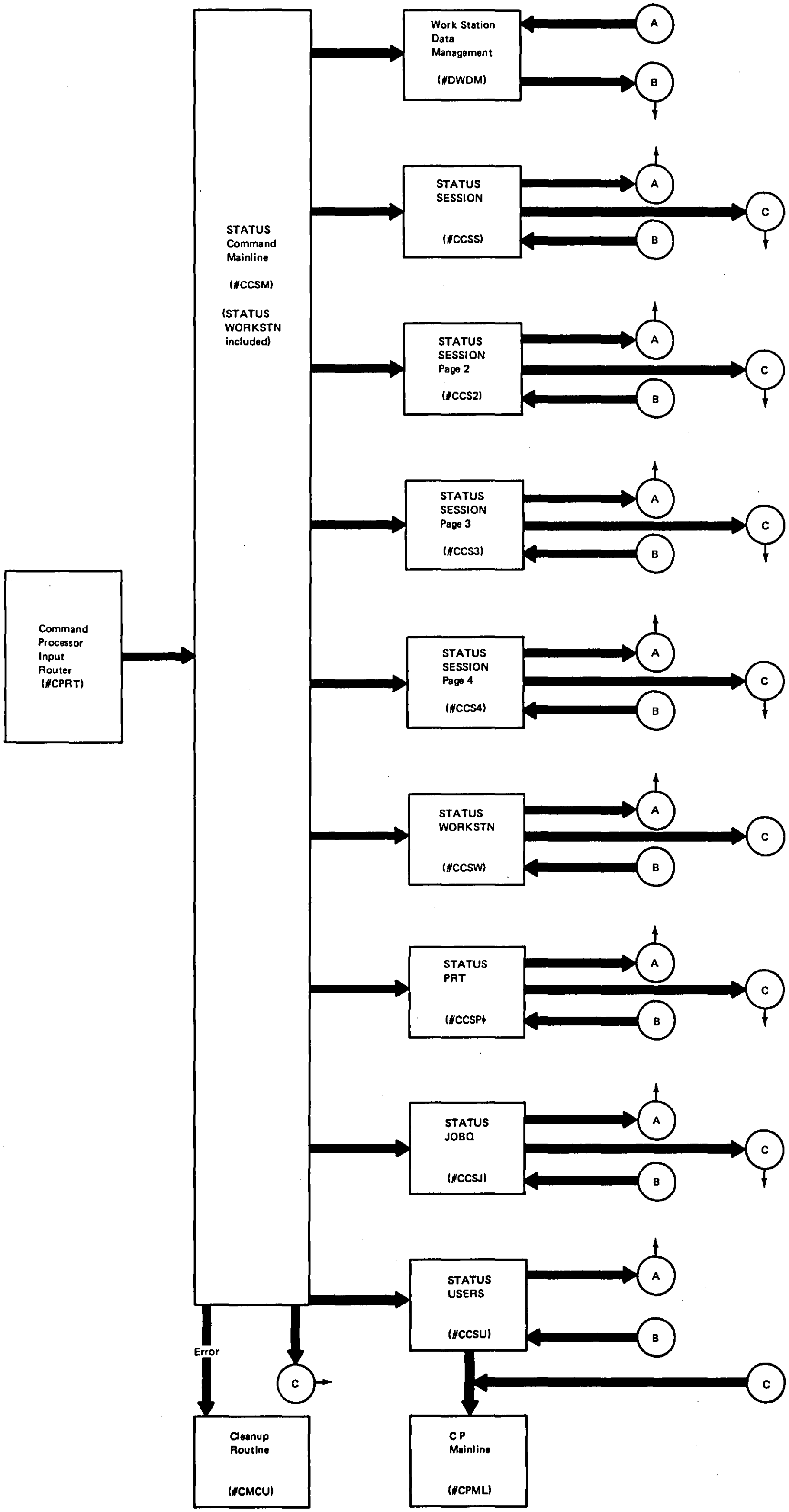

Work Station Data Management (#DWDM)
A
B
STATUS SESSION (#CCSS)
C
STATUS Command Mainline (#CCSM) (STATUS WORKSTN included)
STATUS SESSION Page 2 (#CCS2)
STATUS SESSION Page 3 (#CCS3)
Command Processor Input Router (#CPRT)
STATUS SESSION Page 4 (#CCS4)
STATUS WORKSTN (#CCSW)
STATUS PRT (#CCSP)
STATUS JOBQ (#CCSJ)
STATUS USERS (#CCSU)
Error
Cleanup Routine (#CMCU)
CP Mainline (#CPML)

Figure 2-17. STATUS Command Processing Control Flow

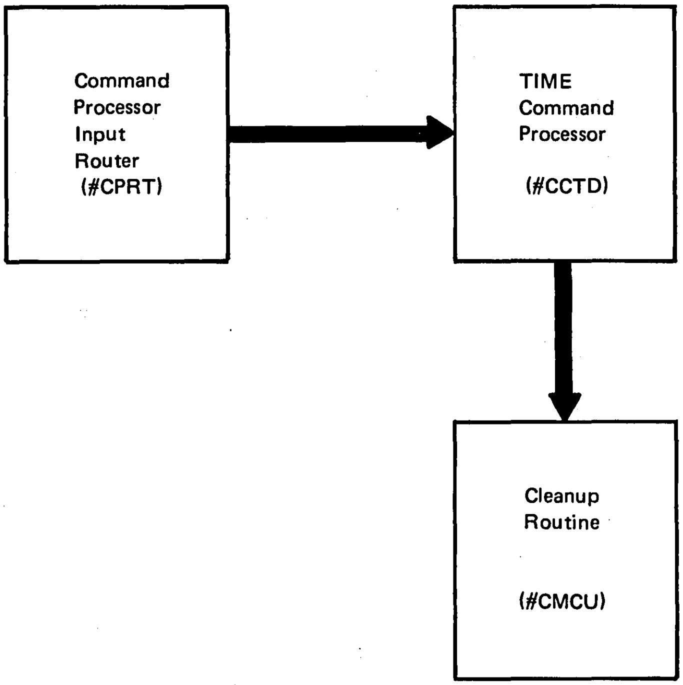

Command Processor Input Router (#CPRT)
TIME Command Processor (#CCTD)
Cleanup Routine (#CMCU)

Figure 2-18. TIME Command Processing Control Flow

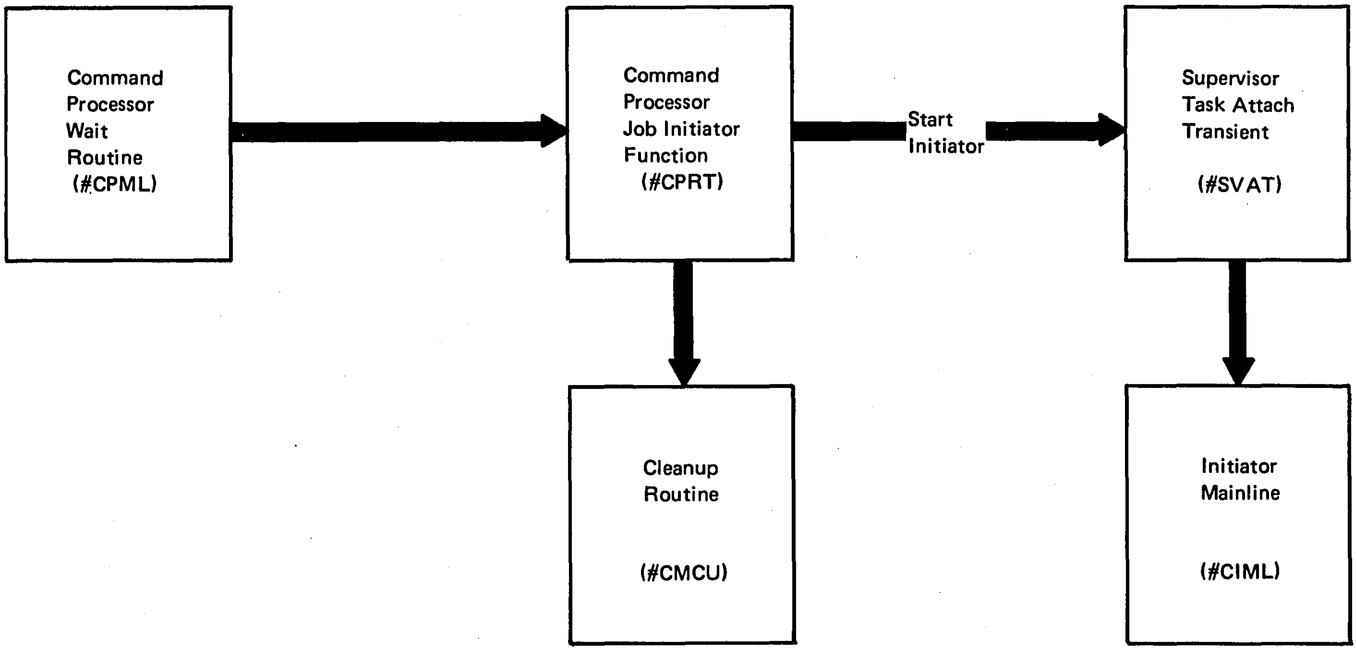

Command Processor Wait Routine (#CPML)
Command Processor Job Initiator Function (#CPRT)
Start Initiator
Supervisor Task Attach Transient (#SVAT)
Cleanup Routine (#CMCU)
Initiator Mainline (#CIML)

Figure 2-19. Job Initiator Control Flow

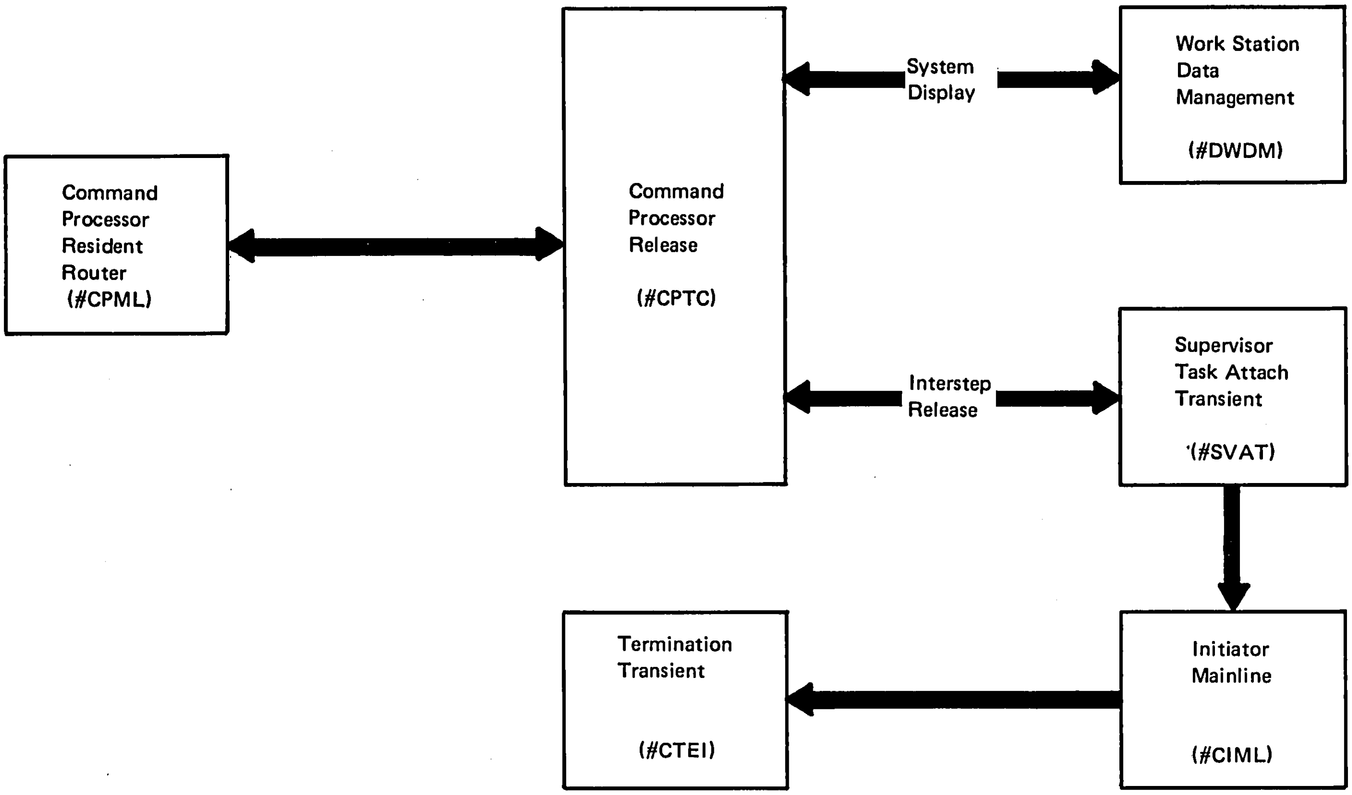


Figure 2-20. Release Control Flow

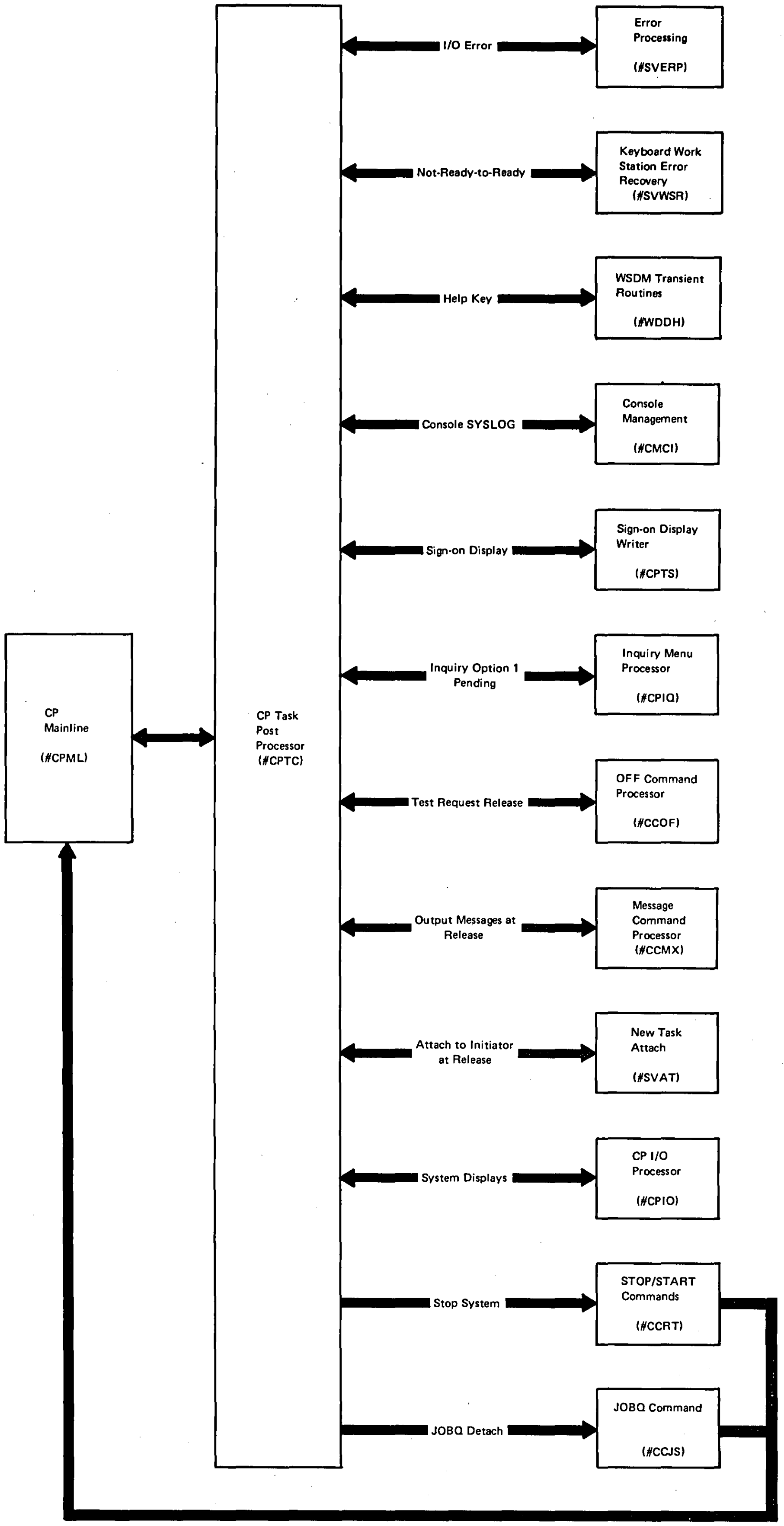

Error Processing (#SVERP)
I/O Error
Keyboard Work Station Error Recovery (#SVWSR)
Not-Ready-to-Ready
WSDM Transient Routines (#WDDH)
Help Key
Console Management (#CMCI)
Console SYSLOG
Sign-on Display Writer (#CPTS)
Sign-on Display
Inquiry Menu Processor (#CPIQ)
Inquiry Option 1 Pending
CP Mainline (#CPML)
CP Task Post Processor (#CPTC)
OFF Command Processor (#CCOF)
Test Request Release
Message Command Processor (#CCMX)
Output Messages at Release
New Task Attach (#SVAT)
Attach to Initiator at Release
CP I/O Processor (#CPIO)
System Displays
STOP/START Commands (#CCRT)
Stop System
JOBQ Command (#CCJS)
JOBQ Detach

Figure 2-21. High-Level Aids and Task-toTask Communications Control Flow Program Organization

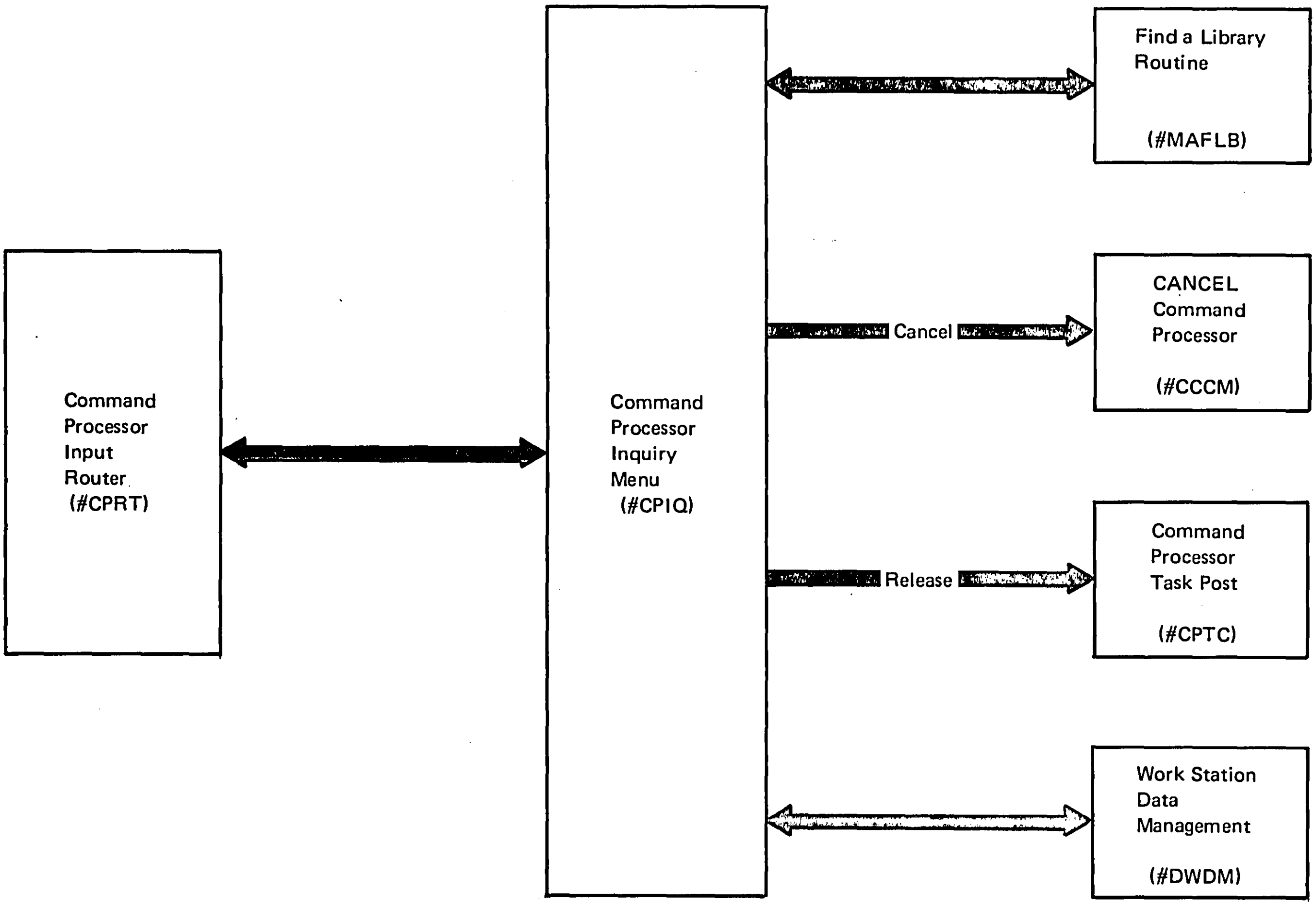

Command Processor Input Router (#CPRT)
Command Processor Inquiry Menu (#CPIQ)
Find a Library Routine (#MAFLB)
Cancel
CANCEL Command Processor (#CCCM)
Release
Command Processor Task Post (#CPTC)
Work Station Data Management (#DWDM)

Figure 2-22. Inquiry Menu Processing Control Flow

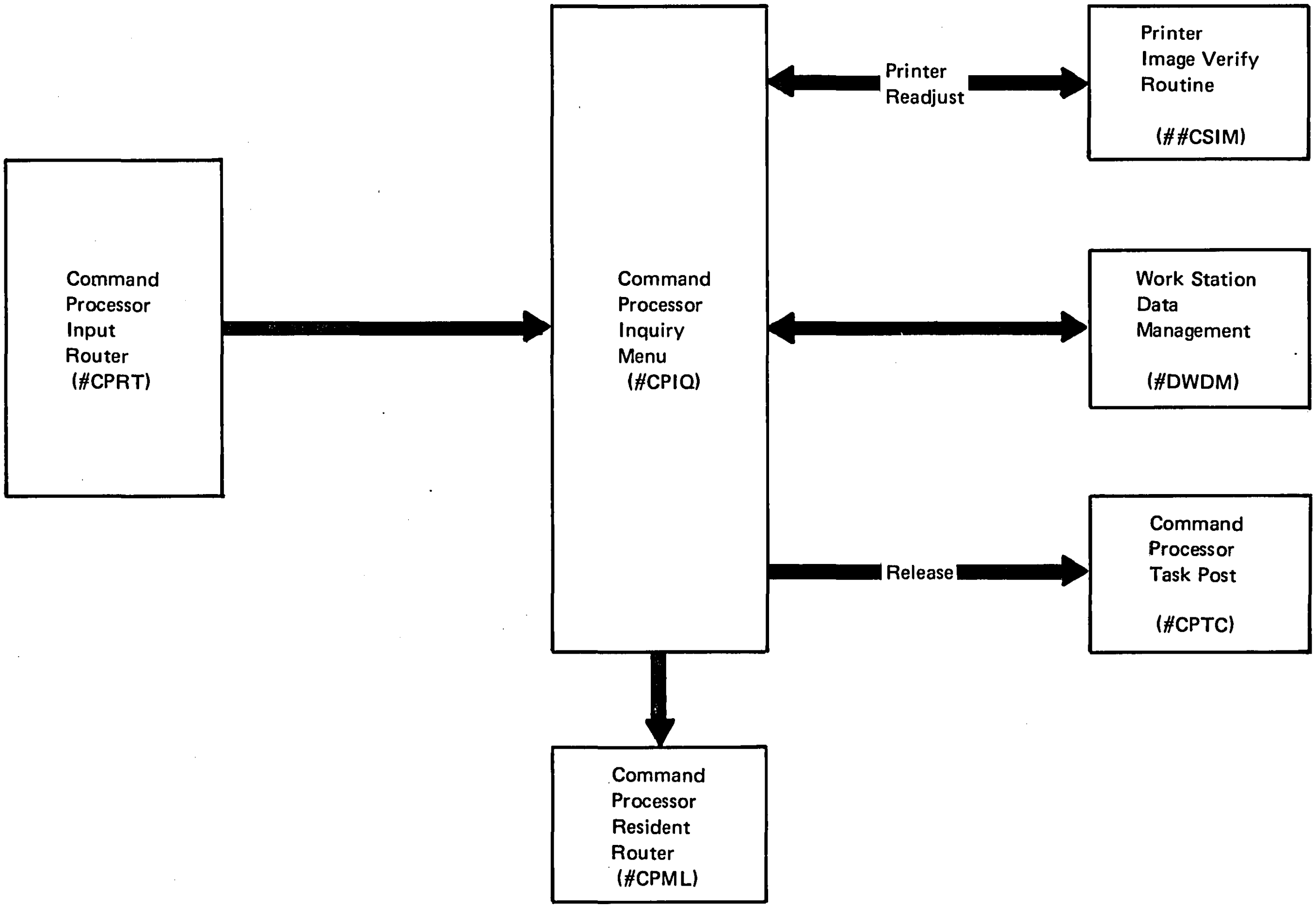

Command Processor Input Router (#CPRT)
Command Processor Inquiry Menu (#CPIQ)
Printer Readjust
Printer Image Verify Routine (##CSIM)
Work Station Data Management (#DWDM)
Release
Command Processor Task Post (#CPTC)
Command Processor Resident Router (#CPML)

Figure 2-23. Resume Function Control Flow

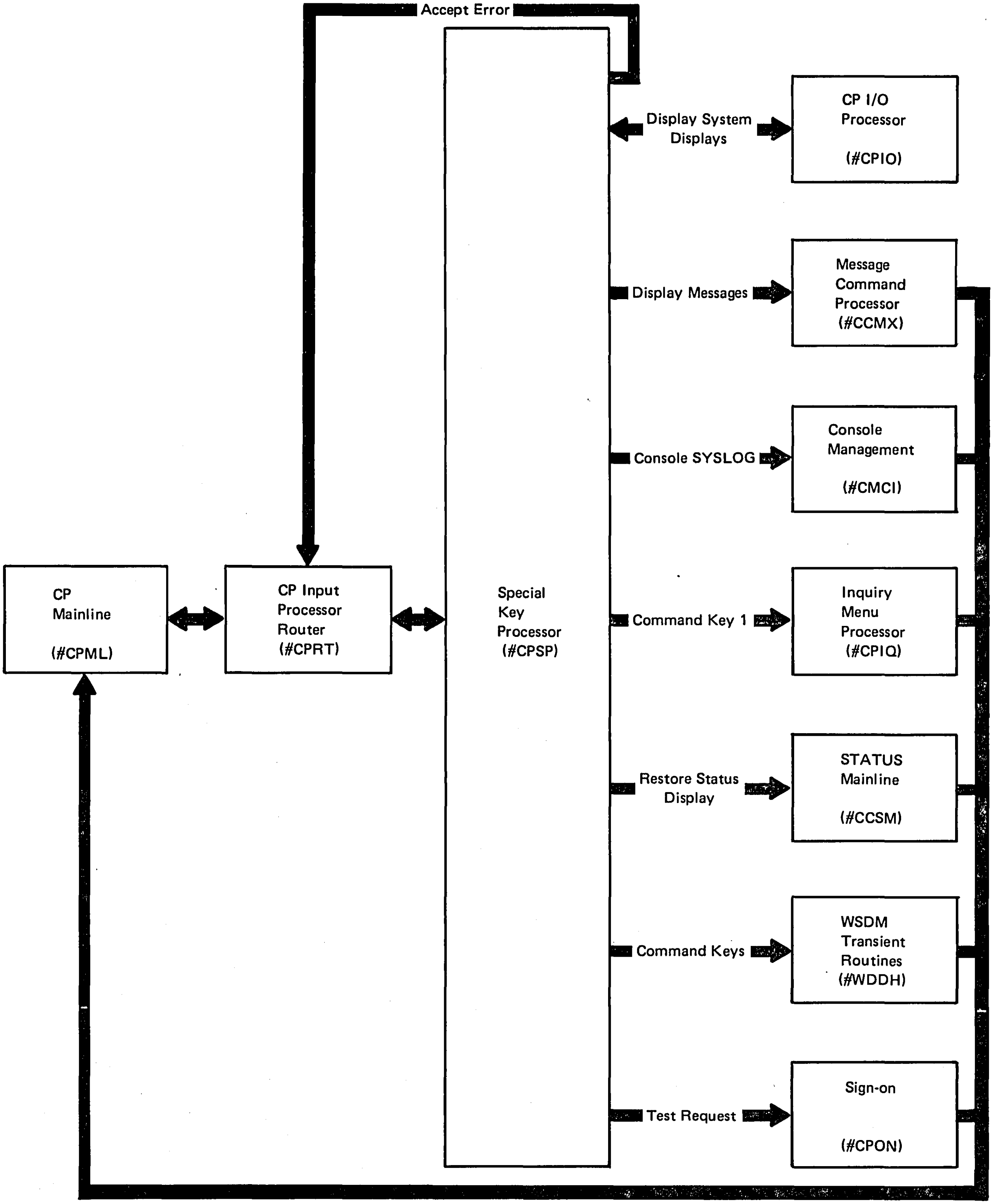


Figure 2-24. Special Command Processor Control Flow

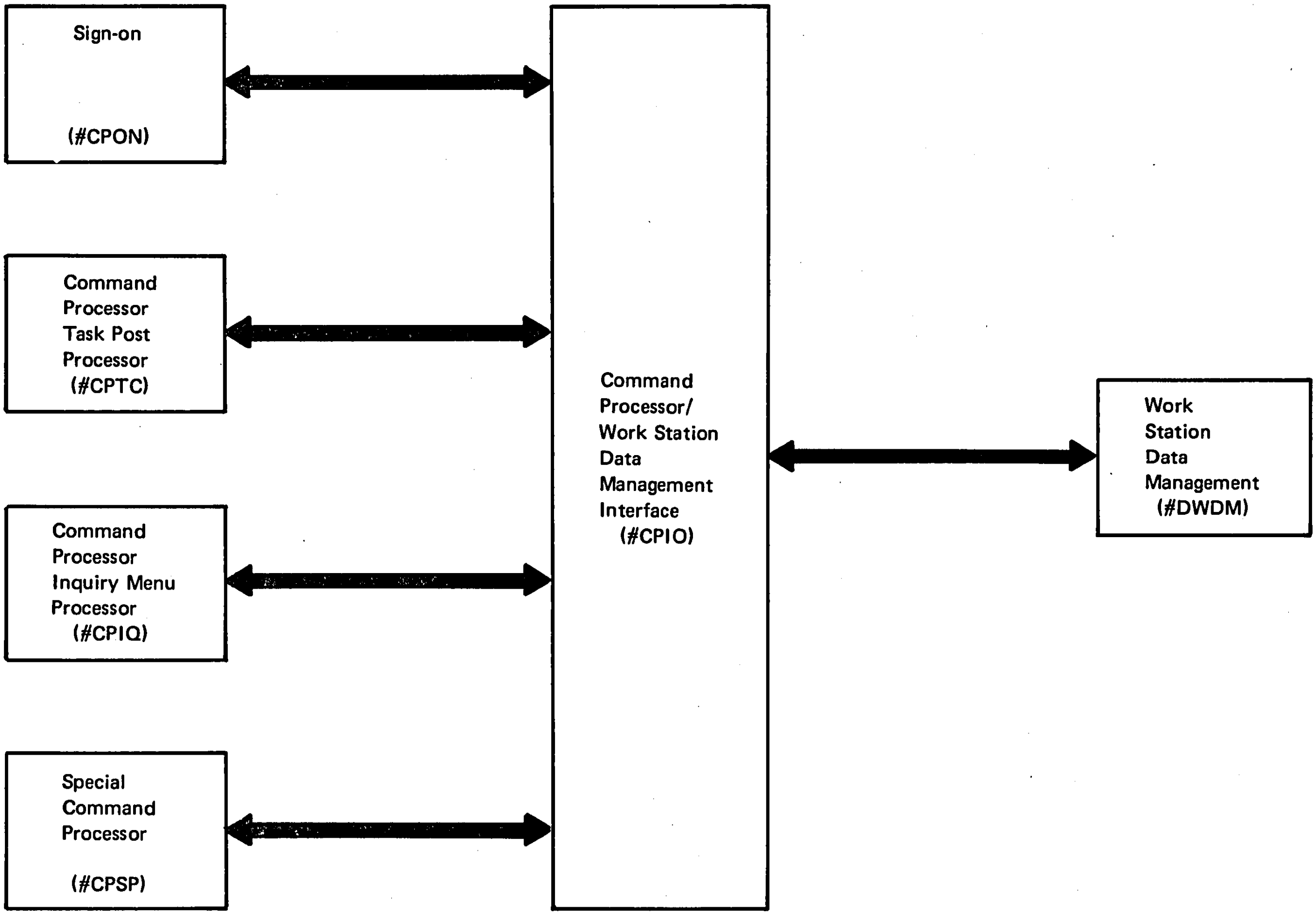

Sign-on
(#CPON)
Command Processor Task Post Processor (#CPTC)
Command Processor Inquiry Menu Processor (#CPIQ)
Special Command Processor (#CPSP)
Command Processor/ Work Station Data Management Interface (#CPIO)
Work Station Data Management (#DWDM)

Figure 2-25. Command Processor/Work Station Data Management Interface Control Flow

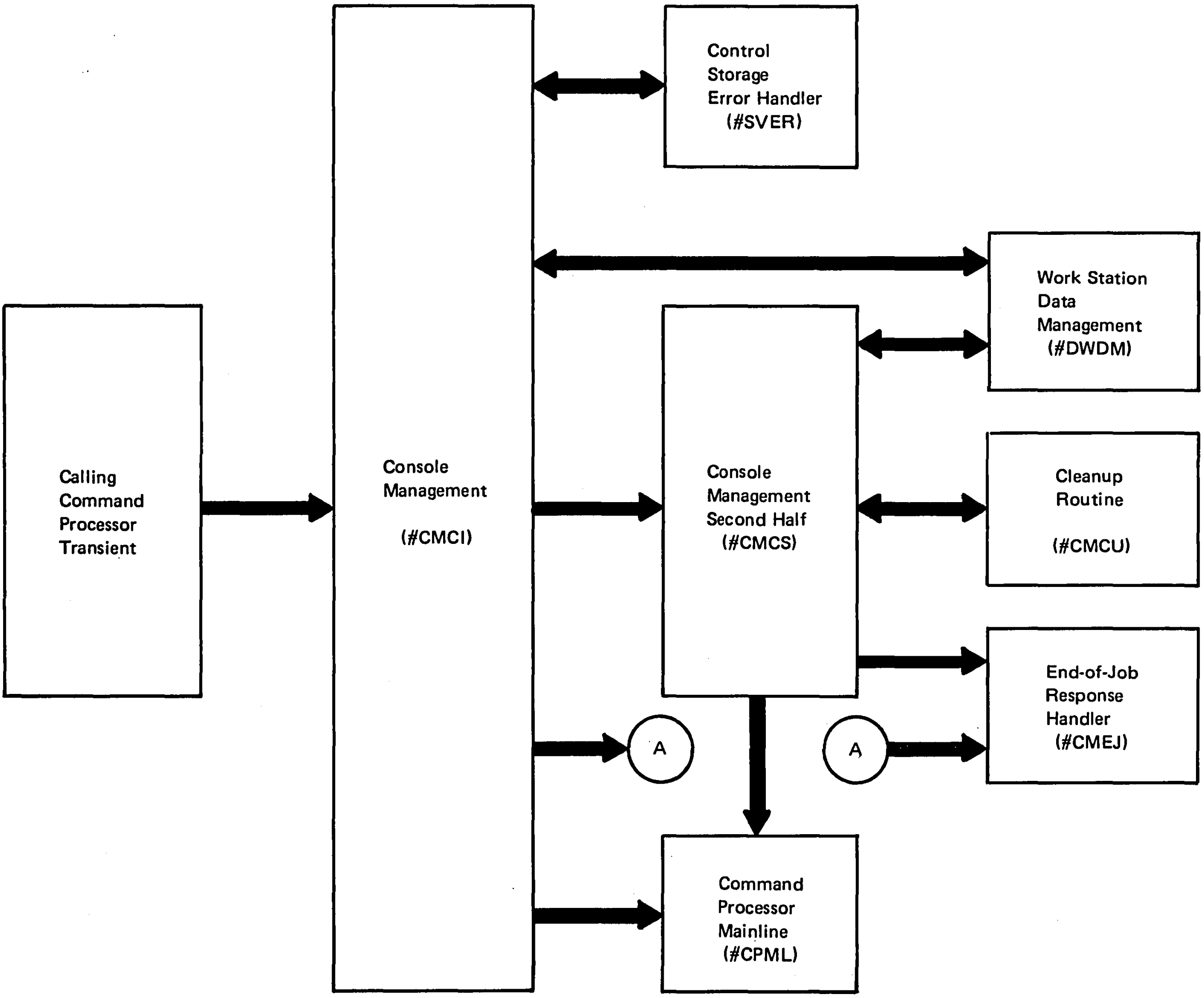

Control Storage Error Handler (#SVER)
Work Station Data Management (#DWDM)
Calling Command Processor Transient
Console Management (#CMCI)
Console Management Second Half (#CMCS)
Cleanup Routine (#CMCU)
End-of-Job Response Handler (#CMEJ)
A
A
Command Processor Mainline (#CPML)

Figure 2-26. System Console Logical I/O Interface Control Flow

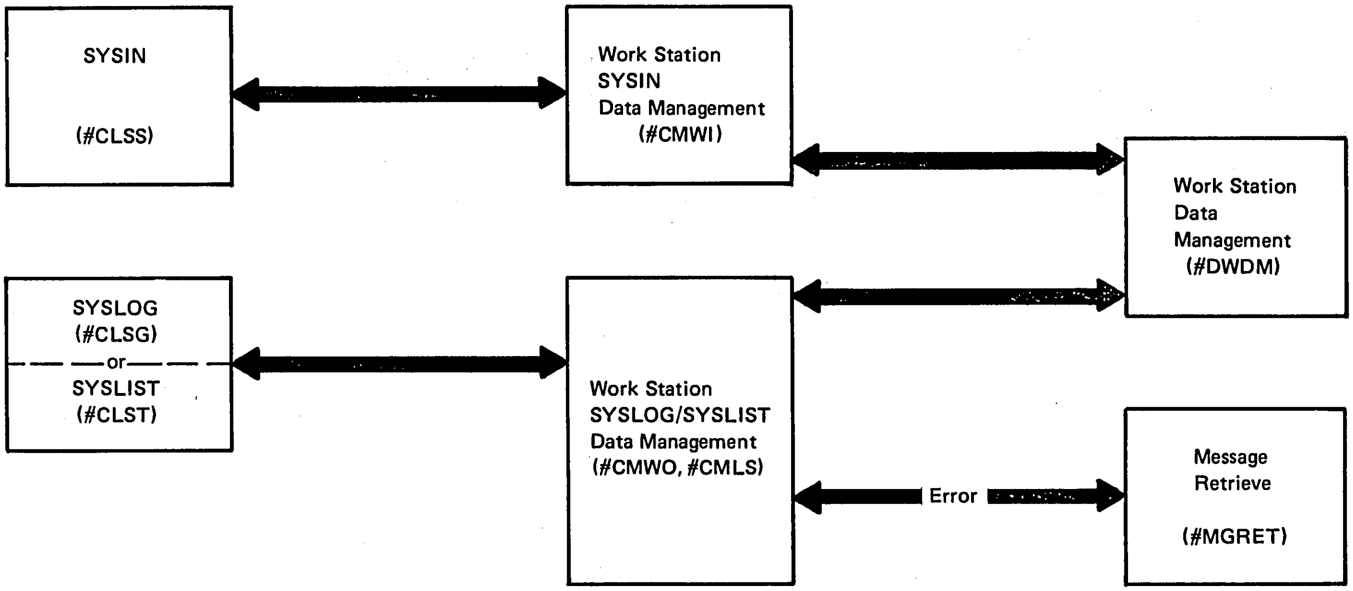


Figure 2-27. Work Station Logical I/O Interface Control Flow

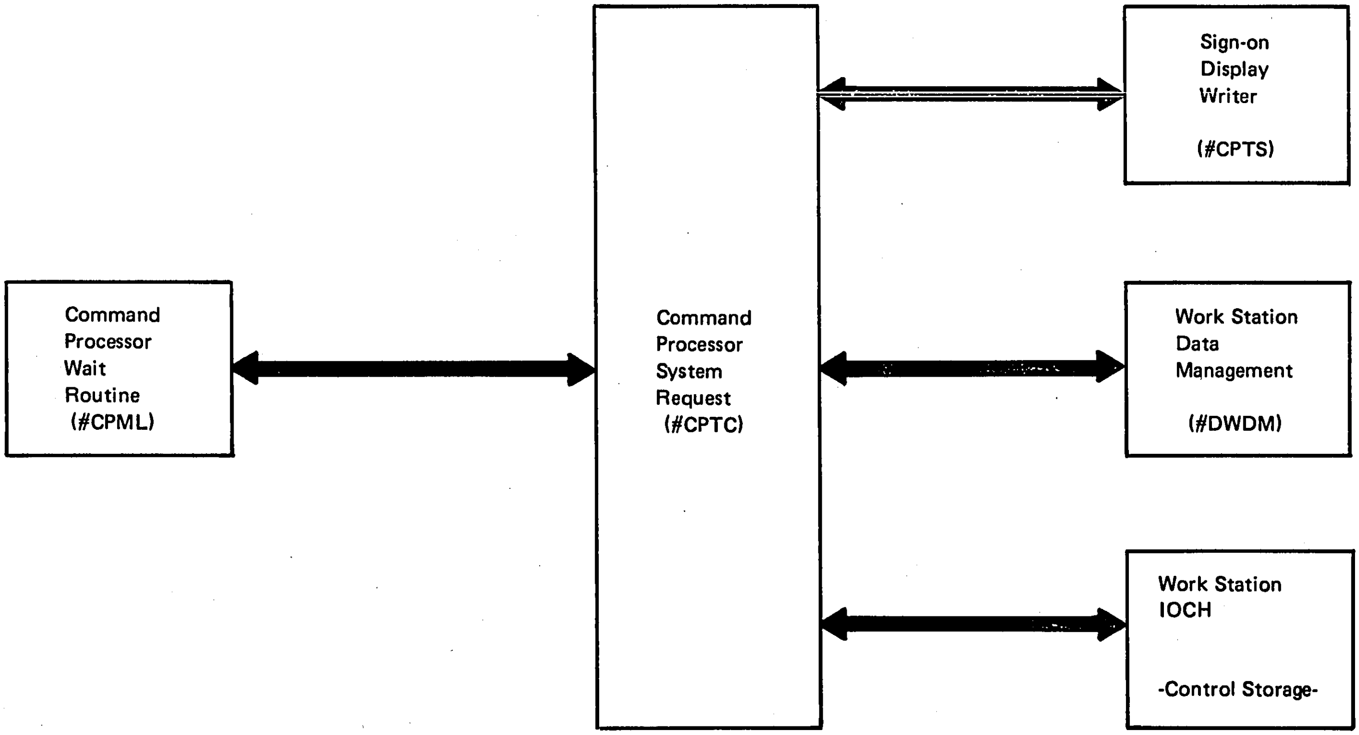


Figure 2-28. System Request Processor Control Flow

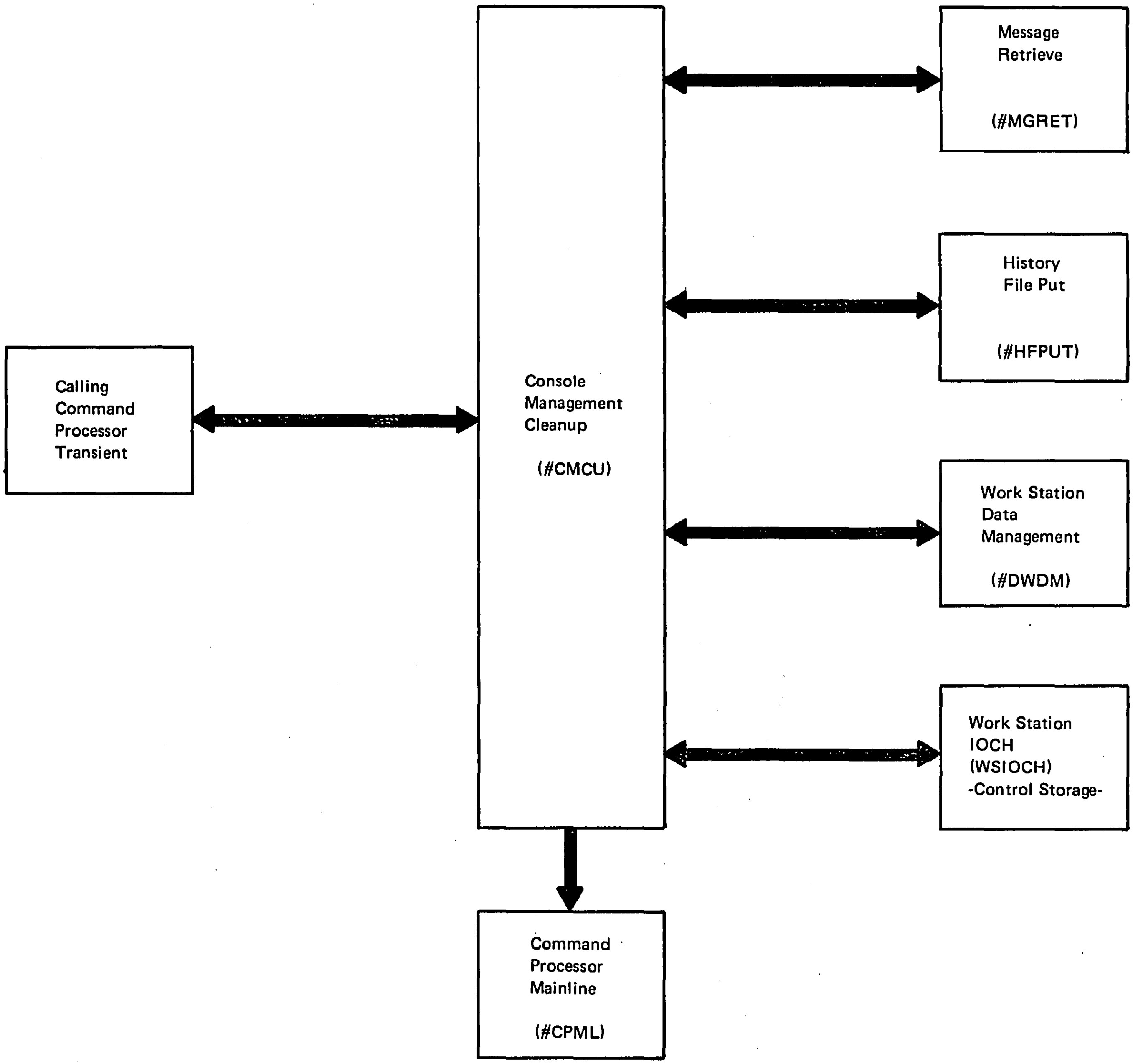


Figure 2-29. Cleanup Routine Control Flow

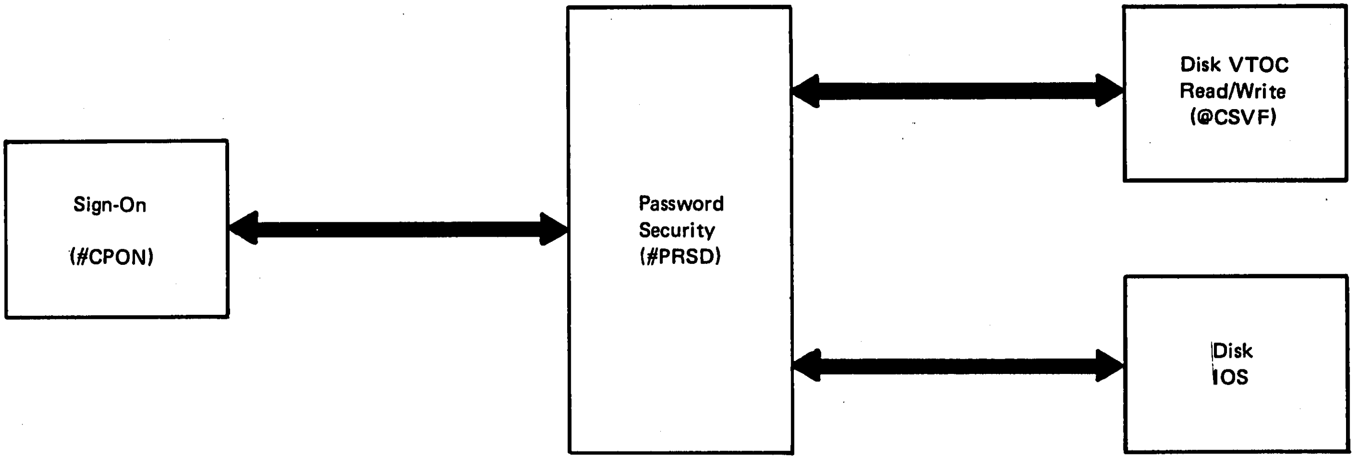


Figure 2-30. Password Security Control Flow

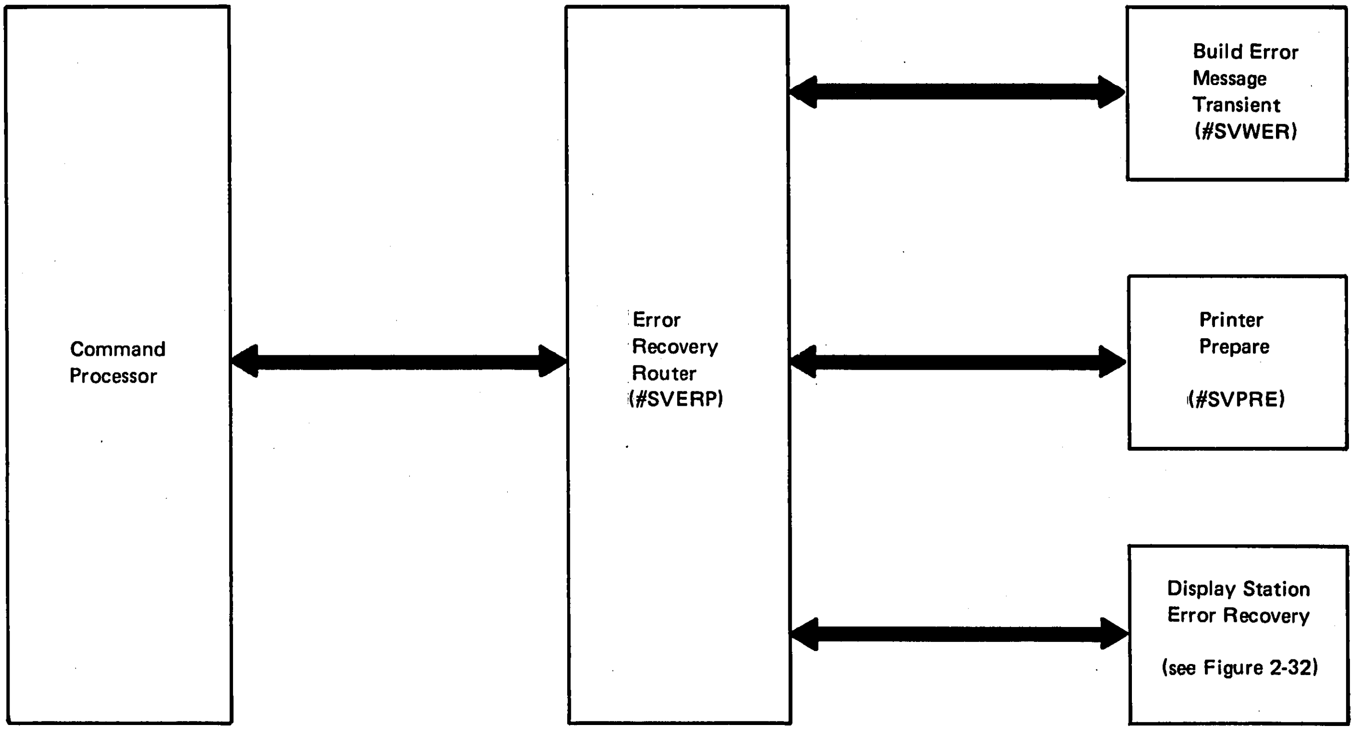
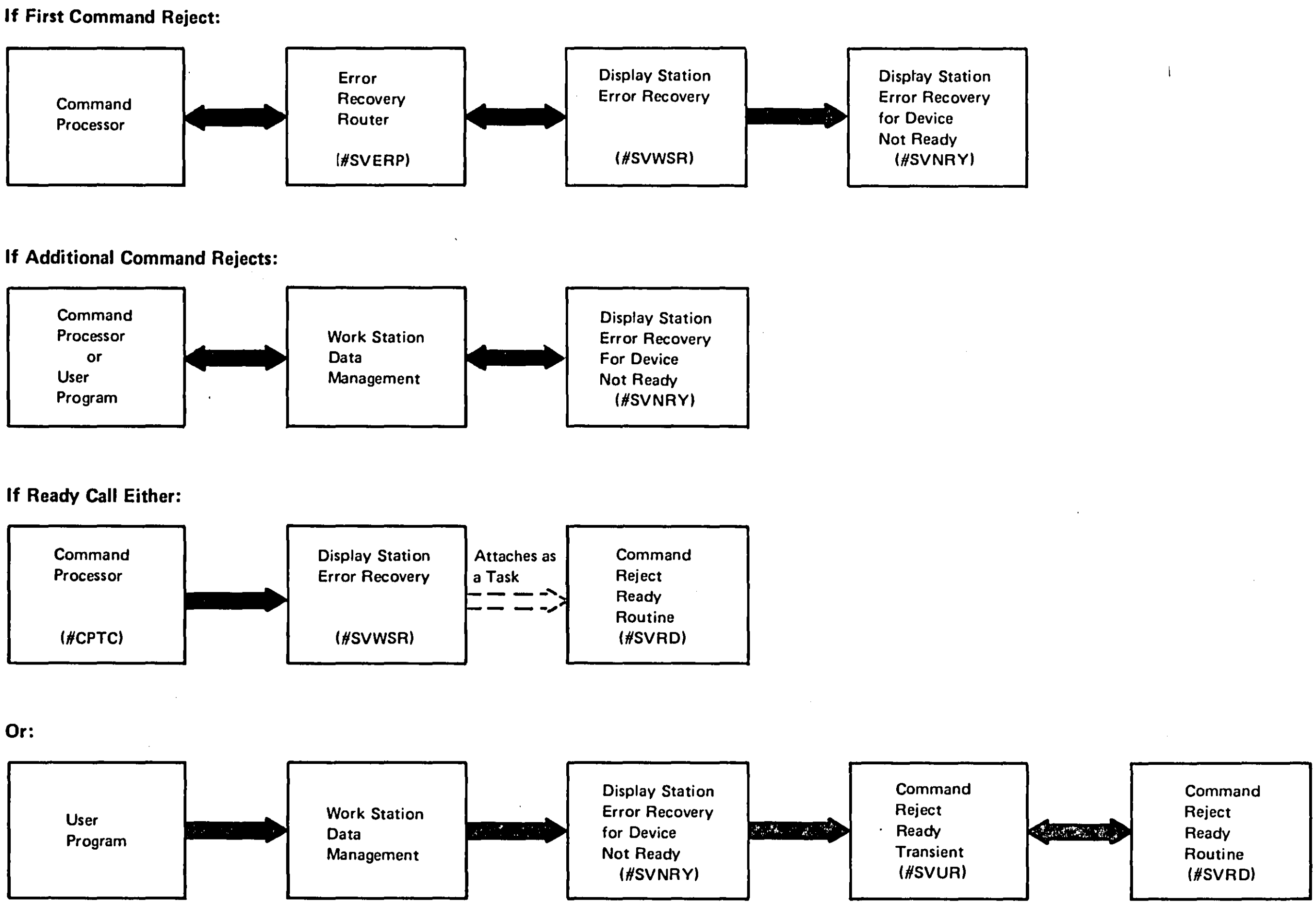


Figure 2-31. I/O Error Recovery Control Flow



If First Command Reject:

Command Processor
Error Recovery Router (#SVERP)
Display Station Error Recovery (#SVWSR)
Display Station Error Recovery for Device Not Ready (#SVNRY)

If Additional Command Rejects:

Command Processor or User Program
Work Station Data Management
Display Station Error Recovery For Device Not Ready (#SVNRY)

If Ready Call Either:

Command Processor (#CPTC)
Display Station Error Recovery (#SVWSR)
Attaches as a Task
Command Reject Ready Routine (#SVRD)

Or:

User Program
Work Station Data Management
Display Station Error Recovery for Device Not Ready (#SVNRY)
Command Reject Ready Transient (#SVUR)
Command Reject Ready Routine (#SVRD)

Figure 2-32. Display Station Error Recovery Control Flow

Introduction

The functions needed to start a job are:

- Initiator
  - Reader/interpreter
  - Disk file initialization
  - Program initialization

- Allocate
  - Normal allocate
  - Special allocate
  - Deallocate

- Open
  - Common-1
  - Disk
  - Work station
  - Diskette
  - Common-2
  - Printer
  - Data communications

INITIATOR

The initiator performs three functions:

- Reader/interpreter

- Disk file initialization

- Program initialization

The reader/interpreter function reads, diagnoses, and interprets operational control language (OCL) statements supported by the system. The reader/interpreter function uses the initiator mainline phase, OCL statement processors, and error handler phase. For improved performance and better diagnostics, the mainline module performs more than one function. The reader/interpreter function checks each statement for valid parameters and stores appropriate data for use by the system in processing jobs. If an invalid parameter is encountered, the initiator error handler routine is called to issue the proper message identification code (MIC).

The disk file initialization function prepares disk files for program processing. Information provided by the FILE OCL statements is used to build the disk file format 1's. When file initialization is complete, control is returned to the interpreter mainline to read and process the next OCL statement.

The program initialization function performs the steps required to load and pass control to the program specified on the LOAD OCL statement. The requested program may reside in the designated user library, and if used, this library is searched first for the program. If the requested program is not in the designated user library, the system library is used. If the program requires source (program products), the program initialization function allocates and opens the required work files. The program initialization function is completed by calling the supervisor attach transient which loads and passes control to the requested program.

Reader/Interpreter Function

The initiator mainline module (#CIML) is loaded by either the command processor (when starting a new job), step termination (when starting a new step within a job), or release (when returning to a procedure from a multiple requester terminal (MRT) program or a released program). It is loaded into the user area at main storage logical address X'C800'. The first sector of #CIML is the initiator work area (IWA). The mainline program follows the IWA and starts at main storage logical address X'C900'.

The initiator mainline (#CIML) calls SYSIN (#CLSS) to read a statement. #CIML then performs preliminary syntax checking of the statement and, if necessary, encodes the parameters.

If the statement format is: // VERB KEYWORD1-PARM1,KEYWORD2-PARM2,●●● #CIML encodes the statement in the initiator work area before calling the appropriate statement processor. The encoded statement format is:

A	B	C	D	E	F	●●●	G

where: A is control bit 1*
B is the length of parameter 1
C is parameter 1
D is control byte 2
E is the length of parameter 2
F is parameter 2
G is the end of encoding indicator (X'FF')
*Each keyword is assigned a unique control byte

Hexadecimal Control Byte	Keyword	Hexadecimal Control Byte	Keyword
01	NAME	10	USER1
02	LABEL	11	USER2
03	DISP	12	LINES
04	UNIT	13	FORMSNO
05	DATE	14	COPIES
06	RECORDS	15	DEFER
07	LOCATION	16	ALIGN
08	RETAIN	17	SPOOL
09	BLOCKS	18	PRIORITY
0A	PACK	19	DEVICE
0B	SOURCE	1A	MRTMAX
0C	INLIB	1B	SIZE
0D	OUTLIB	1C	NEP
0E	PROGRAM1	1D	OFFSET
0F	PROGRAM2	1E	DATA
		1F	SYMID
		20	REQD
		21	RESTORE
		22	LINE
		23	RELEASE
		24	PRINT
		29	MEMBER
		2A	FORMAT

In the above example, if KEYWORD1 has a control byte of X'01' and KEYWORD2 has a control byte of X'02', the encoded statement is:

01	05	PARM1	02	05	PARM2	●●●	FF

All syntax and punctuation checking is done by #CIML. The statement processor modules use the encoded statements in the indicator work area to check parameter validity and process the parameters.

Based on the verb found in the statement, the appropriate initiator routine (statement processor) is loaded at main storage logical address X'F800' and given control (see Figure 3-2 for initiator logic flow).

The initiator error handler (#CIER) is called to handle errors detected by #CIML or the statement processor modules. #CIER is loaded at main storage logical address X'F800'. It checks the error code placed in the initiator work area by the calling routine. #CIER matches the error code to a message identification code (MIC) and issues the proper message.

If no errors are detected by the statement processor, control returns to #CIML via a branch instruction to read the next statement.

The statement processors are not concerned with continuation statements. #CIML reads statements until all continuation statements are read and then calls the appropriate statement processor. For example, if the following OCL statements are entered:

(1) // LOAD #RPG

(2) // COMPILE SOURCE-PROG1,

(3) // INLIB-LIB1,

(4) // OUTLIB-LIB2

(5) // RUN

the LOAD statement processor (#CILD) is called after statement (1) is read. The COMPILE statement processor (#CICM), however, is not called until statements (2) through (4) are read by #CIML. The RUN statement processor (#CIRN) is called after statement (5) is read.

When a LOAD statement is received, a find is performed for the module in the library. If it is not found, an error message is issued, thus providing early diagnostics. On previous systems, the module was not found until the RUN statement was read. If the module is found, the directory entry is saved in the initiator work area for use by the RUN statement processor to load the requested module.

When a FILE statement is received, its syntax is completely checked. All file initialization (such as checking the disk VTOC) is performed at the time the FILE statement is received, thus providing early diagnostics. Taking a 1 option allows the user to reenter the statement if necessary.

When a RUN statement is received, #CIRN overlays the mainline module since the user's program is about to receive control and no more OCL statements need to be read. #CIRN processes the RUN statement, performs all program initialization (such as checking for source and setting up $WORK and $SOURCE files), and finally calls the supervisor attach transient (#SVAU) to load and pass control to the requested program.

System Interlocks

The System/34 multitask environment requires system interlocks to prevent the simultaneous use of certain system resources by two or more active tasks.

The interlocks are:

- Initiation/termination
- VTOC (includes main storage F1's)
- Format 5
- Program dedication interlock
- Procedure name
- History file
- Console SYSLOG

The initiator obtains these resources by setting the interlocks in the TCBs. For locations of the interlocks, see *Task Control Block* and *Queue Header Area* in the *Data Areas Handbook.*

Disk File Initialization Function

After the initiator mainline file routine (in #CIML) processes the file statement and builds the format 1, control is passed to the initiator mainline file initialization routine (also in #CIML).

The file initialization routine examines the format 1 built by the file statement processor and if the unit specified is diskette, file initialization merely moves the format 1 to the active format 1 area (AFA).

If the unit specified in the file statement is disk, the file initialization routine examines the file specification block (FSB) chain. If the file is already on the FSB chain, the format 1 for that file is used. If RETAIN-J was specified on the file statement, the file initialization routine either creates a new format 1 for the file or uses the RETAIN-J format 1 created by a previous step. If the file is being used by another task, and the file is sharable, the existing format 1 is used.

The initiator VTOC routine (#CIVT) is called if the file is not on the FSB chain, not retain J, and not being used by another task. #CIVT examines the disk VTOC trying to locate the file. If the file is found in the VTOC, the VTOC format 1 is added to the format 1 chain and a new FSB is added to the FSB chain. If the file is not found in the VTOC, an indicator in the previously built format 1 is set, indicating a new file, and this format 1 is added to the FSB chain.

When file initialization is complete, control is returned to the initiator mainline to read the next OCL statement.

Program Initialization Function

The RUN statement processor (#CIRN) is called when the initiator mainline encounters the RUN statement. #CIRN coordinates the program initialization function.

After placing the program date in all new disk format 1s, #CIRN examines the directory information placed in the initiator work area (IWA) by the LOAD statement processor. If the requested program requires source information, #CIRN calls the allocate function to allocate $SOURCE, $WORK, and $WORK2 files as required. After the files are allocated, #CIRN calls the disk open transient to open the files.

If a COMPILE statement was entered, #CIRN calls SYSIN to read source statements from the keyboard and then calls disk data management to place the statements in the $SOURCE file. A compiler information block (CIB) is then built in the assign/free area. File information such as begin extent, end of data, and device code is placed in the CIB for use by the compiler.

If the requested program is part of a procedure, the procedure parameter save area (PPSA) contains pointers needed to continue with the next step after the requested program finishes processing. For this reason, #CIRN writes the PPSA buffer in the PPSA on disk to save the information for future use.

All required disk files are enqueued and work stations acquired. The requester work station is released if an ATTR statement with RELEASE-YES was specified.

The last step performed by the program initialization function is to load and pass control to the supervisor attach transient (#SVAU). #SVAU ensures that there is enough main storage space available to prevent the requested program from being permanently swapped out when it is loaded. After ensuring that another task will not be permanently swapped out, #SVAU loads and passes control to the requested program.

ALLOCATE

Allocate consists of three functions:

- *Normal allocate* controls the assignment of disk space, devices, and disk files to the user program. Normal allocate is the only allocate function that can be called by a nonprivileged user.
- *Special allocate* allows a privileged user to allocate disk files without file OCL statements.
- *Deallocate* allows a privileged user to deallocate printer and disk files. The process is similar to file deallocation at step termination time. Deallocate also allows freeing part of a file's disk space.

Normal allocate, special allocate, and deallocate are main storage transients and each is called by an SVC command with a different request indicator byte (RIB).

Input to the allocate function is passed by way of preopen DTFs. See the *Data Areas Handbook* for a description of preopen DTFs. In addition to the normal DTF fields, special allocate and deallocate use additional fields in the preopen disk DTF. The special allocate fields overlay the normal disk DTF fields starting at label $F1NXR. The deallocate DTF is based on a closed disk DTF. A single request byte at label DTFSR2 (request byte 2) of the special allocate DTF specifies the desired deallocate operation. If part of a file's disk space is to be freed, the new end extent can be specified in the DTF.

Normal Allocate

When allocate is required, the user program passes control to the allocate mainline transient (#CAML).

#CAML processes each preopen DTF in the chain that is not conditioned off by UPSI. If a diskette file is to be allocated or if keysort is required, #CAML passes control to the allocate push/pull transient, #CAPS (see Figure 3-3 for normal allocate control flow).

#CAPS determines which allocate functions are required. If diskette allocate (#CAR1) or keysort interface (#CAKS) is required, #CAPS pushes a portion of the user area for use by #CAR1 or #CAKS. All normal allocate modules except #CAR1 and #CAKS are main storage transients.

#CAKS, keysort interface, runs in the first 2K of user storage. When keysort is required, allocate push/pull (#CAPS) pushes the user program to disk and loads #CAKS into the first 2K of a 14K region. #CAKS then loads the keysort program into the remaining 12K of user area.

#CAR1, the diskette allocate module, runs in the first 4K of user storage. When diskette files are allocated, allocate push/pull (#CAPS) pushes the user program to disk, sets up a 4K region, moves the diskette DTF to the beginning of the user area, and loads #CAR1. After diskette files are allocated or keysort runs, control returns to #CAPS. #CAPS pulls the user program back from disk to main storage and updates the necessary DTFs.

#CACM is called if data communications DTFs are to be allocated. #CAPT is called if printer DTFs are to be allocated. #CAF1 is called if new disk files are to be allocated.

#CAF2, the find disk space transient, is called by #CAF1 to find space in the format 5 area.

#CAF3, the load to old disk file allocate transient, is called by #CAF2. If load to old but not new disk file allocate is requested, #CAF3 is called by #CAML, #CACM, #CAPT, or #CAPS (see Figure 3-3).

Control is returned to the user program after all devices and files are allocated.

Special Allocate

A privileged user can allocate a disk file without supplying an OCL statement by passing control to special allocate (#CAS1). See Figure 3-4 for special allocate logic flow.

#CAS1 builds and queues a format 1 and a file specification block (FSB) for each disk file to be allocated. If any files were successfully processed to this point, special allocate – first pass (#CAS2) is called.

#CAS2 performs first pass processing of all preopen disk DTFs in the chain. If keysort is required, #CAS2 passes control to the allocate push/pull transient (#CAPS).

#CAPS performs the same function as for normal allocate. If new or load to old disk files require allocation, #CAPS passes control to new file allocate (#CAF1). If only load to old disk files need allocation, #CAPS passes control directly to load to old disk file allocate (#CAF3).

#CAF1 and #CAF3 may, if #CAPS was not used, be called directly by #CAS2.

Control is returned to the user program after requested disk files are allocated.

Deallocate

When a privileged user requests the deallocate function, control passes to deallocate premainline (#CAD1). (See Figure 3-5 for deallocate logic flow.)

#CAD1 performs first pass processing of all DTFs in the chain. If the deallocate request is for a nondisk device, #CAD1 deallocates the device and returns directly to the user program. If keysort is required, #CAD1 passes control to the allocate push/pull transient (#CAPS).

#CAPS perform the same function as for normal allocate. When used for deallocate, however, #CAPS calls deallocate mainline (#CAD2).

#CAD2 continues processing DTFs, freeing file space, and deallocating files as required. If format 5s are to be updated, #CAD2 passes control to the free format 5 space transient (#CAD3).

#CAD3 updates the format 5s as required and returns control to the user program. If #CAD3 is not required, #CAD2 returns control to the user program.

OPEN

The purpose of open is to prepare for the transfer of data to and from the program's files by:

- Initiating all necessary DTF fields in preparation for data transfer
- Assign buffers and IOBs for data transfer
- Prepare the device for I/O operations to a new data file

The open function is performed by two common open transients (#DMOP and #DMOF) and the necessary device oriented open transients:

Disk Open	#DD1OP and #DD2OP
Work Station Open	#WDOPN
Diskette Open	#DROP
BSC Open	#BSOB

All open modules are transients and reside as load modules in the system library. The open function is initiated by issuing an SVC with the open RIB (X'02') and index register 2 pointing to the first DTF on the DTF chain. The SVC processor loads the first common open module (#DMOP).

To improve system performance, the device oriented modules are designed to be called only once for each DTF chain. Thus, each module performs its functions for all required DTFs on the chain. It searches through the DTF chain to find all DTFs of its device type that are to be opened.

Note: See the *Data Areas Handbook* for a description of disk, diskette, and printer preopen and postopen DTFs.

Common Open 1 (#DMOP)

The first common transient (#DMOP) searches the DTF chain and flags DTFs to be opened. For open to be performed, a DTF must have a valid device type, it must be allocated, the UPSI setting must be correct, and it must be in a preopen state. #DMOP will then call one of the following open transients:

- #DD1OP if a disk DTF is on the DTF chain
- #WDOPN if a work station DTF is on the DTF chain
- #DMOF if neither a disk DTF nor a work station DTF is on the chain

Input to #DMOP is a chain of preopen DTFs addressed by XR2. These DTFs were created and chained when the program was compiled or assembled. XR2 points to the first DTF on the chain. This DTF contains a pointer ($DFCHB) to the second DTF, the second DTF to the third, and so on (see Figure 3-1).

Output from #DMOP is a chain of DTFs with an indicator in each showing which DTF to open.

Disk Open (#DD1OP, #DD2OP)

The device oriented function for disk consists of two transients – #DD1OP and #DD2OP. Input to each is the address of the first DTF on the forward chain in register 2. Each of these modules must loop through the entire DTF chain and process all disk DTFs which need to be opened. Control is passed to #WDOPN if a work station DTF is present on the DTF chain or to #DMOF if not.

The first module of disk open (#DD1OP) performs the diagnostic checking to ensure access to the requested file is allowed under the specified access type. If an error is found, the proper MIC number is established and the SYSLOG routine called to display the error message. In addition, this module initiates the DTF for all access methods other than normal disk processing. For dummy open DTFs, it provides information about the file within the DTF. For ZPAM and ISRI, it initiates the DTF and loads the necessary data management within the user-provided area.

The second module (#DD2OP) formats the necessary control blocks (DTF, IOBs, master index, high key buckets) and the buffer areas. The IOBs and buffer areas are first formatted based upon the access being consecutive/direct/indexed and normal/SIAM. Then the remaining DTF fields are calculated to allow data management to begin processing. Finally, if opening an indexed file, the master index area is initialized and the high key buckets are established, if requested.

Work Station Open (#WDOPN)

The work station open routine is called by common open (#DMOP) or disk open (#DD2OP). Open passes the address or the first DTF on the chain in XR2. The DTF may be chained to other DTFs that are not work station DTFs.

#WDOPN's primary function is to place the format indexes the user requires into the user area for use and reference by the work station data management at execution time.

Chained DTFs may be used to open one or multiple format load members. If only one format load member is to be opened, only one DTF in the chain may contain the name of the format load member; the balance of the DTFs must have blanks in the $WSFMTN field. If multiple format load members are being opened, the $WSINXA field must contain the same address. This is true if the DTFs are chained or multiple open calls are performed to open the multiple format load members.

#WDOPN places format indexes after the indexes previously placed in the user-supplied area. #WDOPN also checks previously opened format indexes to ensure duplicate format names have not been opened. If a duplicate is found, a halt is issued. If the format load member is not found or the load member found is not a format load member, a halt is also issued.

Diskette File Open (#DROP)

When a diskette file DTF is encountered by a call to the open routine, the diskette open transient module is called. Open reads the active format 1 image corresponding to the file to be processed. Open then formats the DTF to its postopen status, formats the IOB and prepares the input/output buffers for processing the file. Information from the preopen DTF is saved in the active format 1 image so the DTF can be restored to its preopen state by Close.

The postopen DTF serves as the interface between diskette data management and the calling program, the IOB furnishes the interface between data management and diskette IOS that performs the physical disk seek, read and write operations.

The calling program must supply a main storage area for diskette data management and put the address of the start of this area into the preopen DTF. Open then loads diskette data management at this location.

For output files, the open routine calls diskette IOS to write an internal control record into the first data sector of the file, if necessary. For input files containing the internal control record, the internal control record is read and placed at the beginning of the input buffer.

For existing files, the open routine performs diagnostics to ensure the file organization and access type are compatible. The record length specified in the DTF is checked to ensure it is the same as that from the data set label. However, if the calling program places X'FFFF' in the preopen DTF record length field, the open routine will place the data set label record length into the postopen DTF.

Common Open 2 (#DMOF)

The second common transient (#DMOF) is called after the DTFs to be opened are flagged by #DMOP and after all disk and work station DTFs are open. #DMOF first calls the proper device oriented open transients to open all remaining DTFs. #DMOF then creates a backward chain of DTFs just opened.

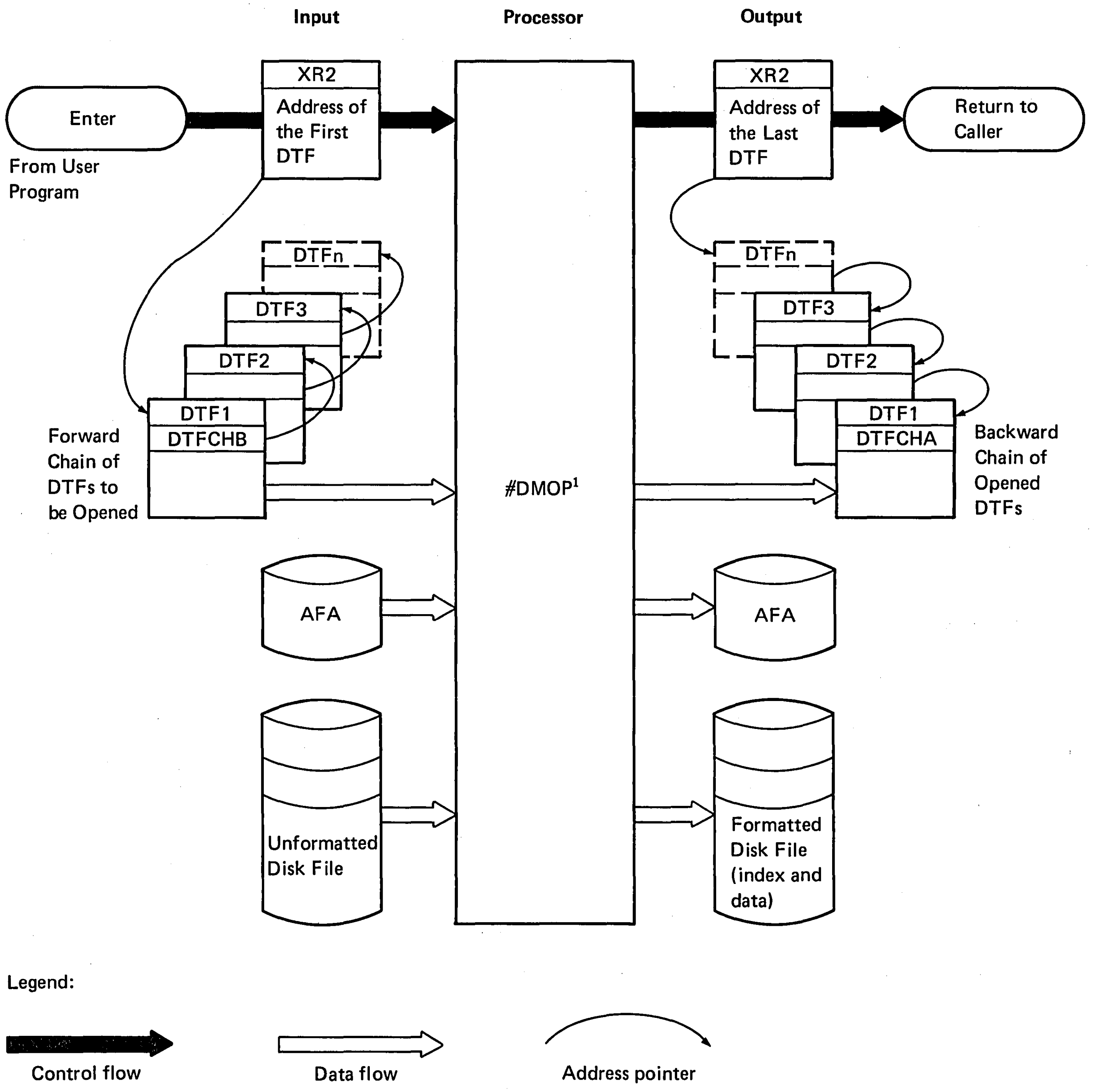
Input to #DMOF is a chain of DTFs with some open and others not open. Output is a chain of postopen DTFs with the last DTF containing a pointer ($DFCHA) to the next to last DTF opened. In this manner, the DTFs are chained back to the first DTF opened (see Figure 3-1).

#DMOF also contains all logic needed to open a print file.

Printer Open (within #DMOF)

The printer open logic is called for any print file to be opened. The print file may be printed directly to a 5211 or a 5256 printer, or, if spooling is active, it may be spooled for later printing. Input to printer open is XR2 pointing to the printer DTF to be opened.

It first performs all diagnostic checking on the file, and if an error is found, it issues the proper MIC through the use of SYSLOG. If no errors are found, space within the assign/free area is obtained for the printer IOB and the IOB is initialized. Then the remaining fields within the DTF are initialized to allow data transfer. Finally, a format command and a forms feed command are issued to the printer. This establishes the proper printer control for the program and positions the carriage on line one of a new page.



[1] Figure 3-6 shows the interrelationship of the transient open modules.

Figure 3-1. Open Main Control and Data Flow

Method of Operation

This section contains functional diagrams for routines needed to start a job. They are:

- Initiator
- Normal allocate
- Special allocate
- Deallocate
- Open disk, diskette, printer, work station, and data communications DTFs

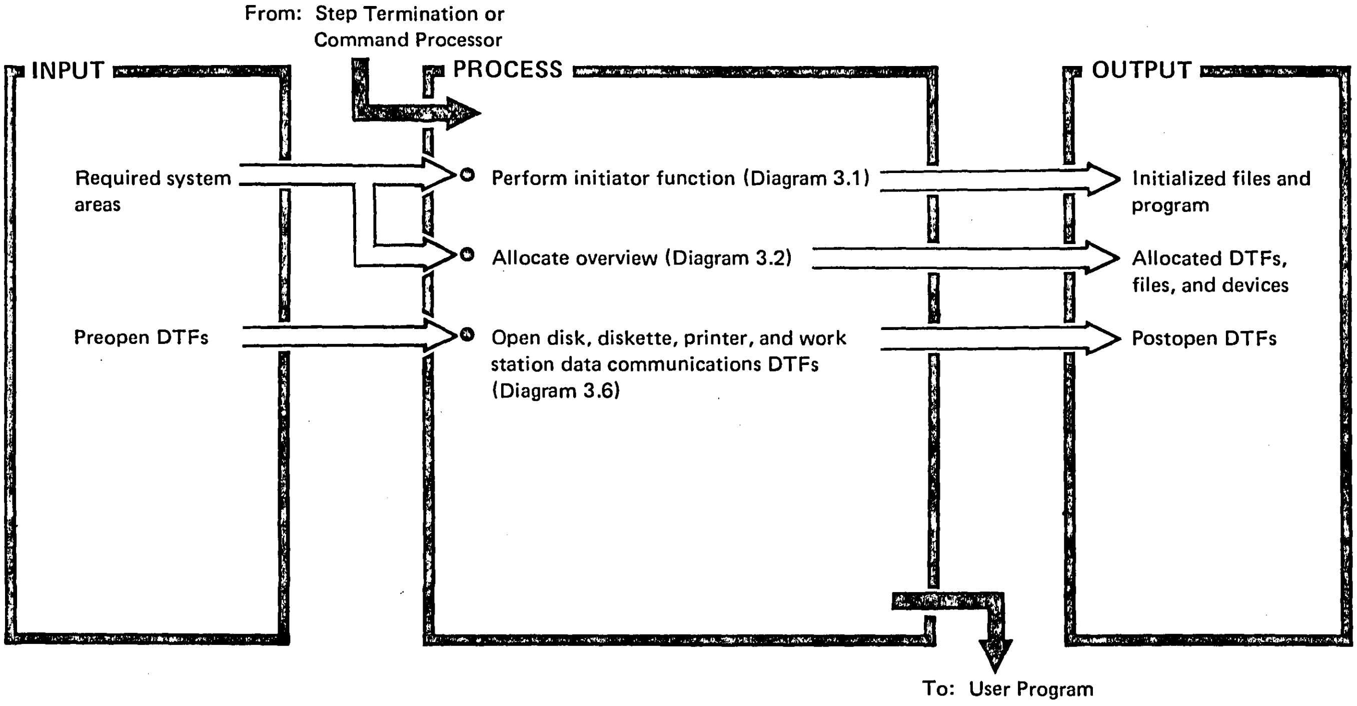
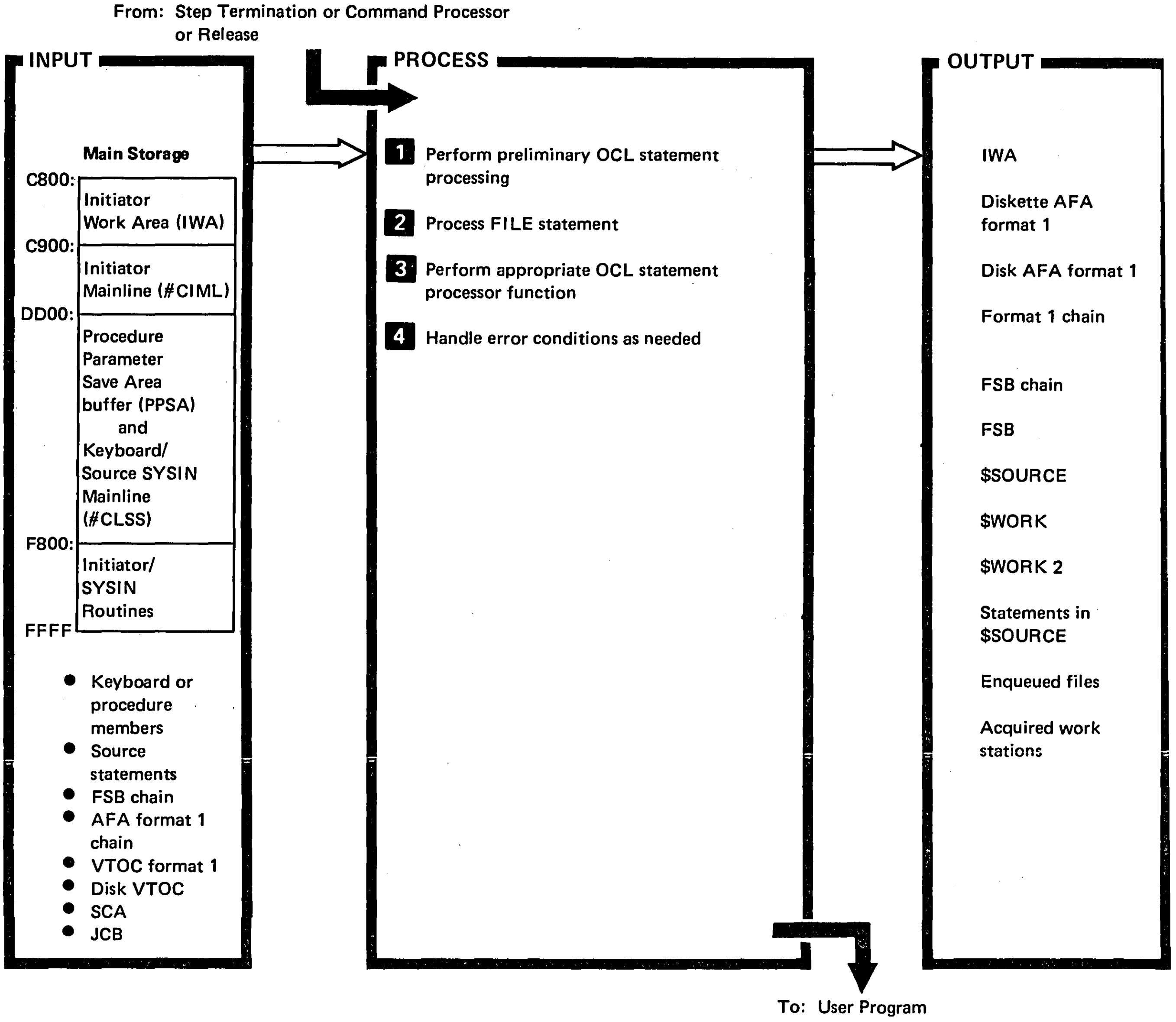

From: Step Termination or Command Processor
INPUT
PROCESS
OUTPUT
Required system areas
Preopen DTFs
Perform initiator function (Diagram 3.1)
Allocate overview (Diagram 3.2)
Open disk, diskette, printer, and work station data communications DTFs (Diagram 3.6)
Initialized files and program
Allocated DTFs, files, and devices
Postopen DTFs
To: User Program

Diagram 3.0. Overview of Starting a Job



From: Step Termination or Command Processor
or Release

INPUT

Main Storage

C800:	Initiator Work Area (IWA)
C900:	Initiator Mainline (#CIML)
DD00:	Procedure Parameter Save Area buffer (PPSA) and Keyboard/ Source SYSIN Mainline (#CLSS)
F800:	Initiator/ SYSIN Routines
FFFF	

- Keyboard or procedure members
- Source statements
- FSB chain
- AFA format 1 chain
- VTOC format 1
- Disk VTOC
- SCA
- JCB

PROCESS

1 Perform preliminary OCL statement processing

2 Process FILE statement

3 Perform appropriate OCL statement processor function

4 Handle error conditions as needed

OUTPUT

IWA

Diskette AFA format 1

Disk AFA format 1

Format 1 chain

FSB chain

FSB

$SOURCE

$WORK

$WORK 2

Statements in $SOURCE

Enqueued files

Acquired work stations

To: User Program

DESCRIPTION	MODULE/ ROUTINE
1 Initialize fields to begin processing next statement.	#CIML
Read record from keyboard or procedure member and place in buffer.	#CLSS
If statement begins with *, read next statement.	#CIML
If statement does not begin with // it is assumed to be procedure call; go to 1 Ⓐ.	
Scan statement for verb.	
Search verb table for verb.	
If verb found in table, go to 1 Ⓑ.	

Diagram 3.1 (Part 1 of 11). Perform Initator Function

DESCRIPTION	MODULE/ ROUTINE
Ⓐ Set up fetch parameter list to give control to Include Statement Processor (#CIIC) and go to 1 Ⓓ.	#CIML
Ⓑ If encoding required switch in verb table is not on, go to 1 Ⓒ. Find keywords in keyword table and place their control bytes in encoding area. Move parameter and length to encoding area.	
Ⓒ If verb is FILE, go to 2. If any errors detected, set error code in IWA and go to 4.	
Ⓓ Fetch appropriate statement processor (relocating loader SVC 52) and go to 3.	
2 Check switch in IWA to ensure FILE statement follows LOAD statement. Scan encoded parameters in IWA and go to appropriate processing routine (within #CIML): ● Name and label parameter. ● Date parameter. ● Pack parameter. ● Retain parameter. ● Unit parameter. ● Records parameter. ● Blocks parameter. ● Location parameter. ● Disp parameter. If no label parameter specified, substitute name parameter for label and place in format 1 area. Ensure: ● Name parameter specified. ● No diskette parameters on disk file statement. ● No disk parameters on diskette file statement. ● No DISP parameter specified with RETAIN-J. Ensure duplicate name not in file specification block (FSB) chain. Search FSB chain for file with same unit, label, and date (if specified). If requested file already on FSB chain: ● Point new FSB at format 1. ● Add new FSB to chain. ● Return to 1 to read next OCL statement. If unit specified on file statement is diskette: ● Add new FSB to chain. ● Move format 1 to AFA format 1 area. ● Return to 1 to read next OCL statement. If file not on FSB chain and RETAIN-J specified on file statement: ● Create new disk AFA format 1. ● Add FSB to chain. ● Return to 1 to read next OCL statement. If file being used by another task and can be shared: ● Add new FSB to chain (FSB points to found format 1). ● Return to 1 to read next OCL statement. If file not on FSB chain, not RETAIN-J, and not being used by another task: ● Call initiator VTOC routine.	

Diagram 3.1 (Part 2 of 11). Perform Initator Function

DESCRIPTION	MODULE/ ROUTINE
A ● Look for file in disk VTOC. ● If file not found in disk VTOC, go to 2 B.	#CIVT
● Compare format 1 built from file statement with VTOC format 1 just read.	#CIML
● If not right one, go to 2 A to see if another file exists in VTOC with same label.	#CIVT
● Add VTOC format 1 to format 1 chain. ● Add new FSB to FSB chain.	#CIML
B ● Indicate in format 1 that file is new (file not on FSB chain, AFA format 1 chain, or in VTOC).	#CIVT
● Add format 1 built from file statement to format 1 chain. ● Add new FSB to FSB chain. If any errors detected, set error code in IWA and go to 4. Return to 1 to read next OCL statement.	#CIML
3 ** or ** Statement Processor* Scan SYSIN buffer for first nonblank character after * or ** and if character not quotation mark ('message'), go to 3 A. Scan SYSIN buffer for second quotation mark (end of message). Place start of message and message length in SYSLOG parameter list.	#CIMS or #CIM2
Call SYSLOG to display message to work station if //* or system console if //** and go to 3 B.	#CLXS
A Convert message identification code (MIC) from EBCDIC characters to packed decimal and place in message retrieve parameter list.	#CIMS or #CIM2
Call SYSLOG to display message to work station if //* or system console if //**.	#CLXS
B If any errors detected, set error code in IWA and go to 4. Return to 1 or termination to return to command mode.	#CIMS or #CIM2
ATTR Statement Processor Check that ATTR statement is not between LOAD statement and RUN statement. If system is in single program mode, return to 1. Scan encoded parameters in IWA and go to appropriate routine (in #CIAT): ● Priority parameter. ● MRTMAX parameter. ● NEP parameter. ● Release parameter. Set priority and MRTMAX in TCB. Set NEP in JCB. Set release indicator in IWA. If any errors detected, set error code in IWA and go to 4. Return to 1 to read next statement.	#CIAT

Diagram 3.1 (Part 3 of 11). Perform Initator Function

DESCRIPTION	MODULE/ ROUTINE
COMPILE Statement Processor Check that COMPILE statement follows LOAD statement. Check directory entry in IWA to see if source required.	#CICM
If no source required, go to 4.	#CIER
Check for second COMPILE statement in this step. Scan encoded parameters in IWA and go to appropriate routine (in #CICM): ● SOURCE parameter. ● INLIB parameter (find library on disk). ● OUTLIB parameter (find library on disk). ● MRTMAX parameter (save in compiler information block). ● NEP parameter (save in CIB). Find source member and save information about it in IWA. Move compiler information block (CIB) to assign/free area and save CIB address in JCB. If any errors detected, set error code in IWA and go to 4. Return to 1 to read next statement.	#CICM
DATE Statement Processor If not beginning of session, go to 3 C (beginning of session is time between sign-on and first load statement received). If session data already specified (in JCB), go to 3 D. C Check that DATE statement follows LOAD statement. Check for second date statement in this step. D Scan input buffer for date, remove delimiters and add zeros if required. Put date in year-month-day and packed format. Put date in JCB. If any errors detected, set error code in IWA and go to 4. Return to 1 to read next statement or call termination to return to command mode.	#CIDT
FORMS Statement Processor Scan encoded parameters in IWA and go to appropriate routine (in #CIFM): ● LINES parameter. ● FORMSNO parameter. Set lines/page in JCB if specified. Set forms number in JCB if specified. If any errors detected, set error code in IWA and go to 4. Return to 1 to read next statement or call termination to return to command mode.	#CIFM

Diagram 3.1 (Part 4 of 11). Perform Initator Function

DESCRIPTION	MODULE/ ROUTINE
IMAGE Statement Processor	#CIIM
Call TWA (SVC 51) access to read work station configuration record.	
If first parameter is HEX or CHAR, go to 3 E.	
Find source member name on statement.	
Search system library for source member.	#MASYG
Read first statement.	
Ensure first statement: ● Begins with //. ● Verb is IMAGE. ● First parameter is HEX or CHAR.	#CIIM
E Convert length in source get/SYSIN buffer to binary.	
Read print belt image into configuration record buffer.	#CLSS or #MASYG
Write updated work station configuration record from config record buffer to configuration record on disk.	
If any errors detected, set error code in IWA and go to 4.	#CIIM
Return to 1 to read next statement or termination to return to command code.	
INCLUDE Statement Processor	#CIIC
If next level of procedure parameter save area (PPSA) not in main storage, use task work area routine to retrieve it.	
Find procedure name on statement.	
Find procedure in user library or system library.	#MASFN
Save start sector, end sector, and record length of procedure in procedure parameter save area buffer.	#CIIC
If this is a MRT, go to 3 F.	
Save parameters in PPSA buffer.	
If not MRT, save parameters in PPSA buffer.	
Save information from job control block (JCB) in PPSA: ● Message member addresses. ● Current library. ● UPSI switches.	
Use task work area routine to write PPSA buffer to PPSA (to inform that, If active function, this procedure is now active).	
Update current PPSA tag in JCB.	
If any errors detected, set error code in IWA and go to 4.	
Return to 1 to read next statement.	

Diagram 3.1 (Part 5 of 11). Perform Initator Function

DESCRIPTION	MODULE/ ROUTINE
MRT Procedure Processor Ⓕ Check that MRT procedure call is not between LOAD and RUN statements. Use task work area (TWA) (SVC 51) access routine to write data (if any) from OCL statement to TWA. Search TCB chain to see if MRT is already active. If MRT is found, go to 3 Ⓖ.	#CIMT
Create new JCB using system configuration record information.	Disk IOS
Go to 3 Ⓗ. Ⓖ Check if MRT is waiting for resources and if so, set waiting indicator in JCB. Build ACE and queue it to work station IOCH queue or TCB complete queue. Post work station controller or post the MRT. Ⓗ If any errors detected, set error code in IWA and go to 4. Return to 1 if initiating new MRT, go to detach routine #SVAU if attached to existing MRT.	#CIMT
LIBRARY Statement Processor Check that LIBRARY statement is not between LOAD and RUN statements. Scan encoded parameters in IWA encoding area and go to name parameter routine.	#CILB
Find specified user library and put library format 1 address in parameter list.	#MAFLB
Find file specification block (FSB) for this library in library FSB chain. Set on library statement bit in FSB. If in a procedure, go to 3 Ⓙ. Set on session library bit in FSB. Ⓙ Set library format 1 address in JCB. If any errors detected, set error code in IWA and go to 4. Return to 1 or termination to return to command mode.	#CILB
LOAD Statement Processor Check for second LOAD statement in this step. Syntax check program name.	#CILD
Search current user library (if any) and system library for specified program.	#MASFN
Save directory information in IWA. Ensure program will fit in user area: ● Convert the program size to 2K blocks and store it in the IWA. ● Compare the program size to the user area size in the SCA.	#CILD

Diagram 3.1 (Part 6 of 11). Perform Initator Function

DESCRIPTION	MODULE/ ROUTINE
Set appropriate indicators in IWA and JCB: ● Program has utility control statements (JCB). ● NEP program (JCB). ● Load statement received this session (JCB). ● Load statement received this job (JCB). ● Load statement received this step (IWA). ● Initiator in intra mode (JCB). ● Statement processors should not call termination (IWA).	#CILD
If this is a MRT program, set up indicators to initiate the MRT.	#CILM
Ⓚ If any errors detected, set error code in IWA and go to 4. Return to 1 to process next statement.	#CILD
LOAD – MRT Routine Check that MRT program is not being loaded from keyboard. Check that MRT program is not being loaded from the job queue. If not MRT procedure set MRTMAX in TCB to zero, indicate program is SRT, and go to 3 Ⓚ. Set MRTMAX in TCB to value in directory information (IWA) – MRTMAX was X'FF' when procedure first entered. If number of requestors not greater than MRTMAX, go to 3 Ⓚ. Calculate number requesters over MRTMAX. Find same number (over MRTMAX) TUBs on chain attached to this MRT and set off no skip bit (indicates requester not active but waiting to attach to MRT). If errors detected, set error code in IWA and go to 4. Go to 1 to process next statement.	#CILM
LOCAL Statement Processor Scan encoded parameters in IWA and go to appropriate routine (in #CILC): ● OFFSET parameter. ● DATA parameter. Use task work area (TWA) access routine to read in local area from TWA. Move data at specified offset into local area buffer and use TWA access to write buffer to disk. If errors detected, set error code in IWA and go to 4. Go to 1 to process next statement or call termination to return to command mode.	#CILC
LOG Statement Processor If not in single program mode, and if spool is supported, return to 1 or to termination to return to command mode. Ensure that first parameter is CRT or PRINTER. Ensure second parameter (if any) is EJECT or NOEJECT.	#CILG

Diagram 3.1 (Part 7 of 11). Perform Initator Function

DESCRIPTION	MODULE/ ROUTINE
Set SYSLOG indicator in SCA.	#CILG
Set eject/noeject indicator in SCA (default is eject).	
If any errors detected, set error indicator in IWA and go to 4.	
Go to 1 or termination to return to command mode.	
MEMBER Statement Processor	#CIMM
Scan encoded statement in IWA and go to appropriate routine (in #CIMM): ● Program 1 parameter. ● Program 2 parameter. ● User 1 parameter. ● User 2 parameter.	
Search current user library and then system library for requested member.	#MASFN
Set relative disk address for message member and library format 1 address in JCB.	#CIMM
If any errors detected, set error code in IWA and go to 4.	
Return to 1 or termination if return to command mode.	
PAUSE Statement Processor	#CIPS
Call SYSLOG to issue PAUSE message.	#CLXS
Return to 1 to process next statement.	#CIPS
PRINTER Statement Processor	#CIPR
Ensure PRINTER statement follows LOAD statement.	
Set up default printer specification block (PSB).	
Scan encoded parameters in IWA and go to appropriate routine (in #CIPR): ● LINES parameter. ● FORMSNO parameter. ● COPIES parameter. ● DEFER parameter. ● ALIGN parameter. ● SPOOL parameter. ● PRIORITY parameter ● DEVICE parameter. ● NAME parameter.	
Check PSB to see if name specified.	
Search PSB chain for duplicate name.	
If device not specified, set default in PSB.	
Add new PSB to chain.	
If errors detected, set error code in IWA and go to 1.	
Return to 1 to process next statement.	

Diagram 3.1 (Part 8 of 11). Perform Initator Function

DESCRIPTION	MODULE/ ROUTINE
REGION Statement Processor	#CIRG
Check that REGION statement is not between LOAD and RUN statement.	
Scan encoded parameters in IWA and go to size parameter routine.	
Set region size in JCB.	
If any errors detected, set error code in IWA and go to 4.	
Return to 1 to process next statement.	
RUN Statement Processor	#CIRN
Ensure that LOAD statement preceded RUN statement.	
Get program date from job control block (JCB) and place in all new disk F1's.	
Check F1's and file specification blocks (FSBs) to ensure no other task is creating a file with the same label and date.	
Bind resources needed by the program: ● Enqueue all disk files.	
● Acquire all required work stations.	#DWDM
If requested program is part of a procedure, call task work area (TWA) access routine (see *Control Storage Logic Manual*) to write procedure parameter save area (PPSA) buffer to PPSA sector in TWA on disk.	#CIRN
Check directory information in initiator work area (IWA) to determine if program requires source; if no go to 3 L.	
Check if file statements provided for $SOURCE, $WORK, and $WORK2 files:	
● Allocate files for which file statements are provided.	#CAML
● Return.	#CIRN
● Allocate files for which file statements are not provided.	#CAS1
● Return.	#CIRN
Open $SOURCE, $WORK, and $WORK2 files.	#DMOP
Return.	#CIRN
If compile statement received:	
● Read source statements from source member.	#SYSG
● Return.	#CIRN
● Write source statements to $SOURCE file.	#DDDM
● Return.	#CIRN
If compile statement not received:	
● Read source statements from keyboard.	#CLSS
● Return.	#CIRN
● Write source statements to $SOURCE file.	#DDDM
● Return.	#CIRN
Close $SOURCE, $WORK, and $WORK2 files.	#DDCL
L If the requested program must be loaded from the system console, ensure that the requester is the system console.	#CIRN
Check for an active dedicated program; if one is found, issue an error message.	
If this is a dedicated program, ensure that no other tasks are active; if other tasks are active, issue an error message.	

Diagram 3.1 (Part 9 of 11). Perform Initator Function

DESCRIPTION	MODULE/ ROUTINE
If this is a high level dedicated program, ensure that the requester is the only signed on command display station; if another command display station is signed on, issue an error message.	#CIRN
If RELEASE-YES was requested, then release the requester display station.	#DWDM
If any errors detected, set error code in IWA and go to 4.	#CIRN
Load the requested program into user area of main storage and pass control to it.	#SVAU
SWITCH Statement Processor If SWITCH statement follows LOAD statement, check if SWITCH statement previously specified for step. Scan SWITCH parameter in SYSIN buffer for proper characters (ones, zeros, and Xs). Set on or off appropriate UPSI switches in JCB. If any errors detected, set on error code in IWA and go to 4. Return to 1 or termination to return to command mode.	#CISW
SYSLIST Statement Processor If parameter is not CRT or OFF go to 3 M. If CRT specified, set indicator in JCB to X'EEEE'. If OFF specified, set indicator in JCB to X'0000'. Go to 3 P. M If parameter is work station printer ID, go to 3 N. If parameter is PRINTER, get printer ID from requester TUB. (If MRT or job queue printer is system printer.) N Find TUB containing printer ID and ensure it is printer TUB. Set printer ID in JCB. P If any errors detected, set error code in IWA and go to 4. Return to 1 or termination to return to command mode.	#CISL
WORKSTN Statement Processor Check that WORKSTN statement follows LOAD statement. Scan encoded parameters in IWA and go to appropriate routine (in #CIWK): ● REQD parameter. ● RESTORE parameter. ● UNIT parameter. ● SYMID parameter. ● PRINT parameter. Check work station block (WSB) to ensure unit specified. Find TUB with specified work station ID and place address in WSB.	#CIWK

Diagram 3.1 (Part 10 of 11). Perform Initator Function

DESCRIPTION	MODULE/ ROUTINE
Check WSB chain for duplicate SYMID or unit. If print parameter not specified, set default address in WSB: ● If MRT or job queue, use system printer. ● If not MRT or job queue, use configured printer. Add WSB to chain. If any errors detected, set error code in IWA and go to 4. Return to 1 to process next statement.	#CIWK
4 Search error table for requested error code (in IWA) and place message identification code (MIC) and options in SYSLOG parameter list. If error code not found, issue disaster error (MIC 300). If name field should be displayed, move name to SYSLOG parameter list and go to 4 A. If only error should be displayed, go to 4 A. If not in procedure, go to 4 A. Set off retry option (option 1) in SYSLOG parameter list (not allowed in procedure). If in batch job, go to 4 A.	#CIER
Display heading and statement in error. A Display error message.	#CLXS
If 0 or 1 option taken, go to 1 to read next statement. If 2 option taken, call termination.	#CIER

Diagram 3.1 (Part 11 of 11). Perform Initator Function

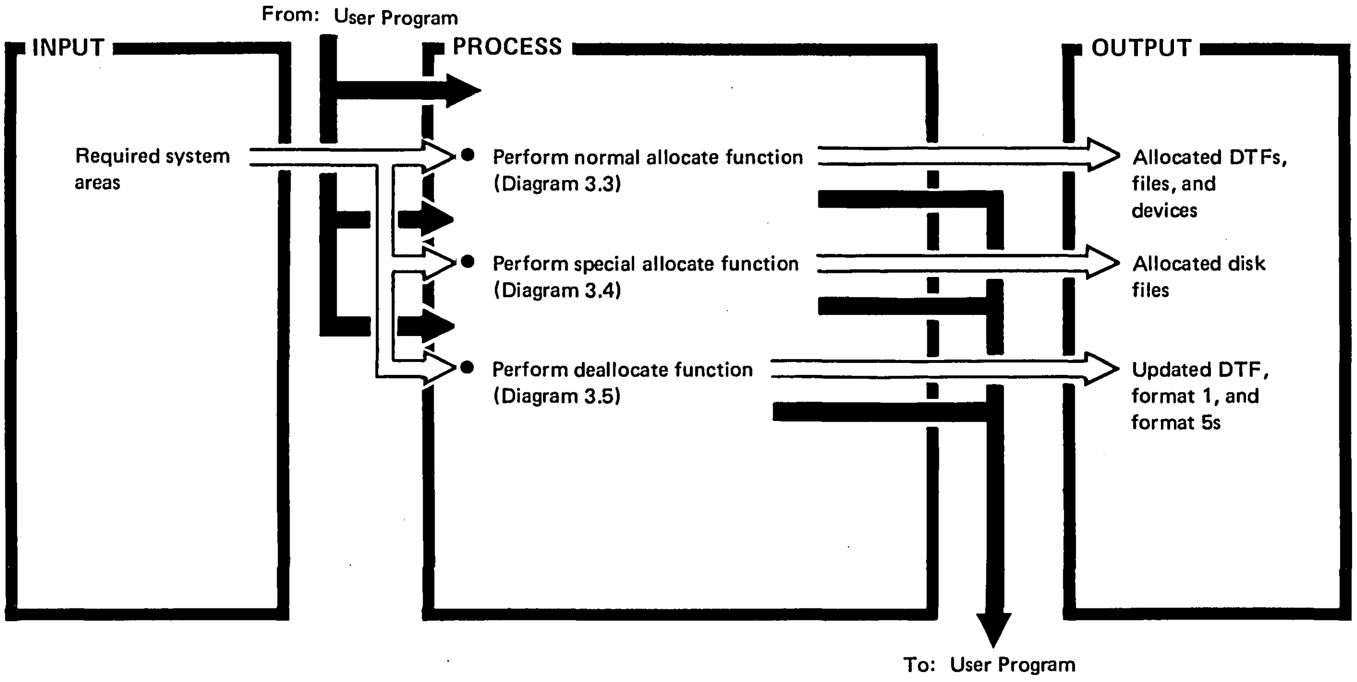
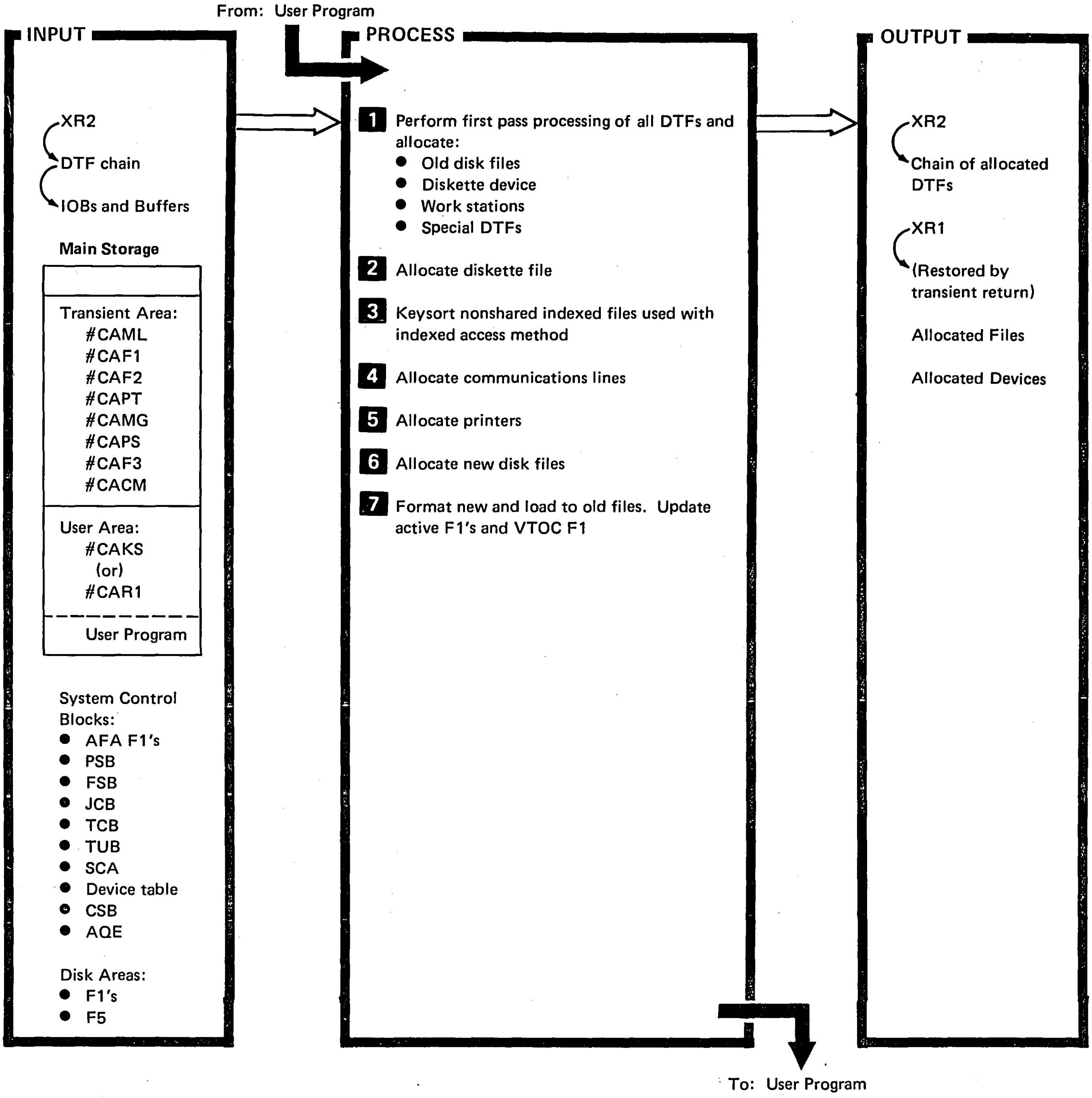

From: User Program
INPUT
PROCESS
OUTPUT
Required system areas
Perform normal allocate function (Diagram 3.3)
Perform special allocate function (Diagram 3.4)
Perform deallocate function (Diagram 3.5)
Allocated DTFs, files, and devices
Allocated disk files
Updated DTF, format 1, and format 5s
To: User Program

Diagram 3.2. Overview of Allocate



From: User Program

INPUT

XR2

DTF chain

IOBs and Buffers

Main Storage

Transient Area:
#CAML
#CAF1
#CAF2
#CAPT
#CAMG
#CAPS
#CAF3
#CACM

User Area:
#CAKS
(or)
#CAR1

User Program

System Control Blocks:
- AFA F1's
- PSB
- FSB
- JCB
- TCB
- TUB
- SCA
- Device table
- CSB
- AQE

Disk Areas:
- F1's
- F5

PROCESS

1 Perform first pass processing of all DTFs and allocate:
- Old disk files
- Diskette device
- Work stations
- Special DTFs

2 Allocate diskette file

3 Keysort nonshared indexed files used with indexed access method

4 Allocate communications lines

5 Allocate printers

6 Allocate new disk files

7 Format new and load to old files. Update active F1's and VTOC F1

OUTPUT

XR2

Chain of allocated DTFs

XR1

(Restored by transient return)

Allocated Files

Allocated Devices

To: User Program

DESCRIPTION	MODULE/ ROUTINE
1 Process each preopen DTF in chain not conditioned off by UPSI. If disk DTF: ● If old disk file: – Complete AFA format 1. – Assign high key bucket if indexed file.	#CAML

Diagram 3.3 (Part 1 of 4). Perform Normal Allocate Function

DESCRIPTION	MODULE/ ROUTINE
– If load to old, set load to old to do bit in request block (RB) and set new creation date. – If not load to old, indicate file allocated in DTF and FSB. – Indicate in FSB and RB if keysort required. ● If new disk file: – Indicate new file to allocate in RB. – Turn on new file switch in DTF. – Update AFA format 1. – Assign high key bucket if required. If diskette DTF: ● Indicate diskette in use in JCB. ● Allocate device. ● Set on allocate request bit in RB. If work station or special DTF, set on allocated bit in DTF. Return to 1 and repeat steps until all DTFs processed. If diskette allocate or keysort requested in RB, go to A. If communications lines, call communications allocate (#CACM) and go to 4. If printers to be allocated, call print allocate (#CAPT) and go to 5. If new disk files to allocate, call new disk file allocate (#CAF1) and go to 6. If load to old files to allocate, call load to old disk file allocate (#CAF3) and go to 7. If no allocate requests, return to user program. A Call allocate push/pull transient (#CAPS) and pass control to it.	#CAML
If diskette file allocate request bit on in RB: ● Push user program to disk and set up 4K region in user area. ● Move DTF to start of user area. ● Load diskette allocate (#CAR1) after DTF in user area and go to 2. B If return from #CAR1, restore user region and pull user program from disk. If keysort request bit on in RB: ● Push user program to disk and set up 14K region in user area. ● Load keysort interface (#CAKS) into first 2K of user area. ● Go to 3. C If return from #CAKS, restore user region and pull user program from disk. If request for communications lines, load #CACM and go to 4. If request to allocate printers, load printer allocate transient (#CAPT) and go to 5. If requested to allocate new disk files, load #CAF1 and go to 6. If requested to allocate load-to-old disk files, load #CAF3 and go to 7. Return to user program.	#CAPS
2 If first time allocate or requested in DTF, perform diskette prepare function.	#CSVI
Find AFA format 1.	#CAR1

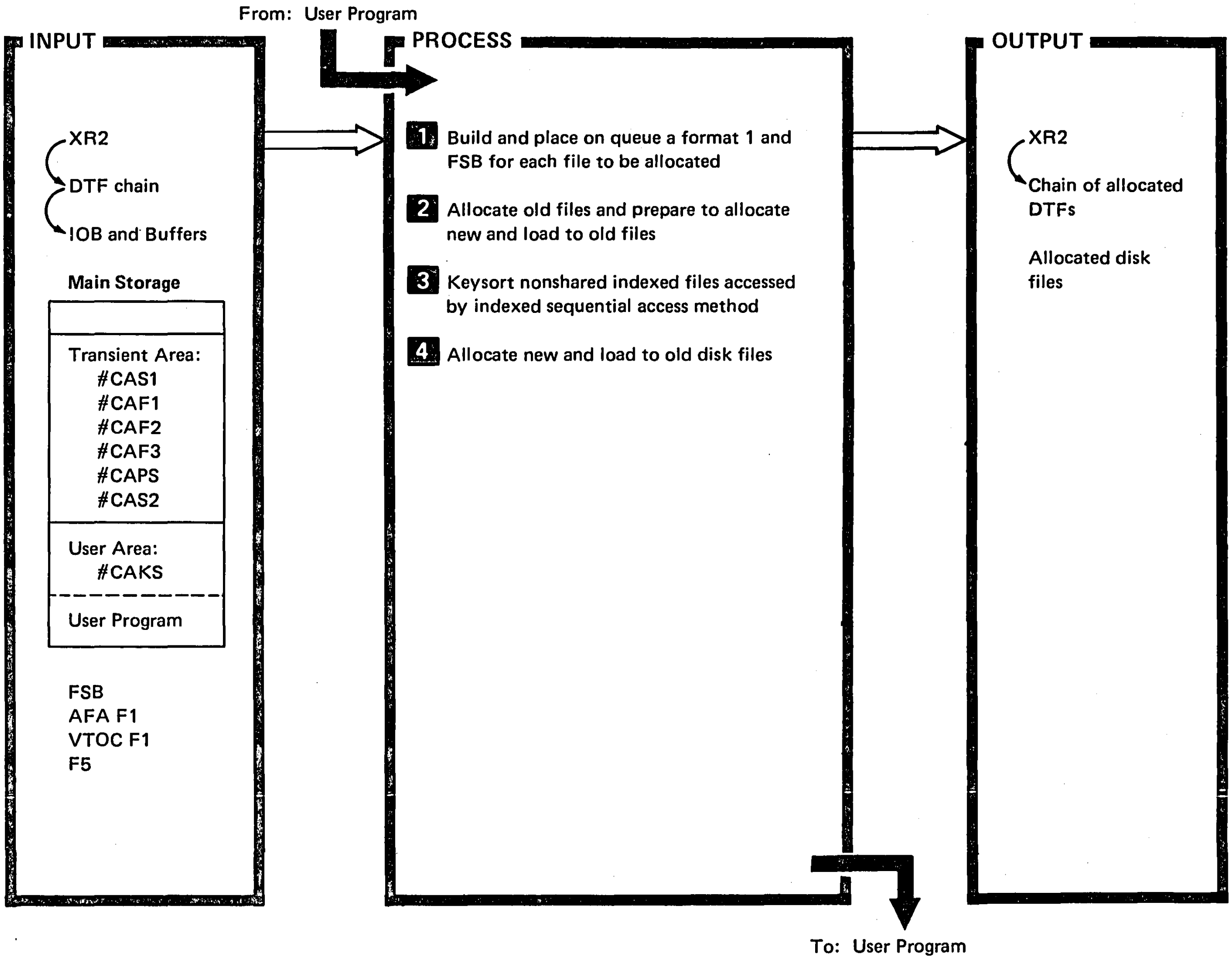
Diagram 3.3 (Part 2 of 4). Perform Normal Allocate Function

DESCRIPTION	MODULE/ ROUTINE
Check pack parameter against volume ID of mounted diskette. Determine if specified file exists. If allocating new diskette file: ● Delete all expired files. ● Ensure clean pack if requested in DTF. ● Allocate new file following existing files. ● Set creation date to proper date. ● Set expiration date. If allocating existing diskette file: ● For Add files: — Ensure last (or only) diskette volume of file inserted. — Ensure last active file on diskette is add file. — Ensure expired files (other than add file) are deleted. ● For input files, ensure first volume of multi-volume file inserted. ● For existing file with RETAIN specified, set new expiration date. Return to 1 B.	#CAR1
3 Check file specification block (FSB) for files requiring keysort. Fir files requiring keysort:	#CAKS
● Update VTOC F1 to allow for keysort failures.	#CSVF
● Load and pass control to keysort Modules. (See *Keysort*.)	#CSDK
● Return here after keysort complete.	#CAKS
▶ ● If duplicate keys returned from keysort, issue halt.	#CSDK ◀
● If keysort functioned satisfactorily, reupdate VTOC F1 to turn off failure indication.	#CSVF
● Return to 1 C.	#CAKS
4 If BSC or SDLC: ● Allocate line. ● Load data management task. ● Update DTF, CSB, and JCB. ● Set allocated bit in DTF. ● If printers to be allocated, call print allocate (#CAPT) and go to 5. ● If new disk files to allocate, call new disk file allocate (#CAF1) and go to 6. ● If load to old files to allocate, call load to old disk file allocate (#CAF3) and go to 7. ● If no allocate requests, return to user program. Search printer specification block (PSB) chain for matching DTF name. If match found, locate TUB with same work station ID as in PSB. If match not found, build default PSB. If output spooled: ● Indicate spool intercepting in PSB, DTF, and TUB. ● Set allocated bit in PSB and DTF. If output not spooled: ● Put device code in DTF. ● Set allocated bit in PSB and DTF. ● Save TUB and PSB address in DTF. ● Set check forms/image in PSB. ● Set call forms/image transient in request block (RB).	#CACM

Diagram 3.3 (Part 3 of 4). Perform Normal Allocate Function

DESCRIPTION	MODULE/ ROUTINE
If needed, do forms/image processing.	#CSIM
If request to allocate new disk files, load #CAF1 and go to 5. If request to allocate load to old disk files, load #CAF3 and go to 6.	#CAPT
6 For each new disk DTF: ○ Set up DTF to indicate: – Minimum and maximum sectors needed. – Spindle desired. – Location desired. ○ Count VTOC F1's needed.	#CAF1
Read format 5.	Disk IOS
Check if enough VTOC F1 space, and call find disk space transient module (#CAF2):	#CAF1
D ○ If requested space is available:	#CAF2
– Update format 5 to indicate space taken.	Disk IOS
– Place return code in DTF. – Pass control to #CAF3. ○ If requested space is not available: – Place return code in DTF to indicate no space available. – Set wait or halt bit in DTF. – Pass control to #CAF3.	#CAF2
7 If #CAF2 found no space: ○ Halt if dedicated system or program. ○ Wait for space in format 5, reread format 5, call #CAF2, and go to D. If #CAF2 found space:	#CAF3
○ Format file: – Write X'FF's to index if indexed file. – Write X'40's to data area if direct file or X'00' if not direct P or T file.	Disk IOS
○ Save start and end of data and start and end of index in DTF.	#CAF3
○ If P or T file, write format 1 to VTOC.	Disk IOS
○ Set allocated bit in DTF and FSB. Return to user program.	#CAF3
Note: Allocate messages, except duplicate key and diskette message, are provided by #CAMG, which interfaces to SYSLOG (duplicate key messages issued by #CSDK and diskette messages issued by #CAR1). If allocate message or halt: ○ Build SYSLOG parameter list with program data: – File name. – File label. – Forms number. – Work station ID or communication line number as required. ○ Call SYSLOG to issue message/halt.	#CAMG
● Call end of job transient when 2 option taken to halt.	#CLXS
Return to calling module.	#CAMG

Diagram 3.3 (Part 4 of 4). Perform Normal Allocate Function

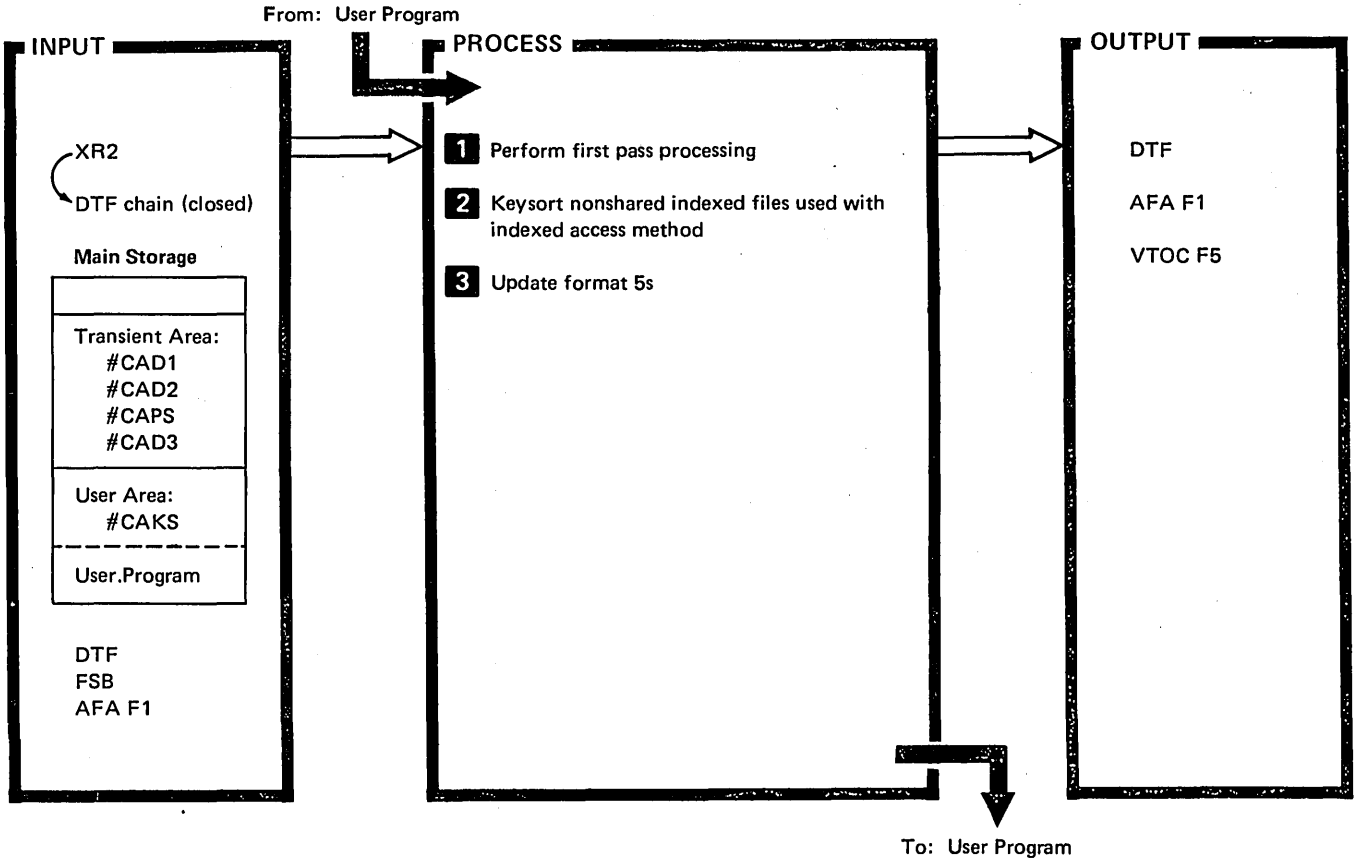

From: User Program
INPUT
XR2
DTF chain
IOB and Buffers
Main Storage
Transient Area:
#CAS1
#CAF1
#CAF2
#CAF3
#CAPS
#CAS2
User Area:
#CAKS
User Program
FSB
AFA F1
VTOC F1
F5
PROCESS
1 Build and place on queue a format 1 and FSB for each file to be allocated
2 Allocate old files and prepare to allocate new and load to old files
3 Keysort nonshared indexed files accessed by indexed sequential access method
4 Allocate new and load to old disk files
OUTPUT
XR2
Chain of allocated DTFs
Allocated disk files
To: User Program

DESCRIPTION	MODULE/ ROUTINE
1 Process each preopen disk DTF requiring special allocate. Build format 1 and FSB: ● If file specification block (FSB) exists (file statement specified) build format 1 into FSB format 1. ● If no FSB for requested file, assign area and build format 1 from DTF information. ● Create or update FSB. ● Create or update AFA format 1.	#CAS1
● If P or T file not in AFA or not on FSB chain, search VTOC for file by label (and date if specified).	#CSVF
● Return. 2 If any file successful, call #CAS2 to do the following for each successful DTF:	#CAS1
● If disk DTF: – Complete AFA format 1. – Assign high key bucket if indexed file. – If load to old, set load to old to do bit in request block (RB) and set new creation date. – If not load to old, indicate file allocated in DTF and FSB. – Indicate in FSB and RB if keysort required.	#CAS2

Diagram 3.4 (Part 1 of 2). Perform Special Allocate Function

DESCRIPTION	MODULE/ ROUTINE
● If new disk file: – Indicate new file to allocate in RB. – Turn on new file switch in DTF. – Update AFA format 1. – Assign high key bucket if required.	#CAS2
3 If keysort is needed, call allocate push/pull transient (#CAPS) and pass control to it.	
From this point, special allocate functions same as normal allocate.	
Use Diagram 3.3, enter at 1 B and continue to end.	
4 Special allocate functions same as normal allocate for new or load to old disk file allocate.	
Use Diagram 3.3, enter at 6 and continue to end.	

Diagram 3.4 (Part 2 of 2). Perform Special Allocate Function

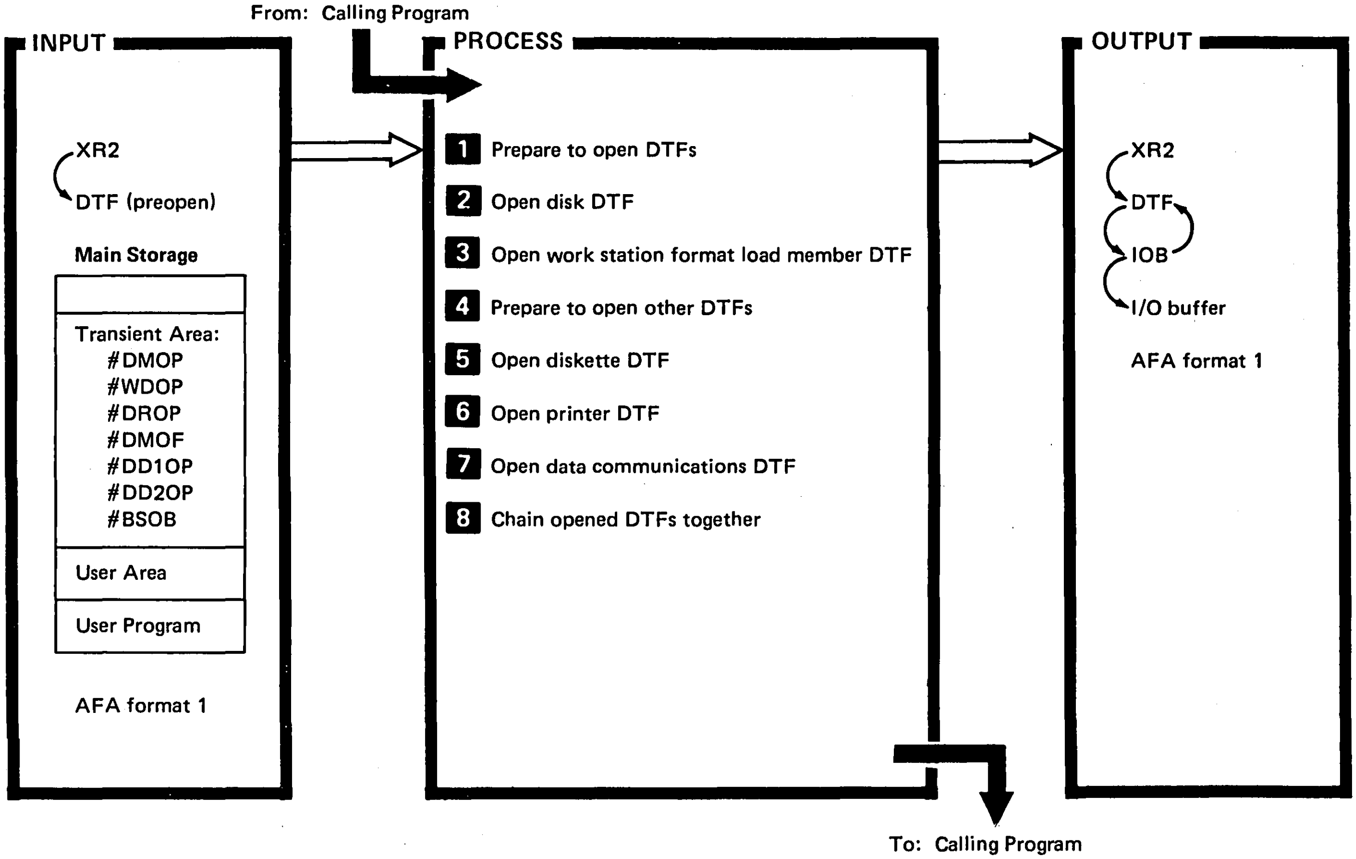


DESCRIPTION	MODULE/ ROUTINE
1 Check DTF device code. If not disk, deallocate device and return to user program. Check each DTF for keysort request: ● Search file specification block (FSB) for name match. ● If keysort requested: – Indicate keysort request in FSB and request block (RB). – Go to 2.	#CAD1
A Assign VTOC format 1 buffer. Set format 1 interlock. If free disk space request: ● Ensure nonshared file. ● Ensure valid end of file extent. ● If ignore format 5 processing bit not on in DTF, set on bit in RB to handle F5's. If file delete request, indicate in DTF to delete as scratch file. If file deallocate request: ● Update last date indicator for new files or old S files.	#CAD2
● Write active format 1 to VTOC for P and T files.	#CSVF

Diagram 3.5 (Part 1 of 2). Perform Deallocate Function

DESCRIPTION	MODULE/ ROUTINE
If S file and ignore format 5 processing bit not on in DTF, set on bit in RB to handle F5s.	#CAD2
Note: Only deallocate function allowed for shared files.	
If handle F5s bit on in RB, load #CAD3 and go to **3**.	
Return to user program.	
2 If keysort request bit on in RB: ● Call allocate push/pull transient (#CAPS) and pass control to it.	#CAD1
● Push user program to disk and set up 14K region in user area. ● Load keysort interface (#CAKS) into first 2K of user area.	#CAPS
● From this point, keysort functions same as for normal allocate.	#CAKS
Use Diagram 3.3 and enter at **3**.	
At end of keysort operation, return to this diagram at **1** Ⓐ.	
3 Enqueue format 5 interlock at 3 level.	#CAD3
Read VTOC F5 from disk and write to buffer.	Disk IOS
Process each DTF, determining start and end extents to be freed.	#CAD3
Add or merge format 5 entries in buffer as required.	
Issue message if: . ● Part of area already free. ● Format 5 too fragmented to use.	
Write updated buffer out to VTOC F5.	Disk IOS
Return to user program.	#CAD3

Diagram 3.5 (Part 2 of 2). Perform Deallocate Function



DESCRIPTION	MODULE/ ROUTINE
1 Check DTF for valid device type.	#DMOP
If device invalid, call SYSLOG to issue error message.	
Mark all DTFs to be opened by setting switch in DTF.	
If not end of DTF chain: ● Point to next DTF. ● Return to **1**.	
A Point to first DTF on chain. ● If disk DTF, indicate disk open required. ● If work station DTF, indicate work station open needed.	
Point to next DTF and repeat above, until end of chain is reached.	
Determine next transient to call: ● If disk open required, go to **2**. ● If work station open needed, go to **3**. ● Otherwise, go to **4**.	

Diagram 3.6 (Part 1 of 3). Open Disk, Diskette, Printer, Work Station and Data Communications DTFs

DESCRIPTION	MODULE/ ROUTINE
2 Perform all diagnostic checking of DTF. If error found, call SYSLOG to issue error message. If dummy open, initiate all required DTF fields. For ZPAM and ISRI, complete DTF and load required data management into user area if required. (Relocating loader SVC 52).	#DD1OP
Apportion IOBs and buffer area as needed. Complete all DTF fields as required. If requested, build master track index.	#DD2OP
Build high key bucket if needed.	Disk IOS
If work station open needed, go to **3**. Otherwise go to **4**.	#DD2OP
3 Check DTF for valid device code.	#WDOPN
If user library assigned, scan library directory for format load member name specified in DTF. If no library assigned, or if load member not in library, scan system library for format load member.	Disk IOS
If format load member found: ● Ensure load member created by $SFGR and total sectors greater than zero.	#WDOPN
● Read format load member index.	Disk IOS
● Count number of format index entries. ● If format index entry count exceeds maximum defined in DTF, call SYSLOG to issue message. ● If previous format load member opened, ensure no duplicate format names exist. ● If no errors found: – Move format indexes to location specified in DTF. – Update index address and number of index entries in job control block (JCB). When all work station DTFs are open, go to **4** to open next DTF or chain together opened DTFs.	#WDOPN
4 Point to first DTF on chain: **A** ● If printer DTF, go to **6**. ● If diskette DTF, go to **5**. ● If data communications DTF, go to **7**. ● If special DTF, indicate DTF opened. **B** Point to next DTF on chain: ● If end of chain reached, go to **8**. ● If not end of chain, go to **4** **A**.	#DMOF
5 Update active format 1. Ensure file properly allocated. Initialize DTF to postopen status. For existing files: ● Ensure file organization and access method compatible. ● Supply, or check record length.	#DROP

Diagram 3.6 (Part 2 of 3). Open Disk, Diskette, Printer, Work Station and Data Communications DTFs

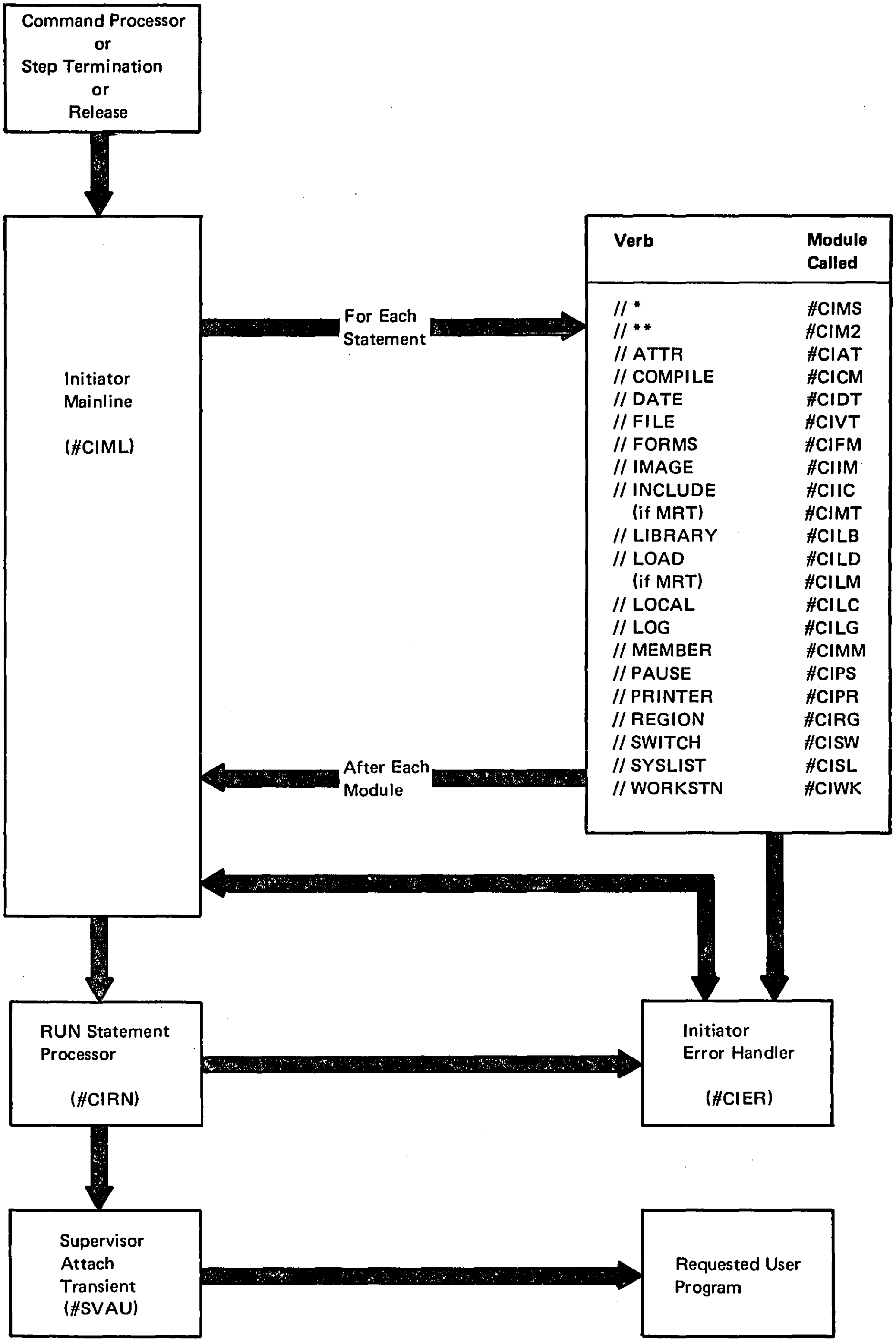
DESCRIPTION	MODULE/ ROUTINE
Build and initialize IOBs: ○ For output files, write internal control record if required.	#DROP
○ For input files, read first data area and set internal control record and/or track alignment indicators in DTF.	Diskette IOS
○ For basic exchange files, determine if full track I/O possible. Initialize DTF end of file field. Update DTF next record pointers and/or prime buffer for add file. Format IOB to postopen status for processing data: ○ For full track I/O requests, adjust to read or write to track boundary. ○ For basic exchange output file, clear output buffer. Process diagnostic errors. Load either diskette data management or sector data management to diskette (SVC 52) into user area if required. Go to 4 B to open next DTF or to chain together opened DTFs.	#DROP
6 Perform all diagnostic checking. If error found, call SYSLOG to issue error message. Obtain space within assign/free for IOB. Initialize IOB. Initialize all required DTF fields. Issue format and skip to line one command to printer.	#DMOF
Route printer IOBs to work station I/O control handler (WSIOCH) or to spool intercept routine (#SPINT).	WSIOCH
Go to 4 B to open next DTF or to chain together opened DTFs.	#DMOF
7 Open data communications DTF (Diagram 1.1 in *SSP Logic: Data Communications*). Go to 4 B to open next DTF or to chain together opened DTFs.	#BSOB
8 Build backward chain to include all opened DTFs. Return to calling program.	#DMOF

Diagram 3.6 (Part 3 of 3). Open Disk, Diskette, Printer, Work Station and Data Communications DTFs

Program Organization

Figures 3-2 through 3-6 show the logic flow of functions needed to start a job. They are:

- Initiator
- Normal allocate
- Special allocate
- Deallocate
- Open


Command Processor or Step Termination or Release
Initiator Mainline (#CIML)
For Each Statement
After Each Module
RUN Statement Processor (#CIRN)
Initiator Error Handler (#CIER)
Supervisor Attach Transient (#SVAU)
Requested User Program

Verb	Module Called
// *	#CIMS
// **	#CIM2
// ATTR	#CIAT
// COMPILE	#CICM
// DATE	#CIDT
// FILE	#CIVT
// FORMS	#CIFM
// IMAGE	#CIIM
// INCLUDE	#CIIC
(if MRT)	#CIMT
// LIBRARY	#CILB
// LOAD	#CILD
(if MRT)	#CILM
// LOCAL	#CILC
// LOG	#CILG
// MEMBER	#CIMM
// PAUSE	#CIPS
// PRINTER	#CIPR
// REGION	#CIRG
// SWITCH	#CISW
// SYSLIST	#CISL
// WORKSTN	#CIWK

Figure 3-2. Initiator Control Flow

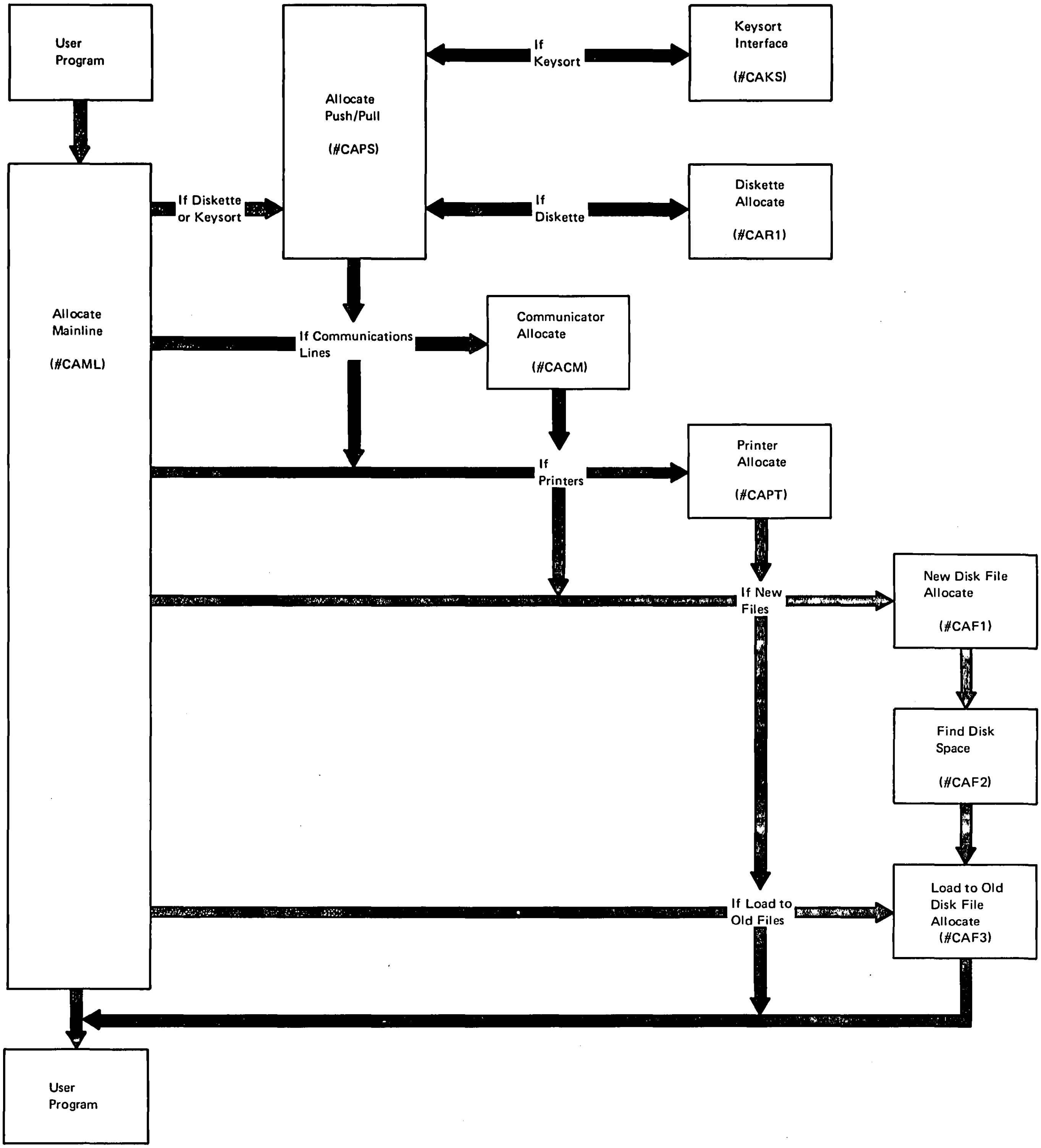

User Program
Allocate Push/Pull (#CAPS)
If Keysort
Keysort Interface (#CAKS)
If Diskette or Keysort
If Diskette
Diskette Allocate (#CAR1)
Allocate Mainline (#CAML)
If Communications Lines
Communicator Allocate (#CACM)
If Printers
Printer Allocate (#CAPT)
If New Files
New Disk File Allocate (#CAF1)
Find Disk Space (#CAF2)
If Load to Old Files
Load to Old Disk File Allocate (#CAF3)
User Program

Figure 3-3 (Part 1 of 2). Normal Allocate Control Flow

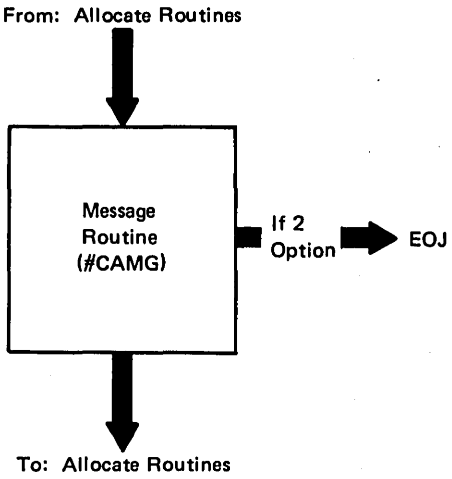


Figure 3-3 (Part 2 of 2). Normal Allocate Control Flow

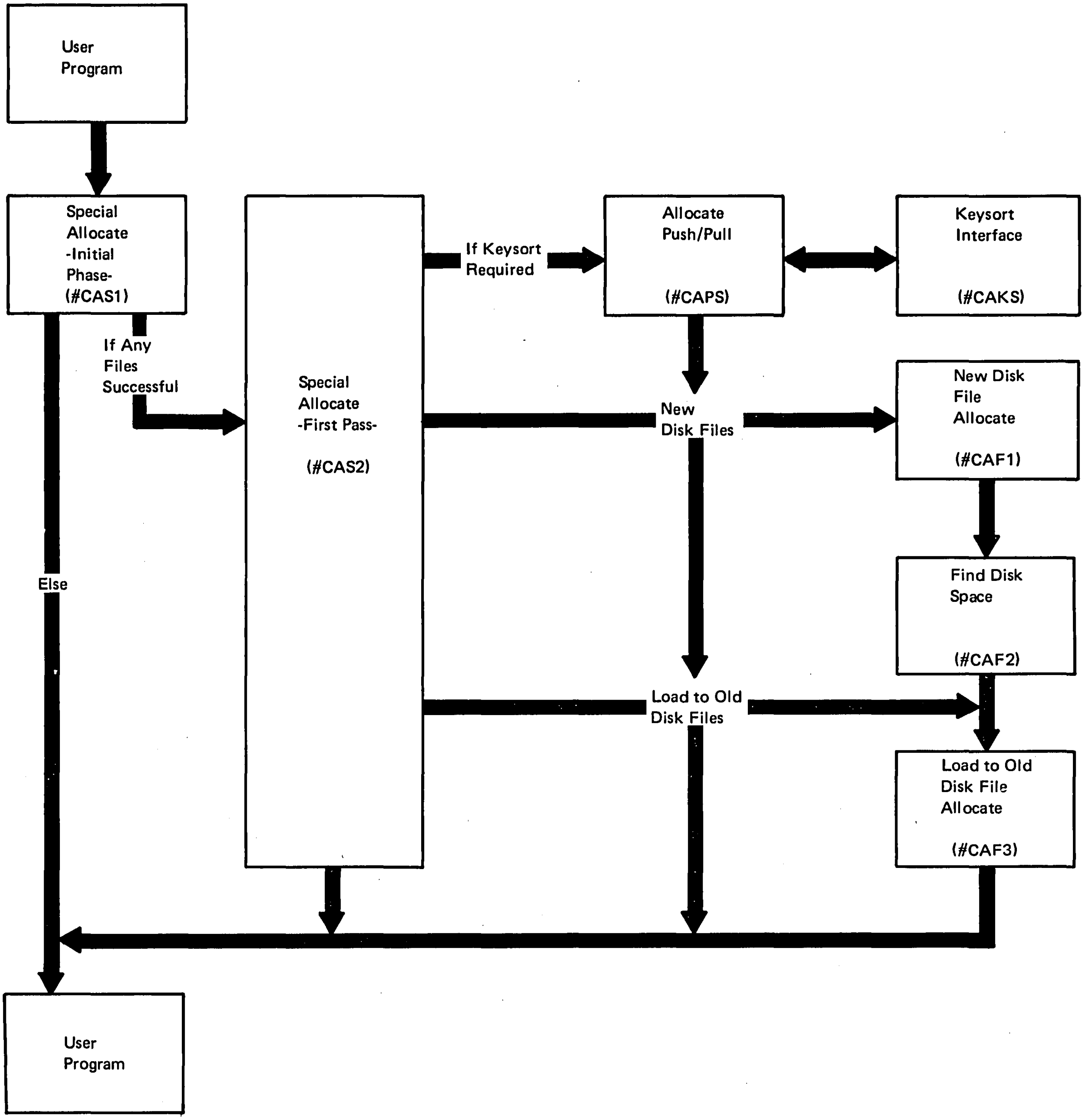

User Program
Special Allocate -Initial Phase- (#CAS1)
If Any Files Successful
Else
Special Allocate -First Pass- (#CAS2)
If Keysort Required
Allocate Push/Pull (#CAPS)
Keysort Interface (#CAKS)
New Disk Files
New Disk File Allocate (#CAF1)
Find Disk Space (#CAF2)
Load to Old Disk Files
Load to Old Disk File Allocate (#CAF3)
User Program

Figure 3-4. Special Allocate Control Flow

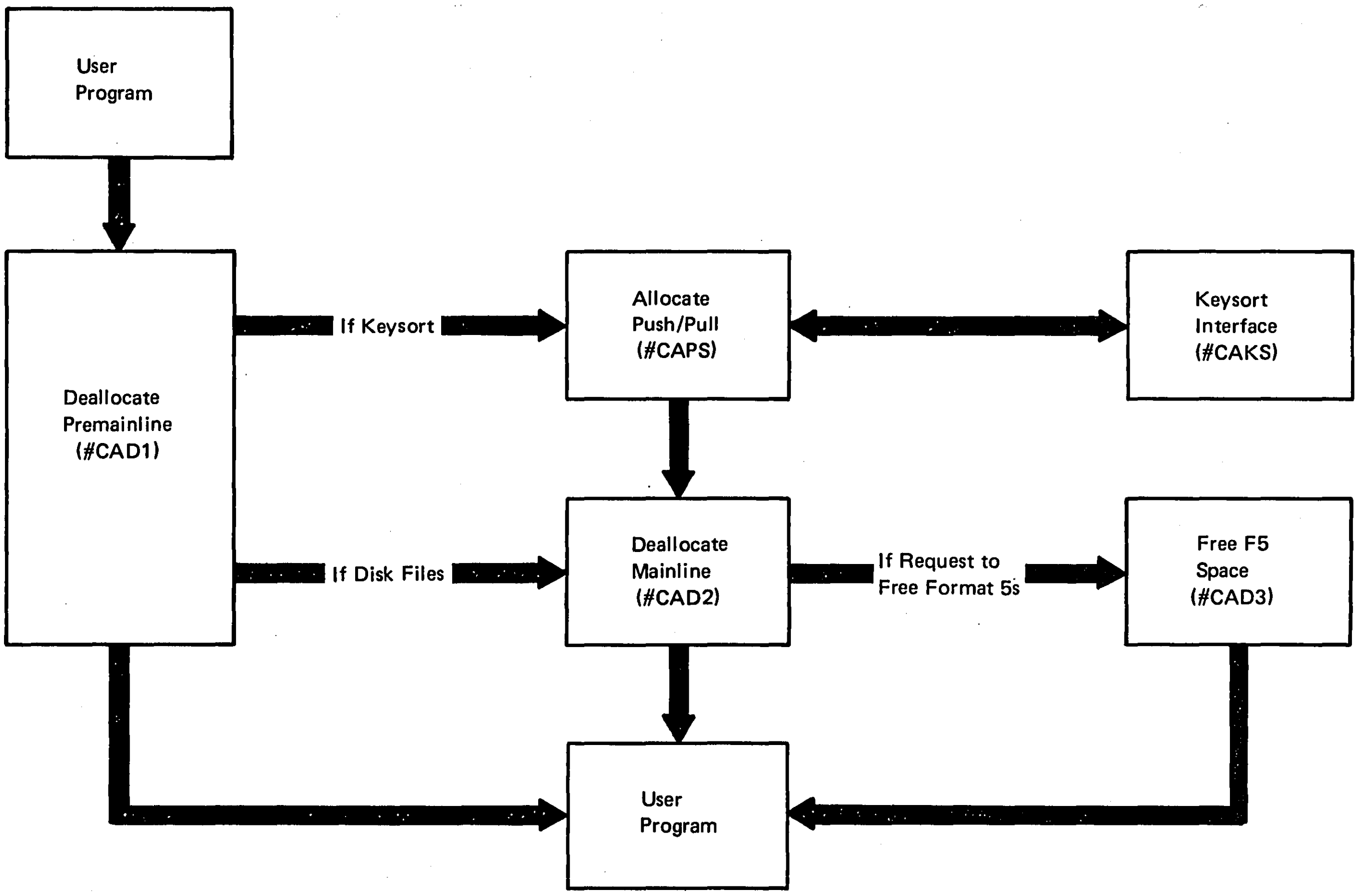
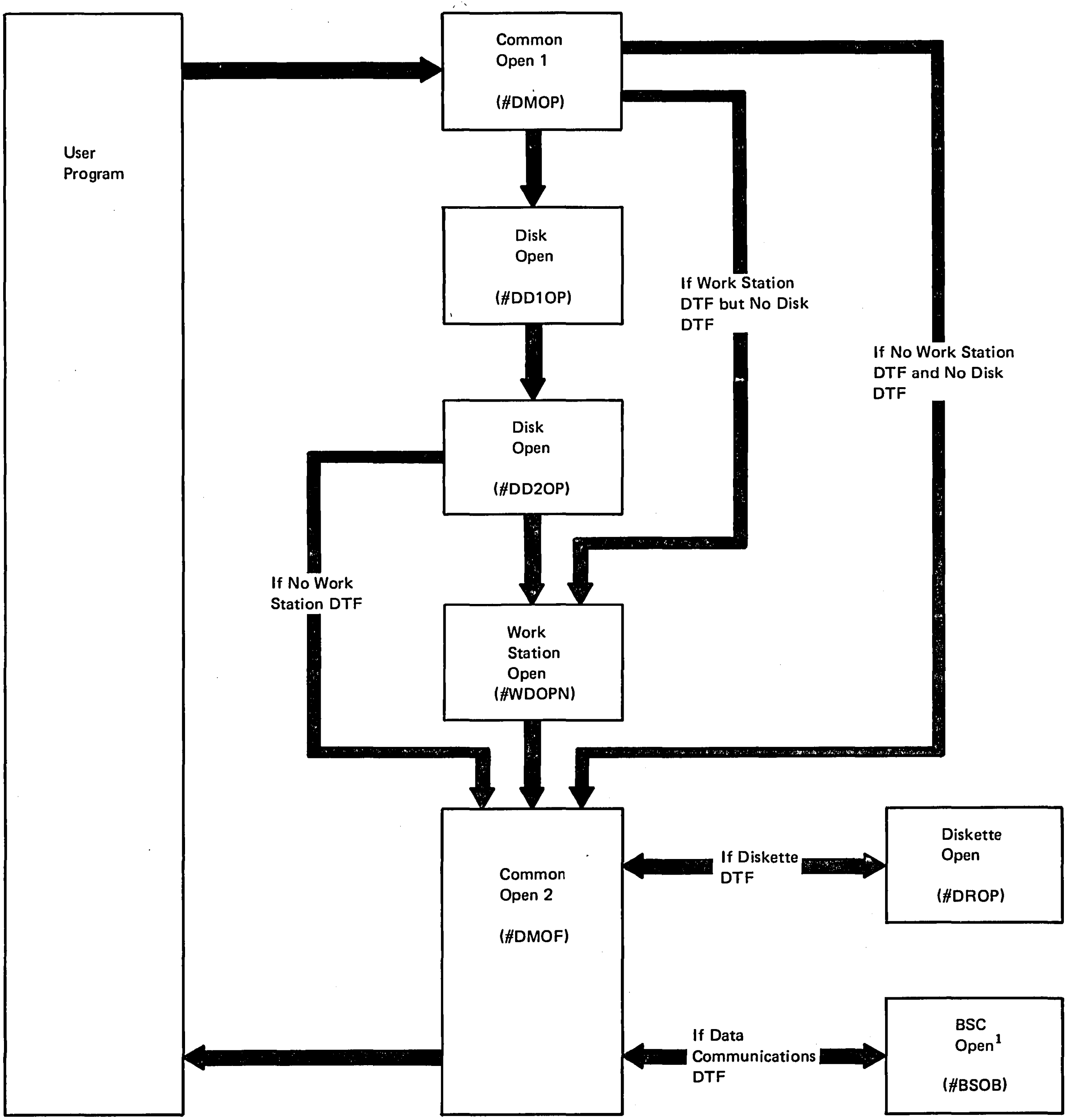


Figure 3-5. Deallocate Control Flow



[1] See Diagram 1.1 in *SSP Logic: Data Communications*

Figure 3-6. Open Control Flow

Introduction

The functions that might be used to run a job are:

- Disk data management
- Sector data management to disk
- Diskette data management
- Diskette end of volume
- Sector data management to diskette
- Printer data management
- Work station data management
- Keysort
- Spool intercept
- Spool print writer

DISK DATA MANAGEMENT

Disk data management functions are performed by a module (#DDDM) that resides in the system nucleus. Figure 4-9 shows #DDDM control flow.

When the user program requests data management services, the request is passed to the router routine. The router, which is part of the disk data management module, determines and passes control to the proper data management module.

If disk data management is requested, control is passed to the main driver routine within #DDDM. The main driver routine examines the first attribute byte in the disk DTF to determine the access method requested.

The proper access method driver receives control from the main driver routine. The driver examines the operation code in the disk DTF to determine which base function or subroutine to call. Figure 4-1 shows the base functions and subroutines used by the various access methods.

The base functions, in conjunction with appropriate subroutines, perform the requested disk I/O operation. Figure 4-2 shows the subroutines used by the four base functions. The actual I/O operation is performed by disk IOS within control storage.

Control is returned to the user program by way of the main driver. Subroutines return control to the base function that called them; base functions return control to the access method driver that called the base function; drivers return control to the main driver and thus control is returned to the user program.

Consecutive Access Method

Figure 4-3 shows the control and data flow in a disk data management operation using a consecutive access method. Data and logic could flow in either or both directions, depending on whether the operation is a retrieval, an update, or an addition of records. Disk data management gets each record from the data buffer or places it in the next adjacent location in the data buffer. When disk data management has filled or emptied the data buffer (a block of records has been processed), the disk I/O supervisor is called to transfer data between the data buffer and the disk.

Control information passes from the calling routine to disk data management through the disk DTF block. Control information passes between disk data management and disk IOS through an IOB (see *Data Areas Handbook).*

Direct Access Method

Input

For an input operation, the calling program passes a relative record number to data management which converts it to a disk address and locates the record in the data I/O buffer (see Figure 4-4). On input, if the record is not in the data I/O buffer, disk data management calls disk IOS to read the disk sectors containing the records into the data I/O buffer. Disk data management places the address of the record in the DTF. The calling program may retrieve the record directly from the data I/O buffer.

Base Function / Subroutine Name

Driver	X PUTD (put data—base function)	X GETD (get data—base function)	X PUTI (put index—base function)	X GETI (get index—base function)	X SRGPI (get/put I/O interface—subroutine)	X SRUKE (check update key error—subroutine)	X SRNIE (get next index entry—subroutine)	Driver Name
CONDRV	X	X						Consecutive Driver
DIRDRV	X	X						Direct Driver
IRADRV	X	X	X	X		X		Indexed Random Driver
ISQDRV	X	X	X	X		X	X	Indexed Sequential Driver

Figure 4-1. Base Functions/Subroutines Used by Access Methods

Base Function Name

Subroutine	X PUTD (put data—base function)	X GETD (get data—base function)	X PUTI (put index—base function)	X GETI (get index—base function)	Subroutine Name
SRRTS	X	X			Convert RRN to SSSD
SRGPI X	X	X	X	X	Get/put I/O interface
SRF1G	X		X		Get format 1 values
SRF1P	X		X		Update format 1 values
SRMOV	X				Move data to I/O buffer

Figure 4-2. Subroutines Used by Base Functions

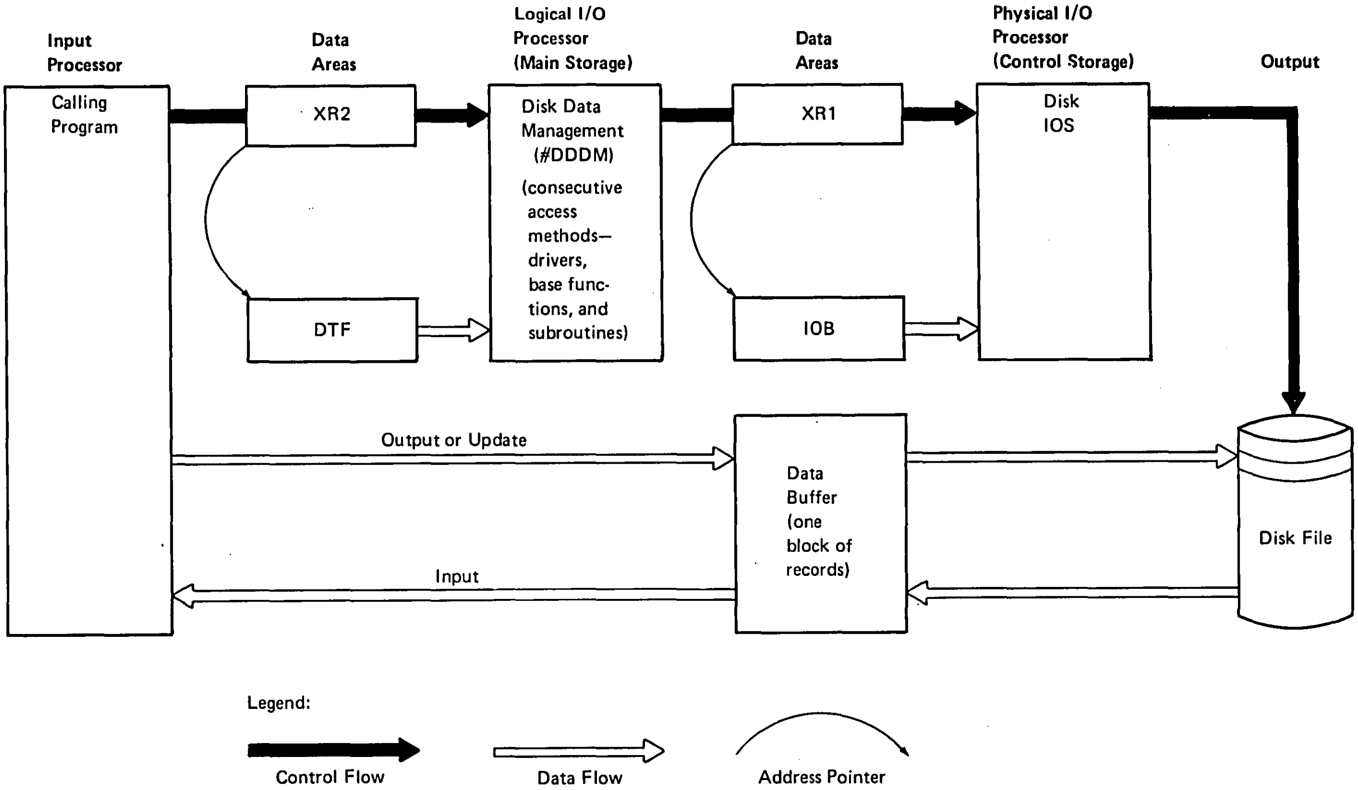

Input Processor
Calling Program
Data Areas
XR2
DTF
Logical I/O Processor (Main Storage)
Disk Data Management (#DDDM)
(consecutive access methods—drivers, base functions, and subroutines)
Data Areas
XR1
IOB
Physical I/O Processor (Control Storage)
Disk IOS
Output
Output or Update
Input
Data Buffer (one block of records)
Disk File
Legend:
Control Flow
Data Flow
Address Pointer

Figure 4-3. Consecutive Processing

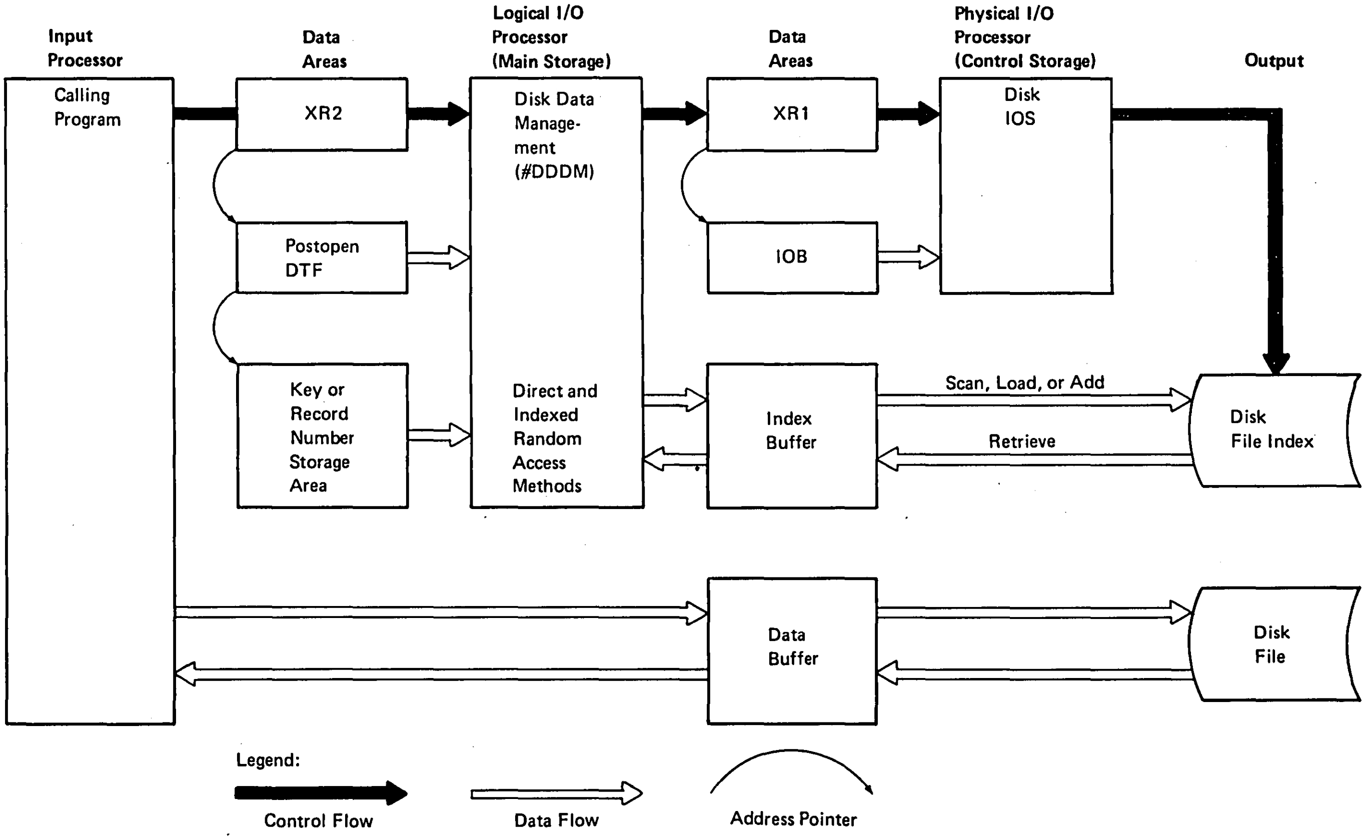


Figure 4-4. Direct and Indexed Random Processing

Update

When updating a record, disk data management retrieves the record as described in the previous paragraph and passes control to the calling program. After the calling program has updated the record, disk data management receives the address of the updated record, moves the record to its original position in the data I/O buffer, and calls disk IOS to write the data I/O buffer. Any retrieved records may be all blanks (as formatted by allocate) and be updated.

Output

For an output operation, disk data management receives the address of the record, moves the record to its location in the data I/O buffer, and calls disk IOS to write the data buffer. Internally, each writing of an output record is immediately preceded by a retrieval of that same record.

Indexed Random Access Method

Input

The calling program passes a key value to disk data management in the key/storage area (Figure 4-7). Disk data management uses the SCAN function to search for the key in the index on the disk. The sector(s) containing the requested logical record are read into the data I/O buffer. The address of the record in the data I/O buffer is passed to the calling routine via the DTF. The calling routine can then obtain the record directly from the data I/O buffer.

Update

When updating a record, the calling program must first retrieve (input) the record as described in the previous paragraph. After the calling program has updated the record, an update operation can be performed. At this time, disk data management receives the address of the updated record, ensures that the key value has not been changed, moves the record to its original position in the data I/O buffer, and calls disk IOS to write the data I/O buffer. The index is not changed. Each writing of an updated record must be immediately preceded by the input of that same record.

Add

When adding a record to a file, disk data management receives the address of the record and searches the index area on disk to see if the key of the record to be added already exists in the index. If the key is a duplicate, an error completion code is returned to the calling routine and the record is not added. For add and output operations, disk data management moves the record to the data I/O buffer and builds an index entry in the index buffer. When either the index buffer or the data I/O buffer must be written to disk, disk IOS is called.

Indexed Sequential Access Method

Disk data management processes indexed sequential files only in ascending key sequence, normally starting with the lowest key in the index (or the lowest key within specified limits) and processing each record in the primary part of the file (Figure 4-5).

Input

Sequential input is accomplished by consecutive reference to each index entry and a retrieval of its associated record. If the requested record is not in the data I/O buffer, disk data management calls disk IOS to read the disk sectors containing the record. When the last index entry in the index buffer has been processed, disk data management calls disk IOS to read the next sector of index.

Update

When updating a record, the calling program must first retrieve (input) the record as described in the preceding paragraph. After the calling program has updated the record, an update operation can be performed. At this time, disk data management receives the address of the updated record, ensures that the key value has not been changed, and moves the record to its original position in the data I/O buffer. Each update operation should be immediately preceded by a retrieve of that same record. When the last record in the data !/O buffer has been processed, disk data management calls disk IOS to write the data I/O buffer to disk if it contains any updated records. The index buffer is not rewritten.

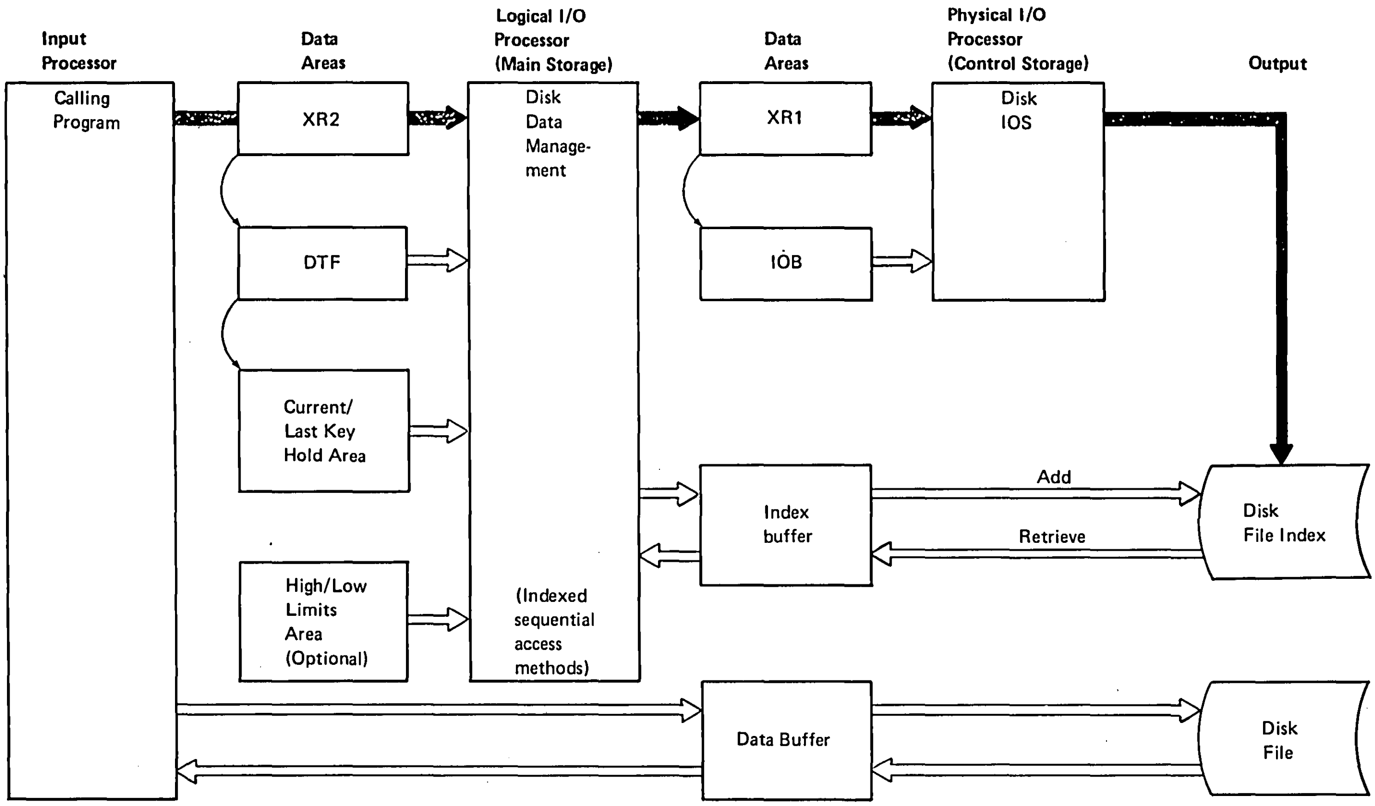

Input Processor
Data Areas
Logical I/O Processor (Main Storage)
Data Areas
Physical I/O Processor (Control Storage)
Output
Calling Program
XR2
DTF
Current/ Last Key Hold Area
High/Low Limits Area (Optional)
Disk Data Manage- ment
(Indexed sequential access methods)
XR1
IOB
Disk IOS
Index buffer
Add
Retrieve
Disk File Index
Data Buffer
Disk File

Figure 4-5. Indexed Sequential Processing

Add

When adding a record to a file, disk data management receives the address of the record and compares its key to the current key and last key values to ensure that the key is in ascending sequence and does not already exist in the file. If the key is not in sequence or is a duplicate, an error completion is returned to the calling routine and the record is not added. For add and output operations, disk data management moves the record to the data I/O buffer and builds an index entry in the index buffer. When a buffer is filled with added records or index entries, or when an input request follows an add operation, disk data management calls disk IOS to write the buffer(s) to disk.

Indexed Sequential/Random Input Access Method

Three types of input operations are performed by the indexed sequential/random input access method. The user may issue a random get and from that point go either forward or backward to sequentially access disk storage.

For a random get operation, $F1KAD contains the address where the record key is located. Record retrieval is the same as for an indexed random input operation.

Get forward is another possible operation. Record retrieval is the same as for an indexed sequential input operation. The record is the next record relative to the last get operation.

Get backward is the third possible operation. Record retrieval is the same as for an indexed sequential input operation. The record is the previous record relative to the last get operation.

Description of Access Method Drivers

Consecutive Driver

Input: For consecutive input, a relative record number is internally generated, (open initializes this DTF field ($F1RRN) to X'0') and this record is retrieved using the get data base function. Then the relative record number is bumped by one and control is returned to the user. The only valid completion code other than normal (X'40') and I/O error (X'41') is end of file (X'42'). The retrieved record is located in the I/O buffer and the DTF field $F1WKB points to the leftmost byte of the record.

The user's I/O buffer must be large enough to contain a complete logical record (having worst-case sector boundary alignment) since this operation functions in locate mode.

Output: For a consecutive output operation, the DTF field $F1WKB must contain the leftmost address of the logical output record. The put data base function is called to move the record to the I/O buffer and write the buffer to disk when necessary. Multiple moves may be required to move the record to the I/O buffer since it is not necessary to have on I/O buffer equal to or larger than the record length. The DTF field $FINXR is used to determine the location for each output record. The only completion code other than normal and I/O error is end of extent (X'70').

Add: An add operation functions similar to an output operation.

Update: Consecutive update first checks to ensure that the previous operation was a get. If this check fails, a completion code of X'45' is set to indicate an update operation was not preceded by a get. When a valid update operation is determined, the internal relative record number is decremented by one to point to the last record retrieved and the record is put to the I/O buffer and disk if necessary.

Special Considerations for Consecutive Processing: When SIAM is specified, the I/O buffer is written and/or read for every operation performed. When doing an update operation, the I/O buffer is reread between the get and put operations. Therefore the user must move the retrieved record out of the I/O buffer prior to doing the output portion of his update.

Direct Driver

Input: When doing a direct input, the caller must have the relative record number of the requested record in the record address area ($F1KAD), pointed to by the DTF. $F1KAD must contain the address of an area 3 bytes in length if the RRN is a binary value or 10 bytes long if the RRN is a decimal value.

A check is made to see if the requested record is in the present I/O buffer. In any case, once the record is in the buffer, it is located via $F1WKB and control is passed back to the calling routine with a normal completion code of X'40'. The only other return code from direct input (except for I/O error) is X'44', record out of extent. All direct input operates in locate mode thus requiring that the user's I/O buffer be large enough to contain at least one complete record with worst case sector boundary alignment.

Update: Direct update first ensures that the previous operation was a get and then the updated record is moved to the I/O buffer and written to the disk if necessary. The same restriction holds true for direct as for consecutive update. (See *Special Considerations for Consecutive Processing.*) A possible return code from direct update operations is X'45', update before input.

Special Considerations for Direct Processing: Only three operations are supported for direct processing. They are the input, update and output op codes. Since a direct file is considered to be full of blank records at allocation time, a user may not add to a direct file. The output operation forces an input followed by an update with data in management. Direct processing always computes the N-byte in the IOB upon entry to any operation to enable dynamic blocking during processing. The N-byte is computed to be the value of the leftmost byte of the block length ($F1BKL) minus one.

Indexed Random Driver

Input: Indexed random input obtains the key of the requested record from the addressed location in $F1KAD. The key is compared to the prime key bucket to see if the record is in the prime portion of the index. If the requested key is higher than the prime bucket, the key does not exist in the prime area so a check is made against the overflow bucket. Again if the compare is high, the key does not exist in the overflow and a return code of X'44' is set indicating no record found. If the key is found to possibly exist in the prime index, a scan is issued with an argument of high or equal. If the scan returns a high or no hit value, the scan starts over in the overflow area with an equal only argument. If the scan of the prime returns an equal indicator, the record has been found and the RRN is retrieved from the index buffer and the associated data record is also retrieved. When the scan of the overflow area returns an equal found, the RRN is retrieved from the index buffer and then the associated data record is retrieved. If the scan of the overflow yields a no hit value, a return code of X'44' is set indicating no record found.

When a scan equal is returned from the scan function, and the RRN has been located, the RRN value is converted to an SSSD value and that sector is read into the data I/O buffer. Then the record is located in the I/O buffer and the leftmost address returned by way of $F1WKB, along with a completion code of X'40'. The only return code other than normal from indexed random input is record not found (X'44').

Add: Indexed random add expects the address of the key for an add record to be in the DTF at $F1KAD. The first step of the add operation is to scan the prime and overflow area if necessary to see if the key already exists. If the key is found to already exist in either the prime or overflow area, a return code of X'60', duplicate add attempted, is set and control returned to the caller. If the key is not found, the add operation may continue. $F1WKB must contain the leftmost address of the record to be added. A check is made to ensure that the record will fit in the data area on disk. If it will not fit, a return code of X'70' is set indicating end of extent. If the record will fit in the extent, the record is moved to the I/O buffer and written to disk if necessary. Next the index entry is built. A check is made to ensure that the index entry will fit in the index extent. If it will not fit, a return code of X'70' is set indicating end of extent and control is returned to the user. If it will fit, the index entry is moved to the index I/O buffer and written to disk. The add key is compared against the overflow key bucket and if the new key is higher, the new key is moved to the overflow key bucket. The SSSD of the new key in the overflow key bucket is also moved to the format 1.

Output: Indexed random output first moves the data record from the work buffer ($F1WKB) to the data I/O buffer, and write it to disk if necessary. Next an index entry is built and moved to the index I/O buffer and written to disk if necessary. No key sequence checking is done and no checks are made to ensure that duplicate records are not put to the file.

Update: Indexed random update first ensures that the key of the record being updated is the same as the key of the last record retrieved. If not an update key error (X'50') is set and control returned to the caller. A check also ensures that the previous operation was a get. If not, error X'45' update before input is set and control is returned to the caller. If all is well up to here, the updated record is moved from the record buffer to the I/O buffer and written to disk. The same restriction holds true for indexed update as did for consecutive update (see special considerations for consecutive processing).

Special Considerations for Indexed Random Processing: When doing index random output, it is possible to put duplicate keys to a file. When processing under indexed random input, and duplicate keys do exist, only the first key entry is accessible. When doing adds to a file, the sort and merge bits in the F1 are set as follows:

	Higher Than Prime	Lower Than Prime
Higher Than OVFL	No Setting	Merge
Lower Than OVFL	Sort	Merge + Sort

Indexed Sequential Driver

Input: When doing indexed sequential input, the first record retrieved is the first index entry followed by each record, by key sequence in the index. The first thing that is checked is to see if processing is within limits. All indexed sequential processing is processed as though limits are specified. If limits are not specified, the limits are set to the start and end of the file. When end of file is reached and limits are specified, the open limits transient may be called to set net limits. This transient call is made only if a get request is received and the completion code contains a X'42', end of file.

Update: Processing for update must be preceded by an input of the same record. The record key is checked to ensure that it has not been updated. If it has been updated, the return code is set to X'50', update key error. The data is moved from the record buffer to the data I/O buffer, and written to disk if necessary.

Add: When doing an indexed sequential add, you must first read the first record past the location into which you want to add. The following example explains this procedure.

If the file you are adding to contains keys 1, 2, 5, 10, 20 and 50, and you wish to add record key 3. You must first read records 1, 2, and 5. At this point you may add records 3 and 4, in that order. If you wish to add record key 17, you must continue reading up to record 20. At this point you may add records 11 through 19. If you wish to add records greater than key 50, you must read to end of file and at that point you may add as many records as there is room in the file. Remember that each add must be in ascending order by key.

Special Considerations for Indexed Sequential Processing: Indexed sequential add operations may not be processed under a file sharing environment. When processing a file containing random adds and the index has not yet been sorted, you do not have access to the records in the overflow area (added records).

Indexed Sequential/Random Input Driver

This access method supports only input operations. Three types of input may be specified.

- Random get — X'80' op code
  The record key must be at the address contained in $F1KAD. The retrieval of this record is the same as any indexed random input operation.

- Get forward — X'84' op code
  This operation is the same as any indexed sequential input operation. The next key, relative to the last get, is retrieved.

- Get backward — X'82' op code
  This operation causes the previous record, relative to the last get, to be retrieved.

Each successive get is based on the last get operation, except for a random get which only gets the record requested. If the first operation is a get forward, the record referenced by the first index entry is the first record in the file to be retrieved. A return code of X'42' is issued whenever end of file is reached, either forward or backward. A X'44' is set if a random get cannot be found.

SECTOR DATA MANAGEMENT TO DISK (#DDSM)

Sector data management resides in the system library and is loaded by disk open. It is provided for user and SSP functions which require movement of considerable amounts of data to or from disk. Sector data management utilizes a single input/output buffer which is filled by a single read operation, or written to disk with a single write operation. Sector data management operates with no consideration of logical record length.

Data Transfer Modes — Locate and Move

The base functions of #DDDM execute in one of two modes of operation: locate mode and move mode.

In locate mode, a record is not moved after the data management module places it in the data I/O buffer. Instead, the address of the record is placed in the DTF, and the calling program retrieves the record directly from the data I/O buffer.

In move mode, the base function routines in #DDDM receives records from a calling program in an area (work buffer) that is separate from the data I/O buffer. In all output operations, the record is received from the calling program's output area. The base function routine moves the resident portion of the record from the work buffer, calls disk IOS to write the data I/O buffer, then moves the remaining portion from the work buffer.

For writing data to disk, data management operates in move mode.

For reading data from disk, data management operates in locate mode.

Double Buffering

Consecutive input and consecutive output files can use double buffering. In double buffering, one buffer twice the size of the specified block length is used.

DISKETTE DATA MANAGEMENT (#DRDM)

Diskette data management resides in the system library and is loaded by diskette open. Data management is capable of processing System/32 created, System/34 created, and basic data exchange files.

Five access types are provided:

1. Put basic record (PBR)
2. Get basic record (GBR)
3. Put system record (PSR)
4. Get system record (GSR)
5. Add system record (ASR)

Basic data exchange files can be described as unspanned fixed length logical records of length less than or equal to sector size with a fixed physical record size equal to the sector size (sector size is 128 bytes or 256 bytes). Physical records may be blocked. Data management ensures that logical records of less than sector size are properly padded with binary zeros on output and that the logical records are properly deblocked on input.

System files can be described as blocked/spanned fixed-length records in fixed-length blocks. Records can span sector and volume boundaries. Record length must not exceed 4K (4096 bytes). Block length (physical I/O buffer size) should be a multiple of the diskette sector size and equal to or greater than the record length. For diskette 1, a block length of 3-1/4K (3328 bytes), which equals one diskette track, should be used if possible for standard format diskettes. For extended format diskettes, ideal buffer size is 4K (4096 bytes), which is equal to the extended format diskette track size. For diskette 2D, a block length of 6-1/2K (6656 bytes) should be used for standard format diskettes and a block length of 8K (8192 bytes) should be used for extended format diskettes.

Either move or locate mode can be used for output as well as input. Using move mode, records are moved by data management between the physical input/output buffer and a logical record area provided by the calling program. Using locate mode, the physical I/O buffer is shared by data management and the calling program, with a record pointer to the leftmost byte of each input or output record set by data management in the DTF at $I1WKB. When using locate mode for output, record size should be a multiple of 128 and may not span I/O buffers.

If a block length (physical I/O buffer size) is equal to the size of a track and track I/O is requested by the calling program, data management will operate with full track I/O requests and may process only part of the I/O buffer on the first call. Using move mode, track alignment is transparent to the calling program.

Using track I/O with locate mode, the calling program must be able to process a partial I/O buffer (as small as 128 bytes). For input files, diskette data management places the partial buffer of data within the last part of the input buffer and sets a pointer ($I1WKB) to the first byte of valid data. For output files, diskette open sets this pointer indicating to the calling program where data should begin within the partial buffer. If the initial buffer size ($I1BKL) which is calculated by open causes a record to span the I/O buffer, the calling program must block or deblock the record.

After the first partial buffer is processed by diskette data management, by calling diskette IOS to write or read data, $I1WKB is restored to point to the first byte of the I/O buffer, and I/O buffer size ($I1BKL) is restored to equal one track (3328, 4096, 6656, or 8192 bytes).

DISKETTE END OF VOLUME (#DRNV)

The end of volume transient routine is normally called by diskette data management after the last sector of data on a diskette has been processed.

For input files, the data set label is rewritten to the diskette VTOC if the expiration date has been changed.

For output or add files, the data set label is written to the diskette VTOC, with a volume sequence number and an indication that the file is continued to another diskette.

A diskette insert message is issued for the system operator and processing is suspended.

When a new diskette is inserted and ready for input files, EOV calls VTOC read/write to search the diskette VTOC for a data set label corresponding to the file being processed. When it is found, the volume sequence number is checked. The DTF and IOB are then updated and processing continues.

When a new diskette is inserted and ready for output files, EOV ensures that the diskette contains no active files. In doing so, expired files may be deleted. The file being processed is allocated at the beginning of the diskette data area and processing continues.

EOV occurs without the calling program regaining control. An indicator is set to let the calling program know that a volume transition has occurred.

In case of a permanent diskette write error during data output, diskette data management calls close to issue an error message which has a continue option. If continued, the file being written is cut off at the end of the last good block of data written and end of volume is called to continue the file to another diskette.

SECTOR DATA MANAGEMENT TO DISKETTE (#DRSM)

Sector data management resides in the system library and is loaded by diskette open. Sector data management is capable of processing System/32 and System/34-created files. It is provided for System/34 SSP functions which require movement of considerable amounts of data to or from diskette. Sector data management uses a single input/output buffer which is filled by a single read operation, or written to diskette with a single write operation. Sector data management operates with no consideration of logical record length.

PRINTER DATA MANAGEMENT

Printer data management is an SSP function that is part of disk data management. While in use, printer data management resides in the system nucleus of main storage along with disk data management, work station data management, and printer IOS.

Printer data management's main function is to convert user print requests within the printer DTF to printer IOB requests to be passed to the printer IOS.

Printer data management uses move mode to transfer data from a user-supplied logical data area into a physical data area. The data is then printed from the physical data area, also referred to as the I/O buffer.

When the printer data management module (#DPDM) is entered, the completion code ($PRCMP) is set to X'40' to indicate normal completion.

Printer data management then moves the skip and space values from the DTF to the IOB ($IOBPSPA).

If print is requested in the DTF, the data to be printed is moved from the user supplied logical buffer into the physical buffer. Also, the data string length is moved into the printer IOB ($IOBPLNG), and a print indication is set in the IOB control byte ($IOBPCTL).

DTF attribute byte three ($PRAT3) is tested to see if forms alignment is requested and print spool is not active. If both conditions are met, the printer alignment transient (#DPAL) is called to supply forms alignment information to the system operator. After the system operator aligns the forms, #DPAL returns control to the user program.

The prepare print buffer supervisor call instruction is issued. This SVC inserts printer control codes into the print buffer to effect the requested skip and space operations; it also maintains a record of the current line number in the printer IOB.

Printer data management moves the current line number from the IOB ($IOBPCLN) to the DTF ($PRCLN), and checks for page overflow. If overflow has occurred, the overflow completion code (X'48') is set in the printer DTF ($PRCMP).

Printer IOS is then entered. Upon return from IOS, printer data management waits until the contents of the print buffer have been moved to the printer. Then the IOB status byte ($IOBPSTA) is checked for a permanent error. If a permanent error has occurred, the DTF completion code ($PRCMP) is set to X'41'. Control is then returned to the user.

WORK STATION MANAGEMENT

Work station management allows the application programmer to present data on a display screen by providing only a string of data fields. The data is displayed on the screen in predefined format. Conversely, on input, the formatted data is taken from the display screen and returned to the user as a string of data fields. All device-dependent control characters, orders, constants, and field attribute characters are inserted or deleted by work station management. The work station management component is made up of two parts: a screen format generator routine and a data management routine.

WORK STATION DATA MANAGEMENT

Work station data management is a supervisor routine which runs as a subroutine under the user task and controls all I/O requests to the work stations. Work station data management is composed of a nucleus resident mainline module and transient routines. The transient routines process requests not handled by the mainline routine.

Two versions of the work station data management are used. Either a main storage resident version or a transient version can be used.

An SVC is issued by the application program for work station data management services. Address of the work station or printer DTF is in index register 2 (XR2).

When the work station data management mainline module is entered, a series of diagnostic checks is made on the request. If terminal errors are encountered, the task is terminated with the appropriate termination code. If less serious errors are found, control is returned to the application program with a return code in the DTF. The symbolic terminal name is resolved into a terminal unit block (TUB) address. If the operation is a:

- *Put:* The format index is scanned for the disk address of the requested format, and the text and field descriptor table (FDT) is read into the data communications buffer area. The application data and the FDT are scanned, inserting the data in the appropriate place in the text stream. If any indicators are specified for overrides in the FDT, the indicators are checked as each field is processed. The IOB in the TUB is marked for a put operation and if the request is for a put-wait, a wait is issued on the TUB. When the wait is satisfied, or immediately for a put-no-wait, control returns to the application program by way of the instruction address register (IAR).
- *Invite input:* The invite bit in the work station IOB is set on and control returns to the user by way of the IAR.
- *Accept input:* The invite input count (TCBINVCT) is checked for zero or no outstanding invites. If this is the case, control returns to the application program with the appropriate return code. If (TCBINVCT) is nonzero, a general wait is issued. When the wait is satisfied, the address of the completed IOB is in XR1. If TUBIIS is on, the input is in response to an explicit invite input, and the data is read into the user program record area. If TUBIMI is on, this is data with a program request. If the program is an MRT, MRTMAX is checked to see if this request will exceed the maximum number of requesters. If the limit has been reached, the noskip bit is set off in the TUB and the wait reissued. If not, the data is moved to the user area and control passed to the application program.
- *Get:* The invite bit is set on in the IOB and a wait on the IOB takes place. Upon completion of the wait, the data is read into the user's record area and control returns to the application program.
- *Stop invite input:* The terminal unit block is checked to see if the invite operation had ended. If it has completed, the application is notified by way of a return code that the stop invite failed and the data is available. If the invited TUB is incomplete, the invite is canceled and the user is notified that the operation was successful.
- *Put overrides:* This operation is handled the same as a put, with the exception that only the FDT is read into the data communications buffer. The text stream is constructed from the fields that have indicators specified for overrides, using the appropriate indicator settings. Only the fields or attributes using overrides is sent to the display.
- *Acquire terminal:* The request is diagnosed, and if valid, #WDDQ attempts to attach the specified work station to the user program. If the work station is unavailable, the application has the option of enqueing the work station. If the option is not specified, a return code notifies the application that the acquire has failed for that reason. If the work station is available, it is attached to the application. A stop invite is issued and control returns to the user.

- *Release terminal:* A call is issued for transient #WDDG which dequeues the TUB from the application program. If the application is an MRT program with more requesters queued (MRTMAX exceeded), the next queued requester will be posted complete. The released terminal unit block is then passed to the command processor.

- *Get terminal attributes:* A call is issued for transient #WDDQ which will build, in the application program record area, a series of data bytes describing the following attributes of the specified work station:
  - Allocation status
  - Screen/printer
  - Screen size
  - Online/offline

- *Save, restore, print, roll, erase, or clear:* #WDDB builds, in the data communications buffer, a data stream to execute the various commands.

KEYSORT

The function of keysort is to arrange indexed disk file index entries in ascending order based on the key portion of the index.

Figure 4-6 is an overview of the keysort program. Index entries are sorted to produce an ordered index.

The keysort user must provide a 12K-byte area in main storage. Figure 4-7 shows how this area is used.

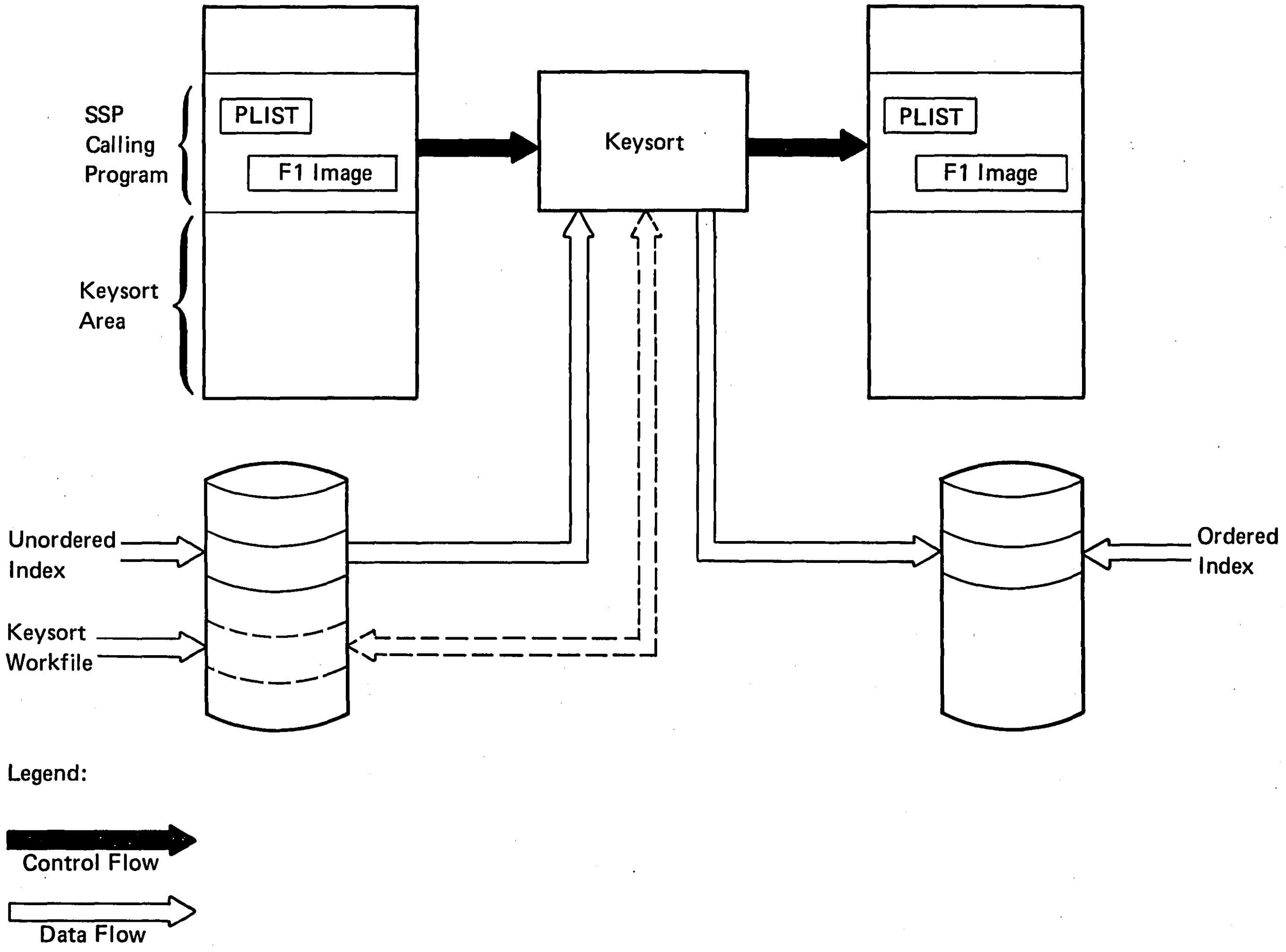


Figure 4-6. Keysort Program Overview

The user must also provide a 12-byte keysort parameter list with XR2 pointing at the list. The parameter list is formatted as follows:

0	LIST ID	DKACB	1
2	F1 image		3
4	KS partition size		5
6	Ret cond byte	keylength	7
8	Duplicate key		9
10	Return to keysort		11

The first 6 bytes are set by the calling program and the last 6 are set by the keysort program.

The parameter list fields are defined as follows:

Displacement	Length	Description
0	1	Parameter list ID, C'K'.
1	1	Duplicate key action control byte (DKACB): ● Bit 0 (X'80') – Activates duplicate key checking. ● Bit 1 (X'40') – Activates detail duplicate key comunication (if bit 0 also on). ● Bits 2-7 – Reserved (must be 0).
2-3	2	Pointer to beginning of format 1 image in translatable storage.
4-5	2	Keysort partition size (bytes).
6	1	Return Condition Byte: ● Bit 0 (X'80') – Duplicate key detail return. ● Bit 1 (X'40') – Duplicate key summary return. ● Bit 2 (X'20') – System error return. ● Bit 3 (X'10') – I/O error within index. ● Bit 4 (X'08') – I/O error on work file – index scrambled. ● Bits 5-7 – Reserved.

Note: The remainder of the list is set only if the Return Condition Byte bit 0 is on.

7	1	Key length.
8-9	2	Pointer to beginning of duplicate key.
10-11	2	Address in keysort to return to after detail duplicate key processing by calling program.

After a special return to the calling program for detail duplicate key processing occurs, the calling program may or may not return to keysort. If the calling program does not return, the index is invalid. If the calling program does return, the duplicate key is retained and processing continues according to the current duplicate key action control byte.

A system error special return is triggered when problem determination is made by keysort. This condition may result from an invalid parameter list, an invalid format 1 field, or a processing error by keysort.

All keysort IOBs are constructed so that Disk IOS will not issue I/O error messages. Permanent disk I/O errors fall in one of three categories during keysort:

1. Those occurring within the index.
2. Those occurring on the keysort work file after the Index has been partly altered.
3. Those occurring on the keysort work file where the Index has not been altered.

An I/O error in category 1 results in bit 3 of the return condition byte being set and a final special return to the calling program.

An I/O error in category 2 results in bit 4 of the return condition byte being set and a final special return to the calling program.

An I/O error in category 3 results in keysort automatically restarting and performing the indicated sort and/or merge without using a work file. This category does not cause a special return.

Logging duplicate key messages, when necessary, is the responsibility of the calling program because no single set of options and resultant actions satisfies the requirements of all calling functions.

The modules that make up the keysort program are:

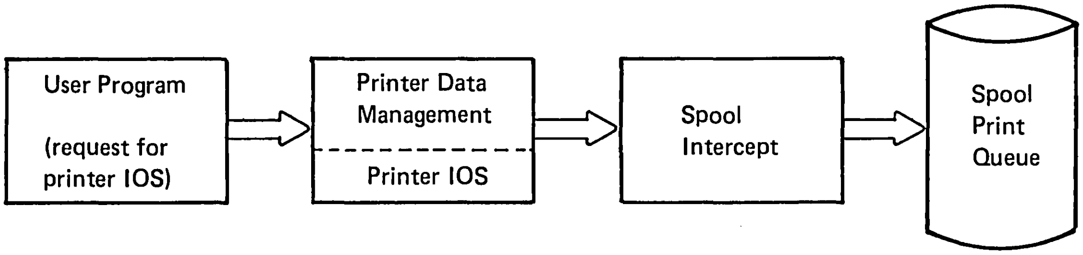
Module Name	Function
#DDKAA	Keysort control (including common)
#DDKAB	Set preliminary internal values
#DDKAC	Design sort
#DDKAD	Auto-allocate work file
#DDKAZ	End of assignment phase
#DDKEP	End of pass
#DDKLL	Three-phase sort control
#DDKSS	Sort-in-place
#DDKWG	Deblock from work file
#DDKWP	Block for work file
#DDK1A	Phase 1 control
#DDK1E	Phase 1 internal sort (repl/sel)
#DDK1G	Deblock and degap from input
#DDK1R	Read input
#DDK2A	Phase 2 control
#DDK2E	Phase 2 intermediate merge
#DDK3A	Phase 3 control
#DDK3E	Phase 3 final merge
#DDK3P	Block and regap for output
#DDK3W	Write output

Figure 4-12 shows the control flow between keysort program modules. Refer to this figure when using the keysort diagram.

SPOOL INTERCEPT

Spool is an optional feature that intercepts system printer commands and places them on disk storage, creating a print queue. When requested, the spool writer retrieves records from the spool queue and outputs them to the system printer.

The spool intercept routine resides in the system nucleus portion of main storage. Intercepted print lines are compressed to remove strings of more than three blanks. This compressed data along with line control information is then written to the spool file.

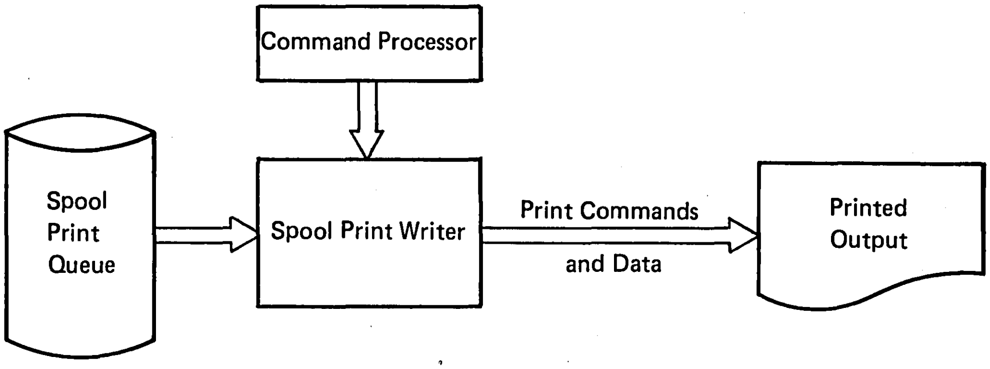


The spool file resides on disk and is made up of a primary file and up to five equal size extents. The primary file, allocated at IPL time, contains information about the spool file entries as well as data areas for the compressed print records. The spool file extents are allocated separately, and only when the primary file and all previously allocated extents become filled. The size given for the spool file at system configuration time is the size of the primary file in number of blocks.

When space is no longer available in the spool file, the error message SPOOL FILE IS FULL is issued. The system operator can display the print queue to determine if the print writer can be started to remove entries from the print queue to free up space in the spool file. The SPOOL FILE IS FULL message can be responded to with a retry option when spool file space is available. If there are no completed entries, the existing spool file is not large enough to accept more records. A CONTINUE option will close the spool file, post the print writer to indicate print output exists, and issue another halt indicating the spool intercept routine is waiting for the spool writer to complete. When the writer completes and the last halt is responded to, spool intercept attempts to obtain file space again and continue processing.

SPOOL PRINT WRITER

The spool print writer runs in main storage to print output from the print queue. The writer is loaded only when output exists to be printed and remains in main storage only while printing entries from the print queue. The writer operates as a utility program, independent of the rest of the system, and requires an 8K-byte user partition. An optional autowriter feature exists which causes the print writer to be loaded without operator command whenever output exists in the print queue. If the autowriter feature is not selected at system configuration time, the operator must issue a START PRT command to initially evoke the print writer.



Once the START PRT command is issued, the writer prints until the queue is empty or a command issued. If the command issued is STOP PRT, the message WRITER STOPPED is displayed and end of job called.

After data has been printed from the spool file, the disk space is freed up. The free entry is placed on the available queue to allow reuse by spool intercept.

Method of Operation

This section contains functional diagrams for routines used to run a job. They are:

- Disk data management functions
- Sectorized disk data management functions
- Diskette data management functions
- Diskette end of volume function
- Printer data management function
- Work station data management function
- Keysort function
- Spool intercept function
- Spool print writer function

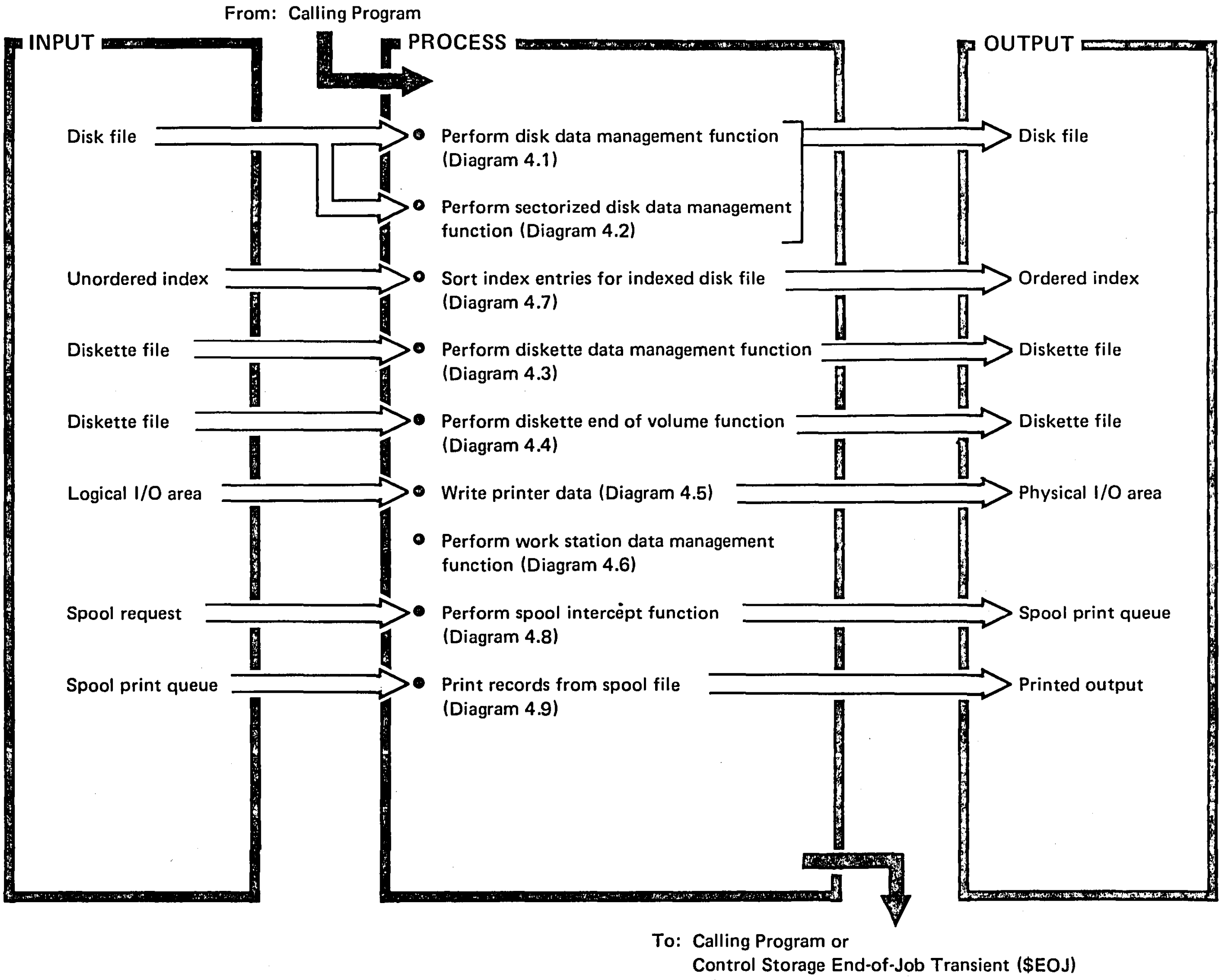
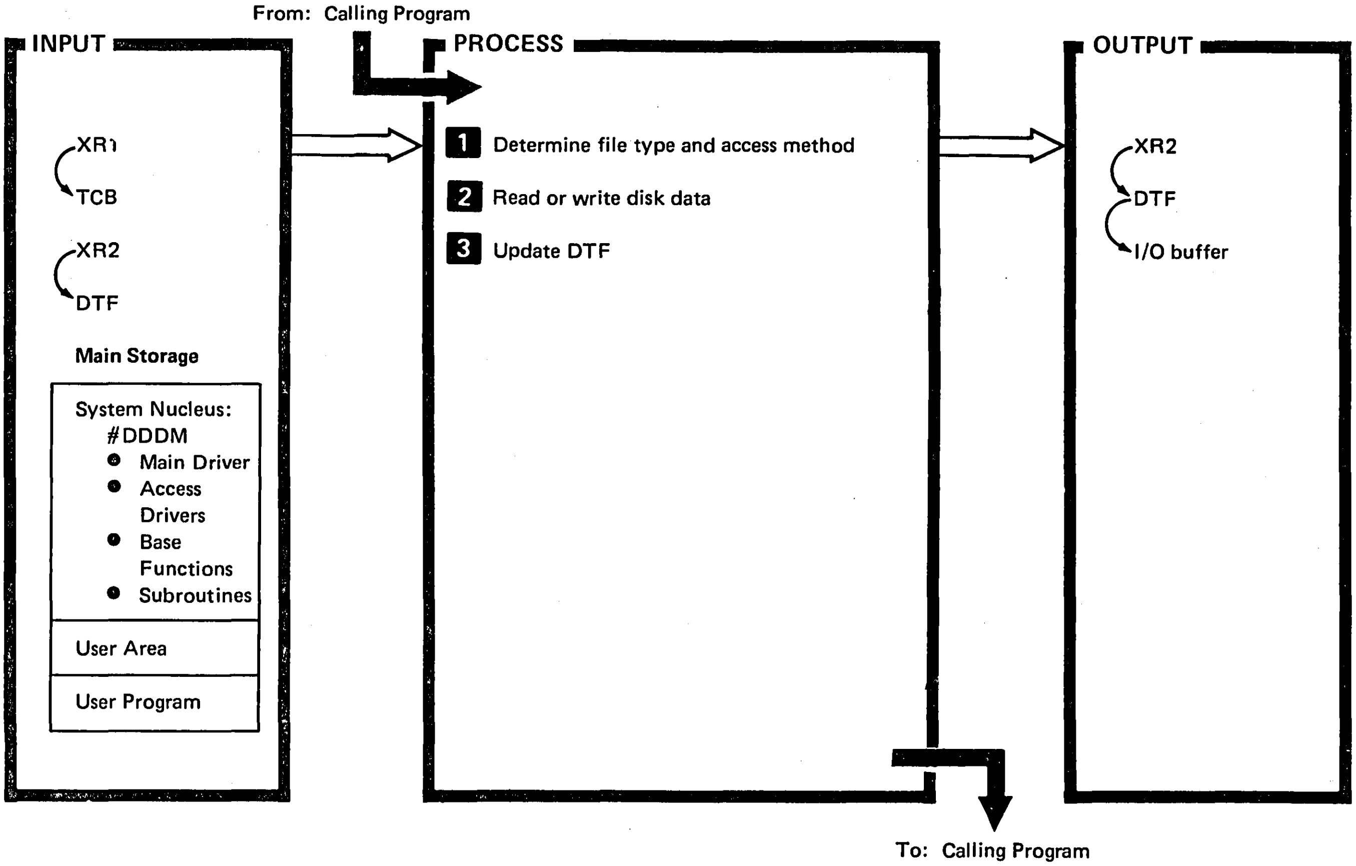

From: Calling Program
INPUT
PROCESS
OUTPUT
Disk file
Perform disk data management function (Diagram 4.1)
Perform sectorized disk data management function (Diagram 4.2)
Disk file
Unordered index
Sort index entries for indexed disk file (Diagram 4.7)
Ordered index
Diskette file
Perform diskette data management function (Diagram 4.3)
Diskette file
Diskette file
Perform diskette end of volume function (Diagram 4.4)
Diskette file
Logical I/O area
Write printer data (Diagram 4.5)
Physical I/O area
Perform work station data management function (Diagram 4.6)
Spool request
Perform spool intercept function (Diagram 4.8)
Spool print queue
Spool print queue
Print records from spool file (Diagram 4.9)
Printed output
To: Calling Program or
Control Storage End-of-Job Transient ($EOJ)

Diagram 4.0. Overview of Running a Job

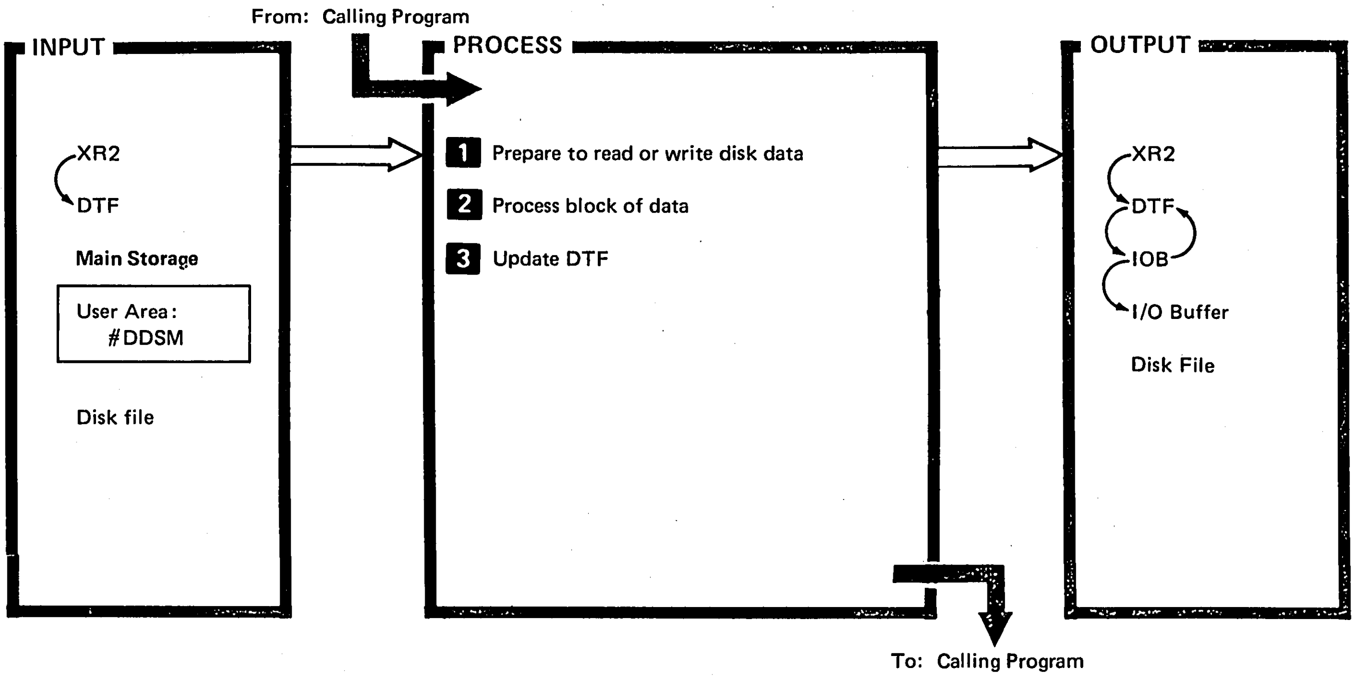


DESCRIPTION	MODULE/ ROUTINE
1 Enter disk data management from calling program.	#DDDM
Ensure DTF is open.	Router
Inspect DTF device code ($F1DEV) to determine data management type requested.	
If disk data management, call main driver (MAINDRV).	
Do sector enque for add operation.	
Inspect DTF attribute bytes to determine access method requested.	MAINDRV
Call driver corresponding to requested access method (Figure 4-8).	
2 Inspect DTF operation code ($F1OPC) to determine base function to call (Figure 4-8).	Appropriate Driver
Call appropriate base function.	
Determine subroutines to call (Figure 4-8).	Appropriate Base Function
Update IOB for IOS use.	
Do sector enque when necessary for update.	

Diagram 4.1 (Part 1 of 2). Perform Disk Data Management Functions

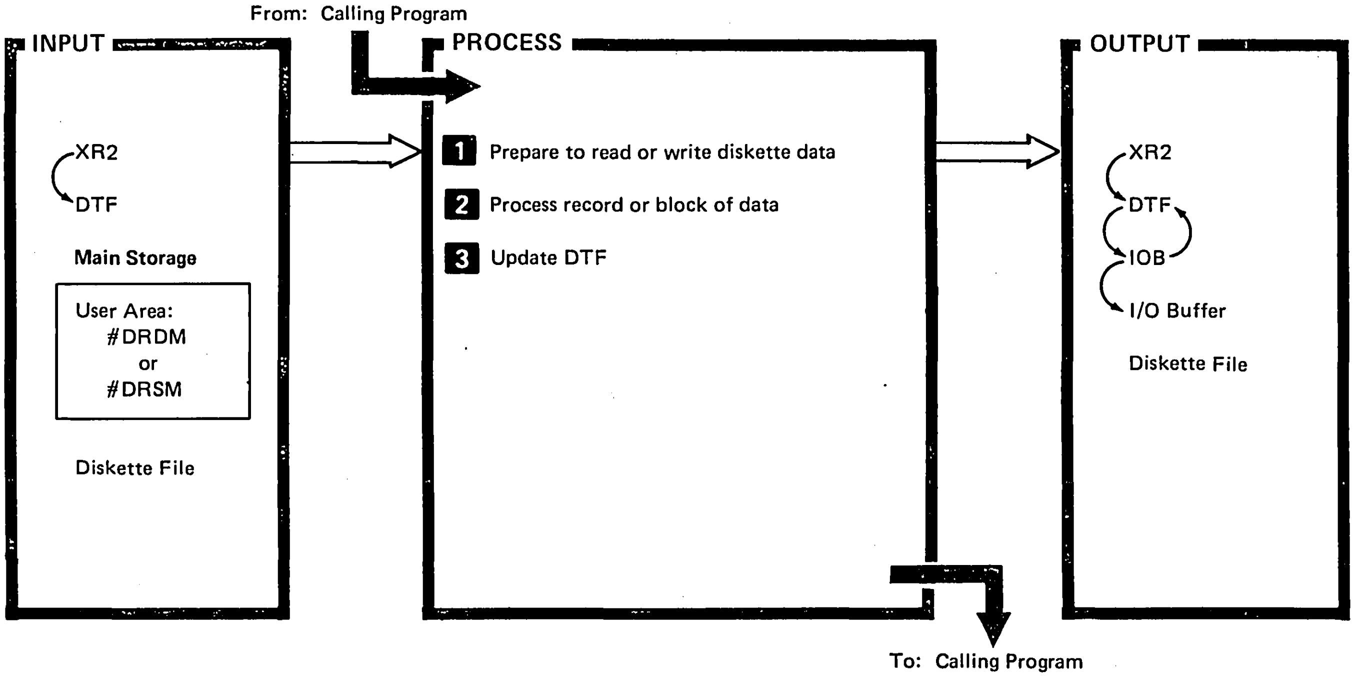
DESCRIPTION	MODULE/ ROUTINE
Call disk IOS to perform read/write operation.	SRGPI
Do sector deque for update.	Disk IOS
3 Set completion code in DTF.	Appropriate Driver
Update DTF pointer.	
Do sector deque for add operation.	
Return control to user program.	MAINDRV

Diagram 4.1 (Part 2 of 2). Perform Disk Data Management Functions



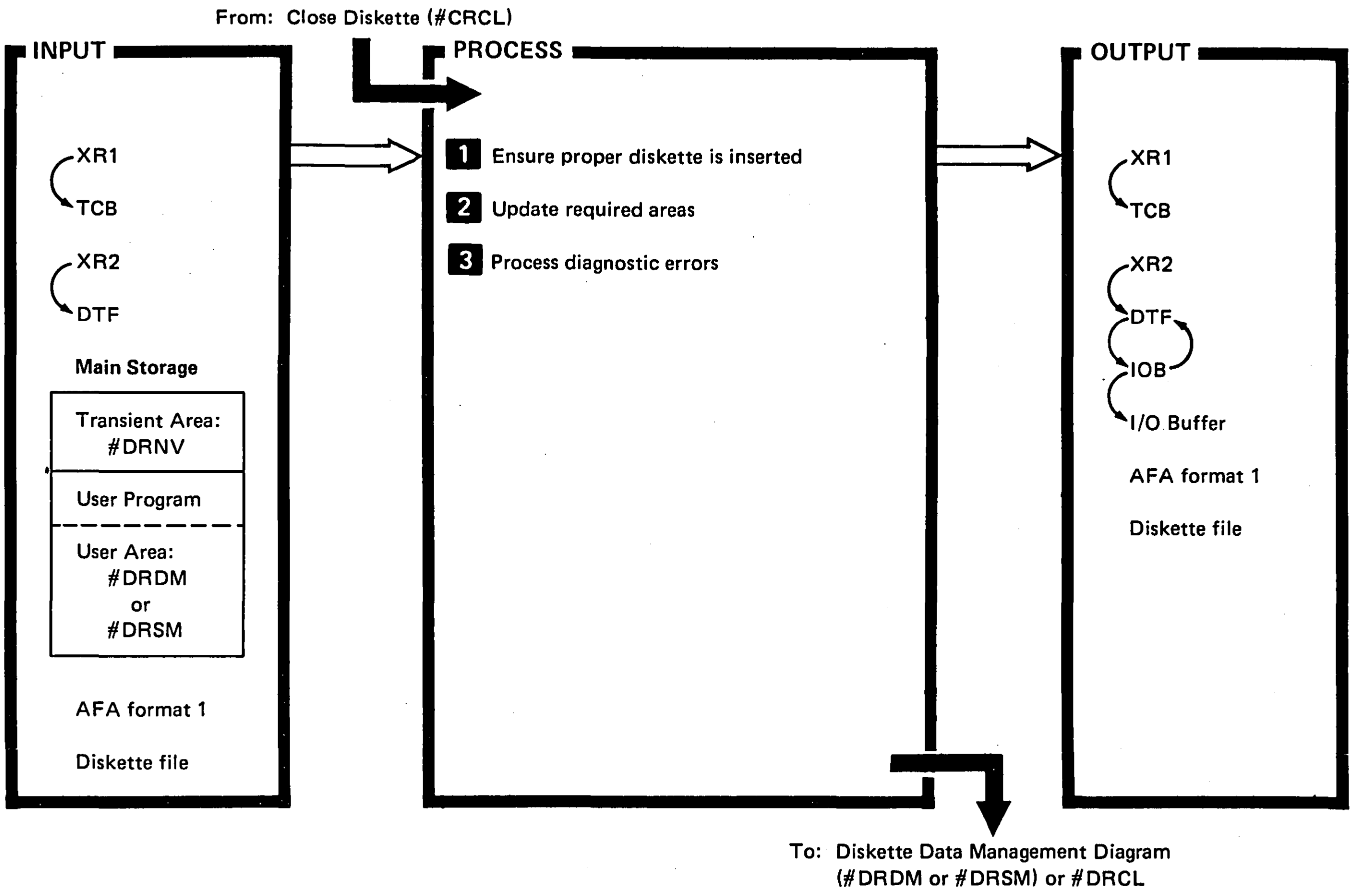
DESCRIPTION	MODULE/ ROUTINE
1 Execute initialization routine and prime buffer for input or add files. Update DTF record pointers for current record or block. Check for end of data: If end of input data, return to caller. **2** Read or write complete I/O buffers.	#DDSM
Issue SVC instruction for disk IOS services.	Disk IOS
3 Set appropriate completion code in DTF. Return control to user program.	#DDSM

Diagram 4.2. Perform Sectorized Disk Data Management Function



DESCRIPTION	MODULE/ ROUTINE
1 Execute initialization routine and prime buffer for input or add files. Update DTF record pointers for current record or block. Check for end of data: ● If end of input data, return to caller. ● If end of volume, set attribute bit and call close.	#DRDM or #DRSM
2 Block or deblock records between work buffer and I/O buffer (move mode). Locate input data or location for output data within I/O buffer (locate mode). Read or write complete I/O buffers.	
Issue SVC instruction for diskette IOS services.	Diskette IOS
3 Restore pointers and data areas after first I/O operation on full track request. Set appropriate completion code in DTF or initiate end of volume as requested. Return control to user program.	#DRDM or #DRSM

Diagram 4.3. Perform Diskette Data Management Functions



From: Close Diskette (#CRCL)

INPUT

XR1

TCB

XR2

DTF

Main Storage

Transient Area: #DRNV

User Program

User Area: #DRDM or #DRSM

AFA format 1

Diskette file

PROCESS

1 Ensure proper diskette is inserted

2 Update required areas

3 Process diagnostic errors

OUTPUT

XR1

TCB

XR2

DTF

IOB

I/O Buffer

AFA format 1

Diskette file

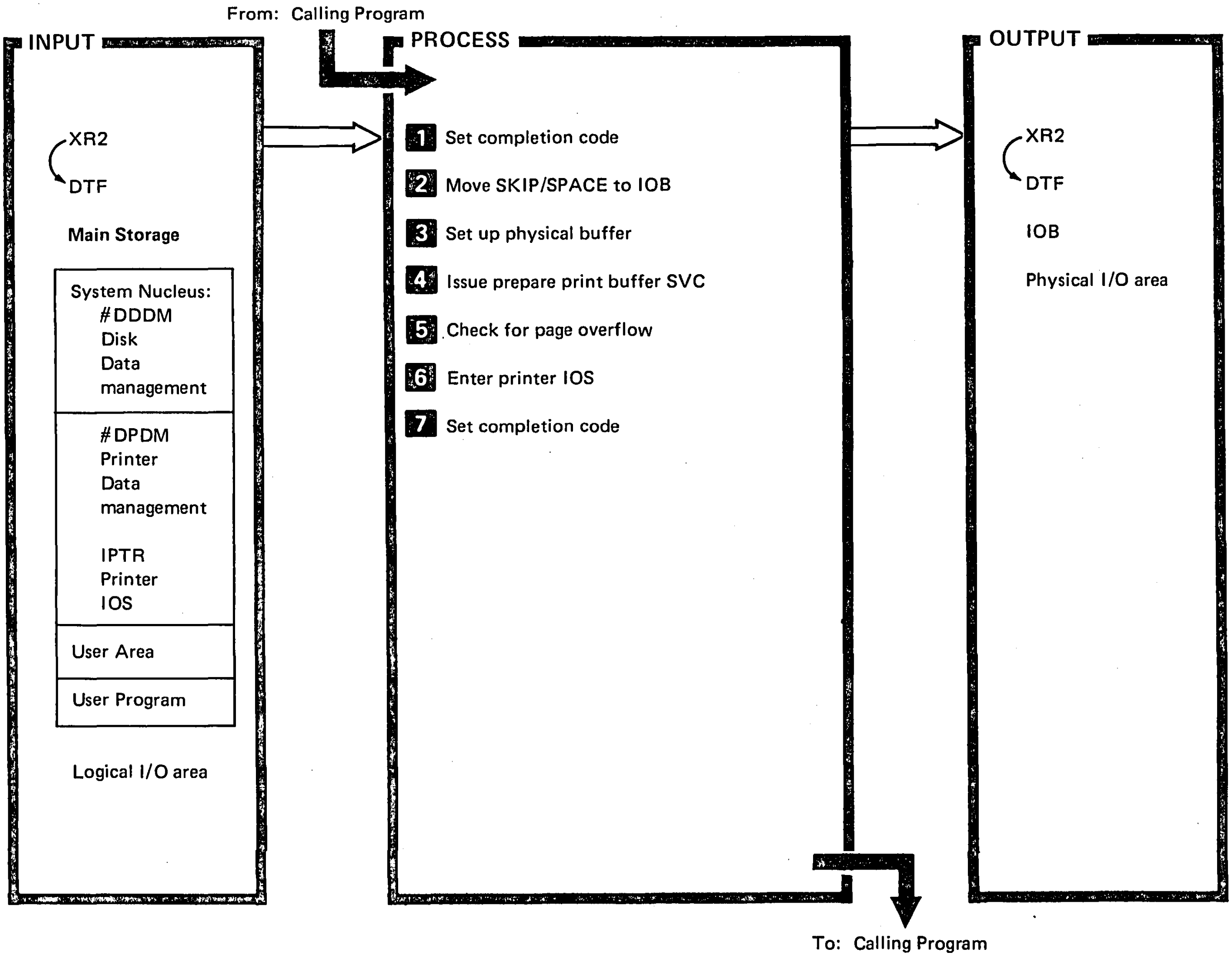
To: Diskette Data Management Diagram (#DRDM or #DRSM) or #DRCL

DESCRIPTION	MODULE/ ROUTINE
1 Issue task work area (TWA) request (SVC 51) to write one sector of I/O area to disk.	#DRNV
If output operation, write VTOC to diskette.	#CSVI
Return.	#DRVN
Issue insert next volume message.	#CLXS
Return.	#DRNV
Read VTOC of next volume.	#CSVI
Check volume I.D.	#DRNV
Setup I/O buffer to read diskette VTOC format 1's.	
Set up system date information for deleting expired files.	
Read diskette data set labels from VTOC.	#CSVI
If output file, ensure diskette contains no active files.	#DRNV

Diagram 4.4 (Part 1 of 2). Perform Diskette End of Volume Function (EOV)

DESCRIPTION	MODULE/ ROUTINE
Initialize multivolume indicator and enter volume number.	#DRNV
If input file, check volume sequence number, and if required, check creation date.	
2 Update DTF and IOB for new volume.	
Issue TWA request (SVC 51) to restore sector back to I/O area in core.	
Exit to diskette data management, or if an error occurred in flushing final buffer, return to diskette close.	
3 Process diagnostic errors at time they are discovered by calling SYSLOG routine (#CLXS).	

Diagram 4.4 (Part 2 of 2). Perform Diskette End of Volume Function (EOV)

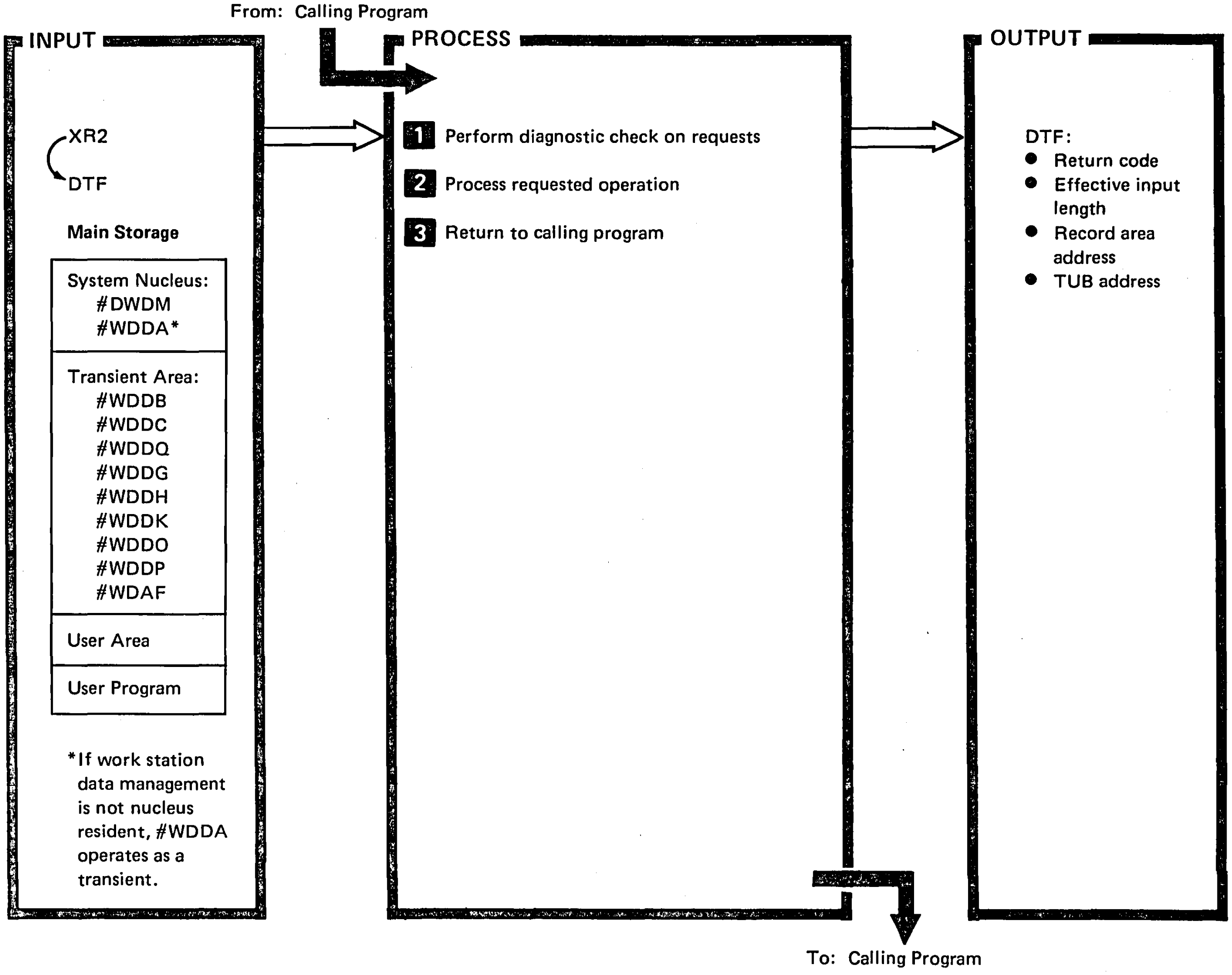


DESCRIPTION	MODULE/ ROUTINE
1 Set normal completion code (X'40') in DTF.	#DPDM
2 Move skip and space values from the DTF to the IOB.	
3 If print requested in DTF: • Move logical buffer into physical buffer. • Move data string length to IOB. • Set print indication in IOB control byte.	
4 If page alignment is requested, set up to issue forms alignment message.	#DPAL
Print the line.	WSIOCH
Return to user when 0 option is selected.	#DPAL

Diagram 4.5 (Part 1 of 2). Write Printer Data

DESCRIPTION	MODULE/ ROUTINE
5 Issue prepare print buffer SVC 26.	#DPDM
Move current line number from IOB to DTF.	
If page overflow has occurred, set overflow completion code in DTF.	
6 Print the line; or	WSIOCH
Write the line to the spool file.	#SPINT
Wait until contents of print buffer has been moved to printer.	#DPDM
7 If permanent error occurs, set permanent error completion code in DTF.	
Return to user.	

Diagram 4.5 (Part 2 of 2). Write Printer Data



From: Calling Program

INPUT

XR2 → DTF

Main Storage

System Nucleus: #DWDM #WDDA*
Transient Area: #WDDB #WDDC #WDDQ #WDDG #WDDH #WDDK #WDDO #WDDP #WDAF
User Area
User Program

*If work station data management is not nucleus resident, #WDDA operates as a transient.

PROCESS

1 Perform diagnostic check on requests

2 Process requested operation

3 Return to calling program

OUTPUT

DTF:
- Return code
- Effective input length
- Record area address
- TUB address

To: Calling Program

DESCRIPTION	MODULE/ ROUTINE
1 If status inquiry request:	#DWDM
● Set return code to reflect outstanding invites, op-ended invites, and STOP system in effect.	#WDDG
● Return to calling program.	
If accept input request:	#DWDM
● Wait 'any'.	#WDDG
● Read data into user's area.	WSIOCH
● Move user's parameter list back into DTF.	#WDDG
● Return to calling program.	
Call #WDDA.	#DWDM
If status inquiry op code:	#WDDA
● Set return code to reflect outstanding invites, op-ended invites, and STOP system in effect.	#WDDG
● Move user's parameter list back into DTF.	
● Return to calling program.	

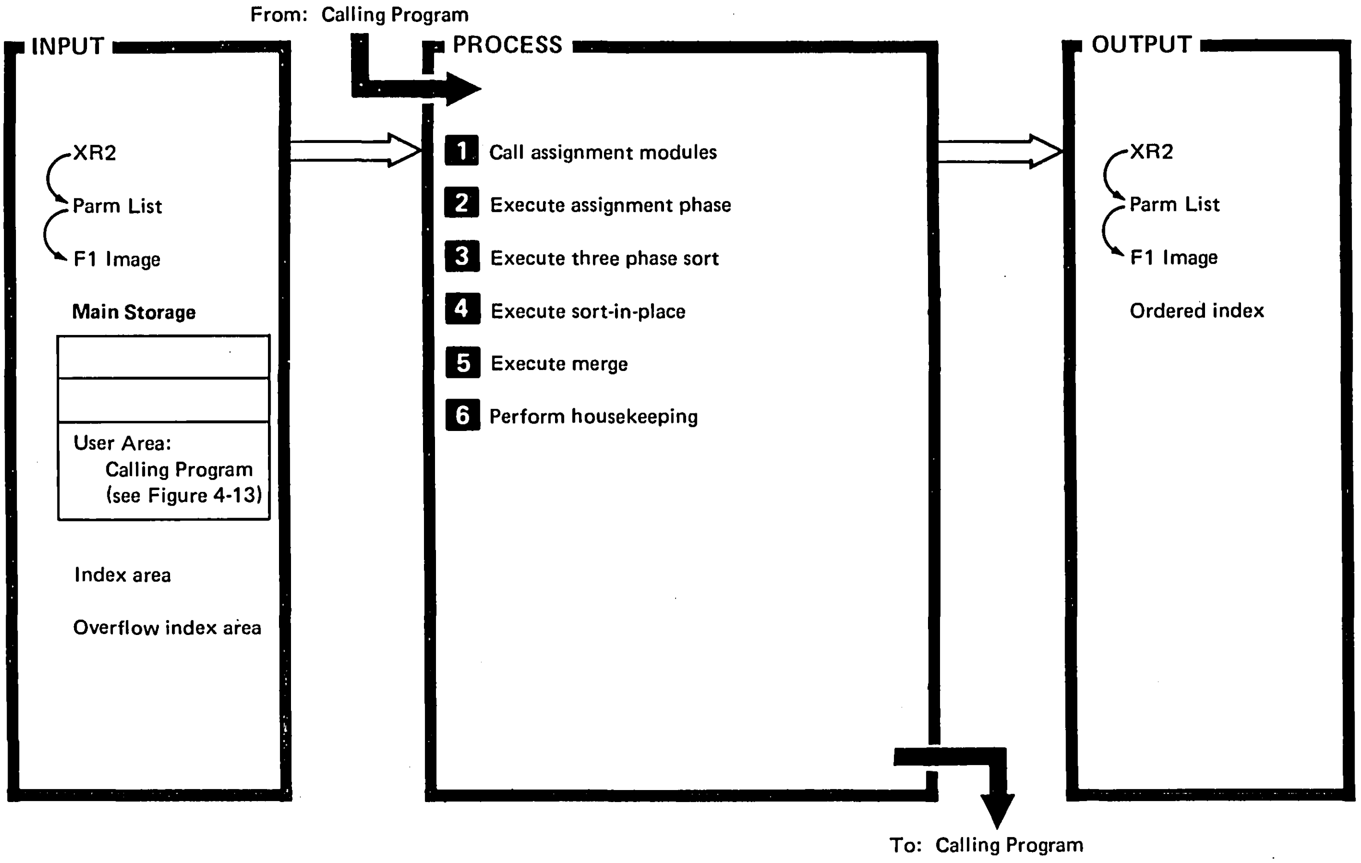
Diagram 4.6 (Part 1 of 3). Perform Work Station Data Management Function

DESCRIPTION	MODULE/ ROUTINE
2 Validate symbolic terminal name provided by user with operation.	#WDDA
If get attributes request:	#WDDG
● Determine if display or nondisplay.	#WDDQ
● Display screen size.	
● Determine attachment type.	
● Determine if online or offline.	
● Determine allocation status of work station.	
● Determine invite status.	
● Determine inquiry mode.	
● Go to **3**.	
If acquire op-code:	#WDDA
● Diagnose acquire terminal request.	#WDDQ
● Set work station ownership to requesting task.	
● If work station available, stop invites.	
● Go to **3**.	
Check that user owns requested work station.	#WDDA
If get operation:	
● Ensure that work station is online.	#WDDG
● Wait for IOB completion.	
● If function key, call #WDDH.	
● If Print key, call #WDDK:	#WDDH
– Determine printer to be used.	#WDDK
– Attach Print key task (#WDDP)	#SVAT
– Allocate buffer and read screen.	#WDDK
– Write data to swap area.	
– Post #WDDP:	
a. Wait for post from #WDDK.	#WDDP
b. Read data from disk.	FDIOS
c. Allocate printer; if not successful, call SYSLOG.	
d. Print screen.	#DPDM
e. Go to end of job.	#WDDP
– Call #WDDH to display error message stating that print was scheduled.	#WDDK
● If low-level Help key or other function key,	#WDDH
issue write error message.	WSIOCH
● If high-level Help key and if user owns work station, assign new TUB.	#WDDH
● Retrieve help text.	#MSGRE
● Issue message.	#WSIOCH
● If user, free new TUB.	#WDDH
● Call WSIOCH to read data.	#WDDG
● Read data into user's record area.	WSIOCH
If release request:	#WDDA
● If release single requester terminal (SRT) request, set off SRT release requester indicator in TUB.	#WDDG
● If release multiple requester terminal (MRT) request, set off waiting on MRTMAX.	
● If non-MRT release request, indicate RIB released by non-MRT.	
● Go to **3**.	
If terminal invited:	#WDDA
● Stop invite.	#WDDB
● If necessary, set cancel command in TUB.	
If stop invite op-code,	#WDDA
give return code.	#WDDB

Diagram 4.6 (Part 2 of 3). Perform Work Station Data Management Function

DESCRIPTION	MODULE/ ROUTINE
If special request, do one of the following:	#WDDA
● Roll request. ● Clear request. ● Reset request. ● Erase request. ● Restore request. ● Save request. ● If print request:	#WDDB
– Use user's DTF if possible.	#WDDC
– Push user.	#WDAF
▶ – Read screen into user area.	#WDDC ◀
– Print screen.	#DPDM
– Pull user.	#WDAF
If put request: ● Conditionally assign work station queue space for text stream and format.	#WDDA
● If assign fails: – If work station queue space is greater than text stream and format, unconditionally assign work station queue space. – If work station queue space is less than text stream and format, push user.	#WDAF
● Read format and build text stream: – Find format index entry corresponding to format. – Ensure that format entry is valid.	#WDDA
▶ ● Read field descriptor table and text from disk.	FDIOS ◀
● If put override, call #WDDO.	#WDDA
● Format output data. Process override of: ● Screen S specifications. ● Blinking display. ● Blinking cursor. ● Sound audible alarm. ● Reset keyboard. ● Insert cursor. ● Bypass field. ● Detail field definition D specification. ● Nondisplay field. ● Reverse image field. ● High intensity field. ● Blinking field. ● Underscore field. ● Output field. Process suppress invite indicator. If erase or put override not in operation: ● Process erase indicator. ● Process put override indicator.	#WDDA/ #WDDO
Scan for more output data.	#WDCP
Output data to work station.	WSIOCH
If user was pushed,	#WDDA
▶ pull user.	#WDAF ◀
3 Move user's parameter list back into DTF. Return control to calling program.	#WDDA/ #WDDQ/ #WDDG

Diagram 4.6 (Part 3 of 3). Perform Work Station Data Management Function



DESCRIPTION	MODULE/ ROUTINE
1 Call modules used during assignment phase (phase 0): ● Three phase Sort Control (#DDKLL). ● Set Preliminary Internal Values (#DDKAB). ● Design Sort (#DDKAC). ● Auto Allocate Work File (#DDKAD). ● End of Assignment Phase (#DDKAZ).	#DDKAA
2 Initialize common. Obtain index information from format 1 image.	#DDKAB
Design sort operation based on external parameters: ● Determine block size for input, work, and output areas. ● Determine size of record storage area (RSA). ● Determine number of records to fit in RSA. ● Determine order of merge for intermediate and final merge passes. ● Decide if work file is needed.	#DDKAC
If work file is needed, attempt to allocate space.	#DDKAD
If in debug mode, issue problem determination and debug information.	#DDKAZ
If sort not indicated, go to **5**. If work file not allocated, go to **4**.	#DDKAA

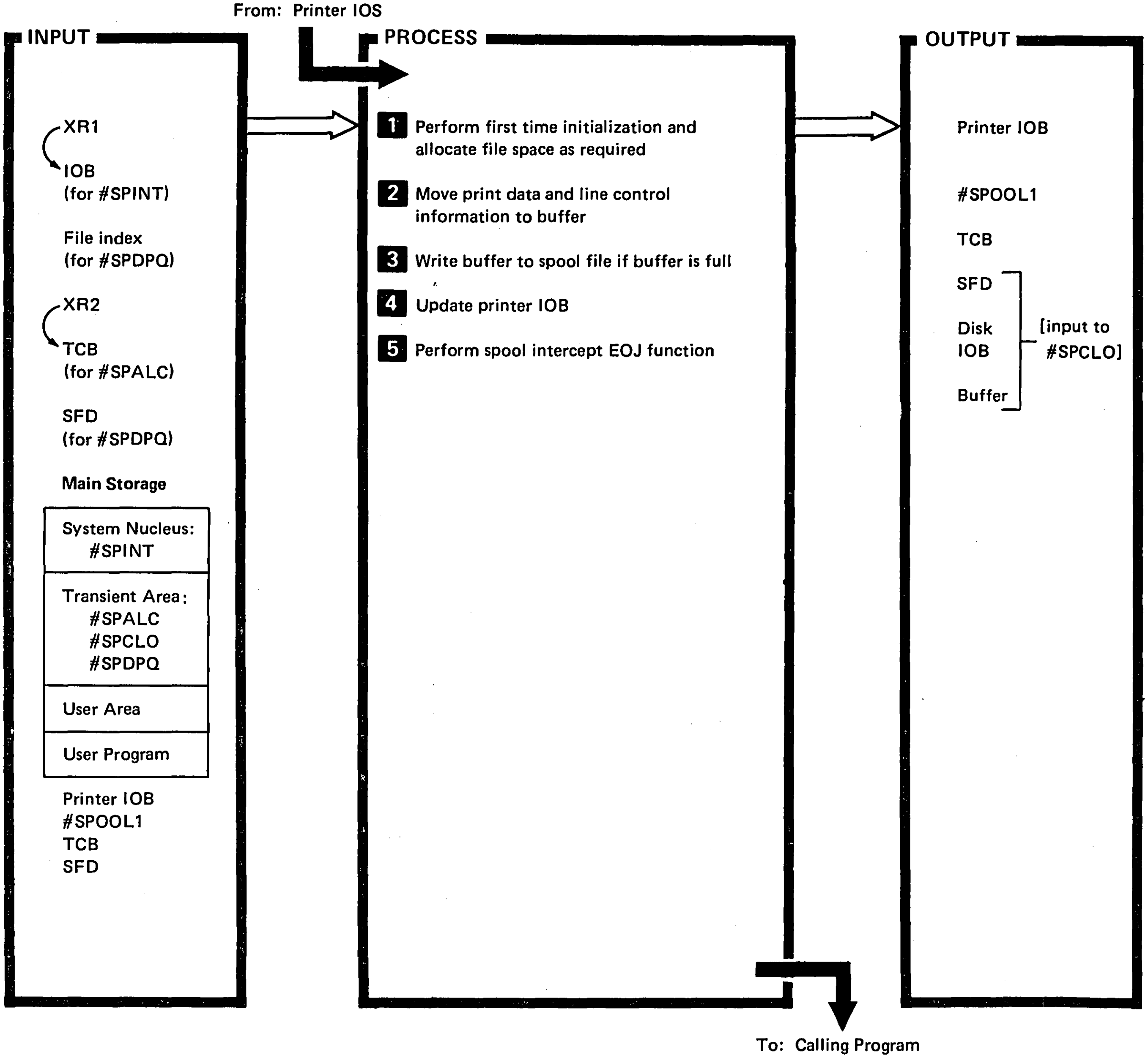
Diagram 4.7 (Part 1 of 3). Sort Index Entries for Indexed Disk File

DESCRIPTION	MODULE/ ROUTINE
3 Pass control to #DDK1A (phase 1).	#DDKLL
Indicate phase 1 entered.	#DDK1A
Pass control to #DDK1E.	
Allocate main storage for I/O buffers and record storage area (RSA).	#DDK1E
Read index entries.	#DDK1G and #DDK1R Disk IOS
Sort index entries into strings.	#DDK1E
Write intermediate strings of index entries to disk work file.	Disk IOS
Return.	#DDK1E
Determine validity of phase 1 execution.	#DDKEP
Determine if number of strings small enough to go to phase 3. If yes, skip phase 2.	#DDKLL
Indicate phase 2 entered.	#DDK2A
Pass control to #DDK2E (phase 2).	
Allocate main storage for I/O buffers.	#DDK2E
Read intermediate strings of index entries from work file.	#DDKWG Disk IOS
Merge strings of index entries together.	#DDK2E
Write intermediate strings of index entries to work file.	#DDKWP Disk IOS
Determine validity of phase 2 execution.	#DDKEP
Determine if number of strings small enough to go to phase 3. If no, repeat phase 2.	#DDKLL
Indicate phase 3 entered.	#DDK3A
Pass control to #DDK3E (phase 3).	
Allocate main storage space for I/O buffers.	#DDK3E
Read intermediate strings of index entries from work file.	#DDKWG Disk IOS
Merge strings of index entries together.	#DDK3E
▶ Write final sort output back to index.	#DDK3P and #DDK3W Disk IOS ◀
Go to 5 (to determine if merge is required).	#DDKAA

Diagram 4.7 (Part 2 of 3). Sort Index Entries for Indexed Disk File

DESCRIPTION	MODULE/ ROUTINE
4 Pass control to #DDKSS (Sort in Place).	#DDKAA
Assign main storage area for sort.	#DDKSS
Perform sort in place using quick-sort.	
Manage index segments entering record storage area (RSA).	
Call disk IOS to move index entries between disk storage and main storage.	Disk IOS
When sort is complete, determine if merge (#DDKMM) is required.	#DDKSS
If merge not required, go to 5.	#DDKAA
5 Reset sort, merge, and sort-in-place bits in format 1 image.	
Update last primary index entry point in format 1 image.	
If error return code, update parameter list.	
Return to calling program.	

Diagram 4.7 (Part 3 of 3). Sort Index Entries for Indexed Disk File



From: Printer IOS

INPUT

XR1 → IOB (for #SPINT)

File index (for #SPDPQ)

XR2 → TCB (for #SPALC)

SFD (for #SPDPQ)

Main Storage

System Nucleus: #SPINT
Transient Area: #SPALC #SPCLO #SPDPQ
User Area
User Program

Printer IOB
#SPOOL1
TCB
SFD

PROCESS

1 Perform first time initialization and allocate file space as required

2 Move print data and line control information to buffer

3 Write buffer to spool file if buffer is full

4 Update printer IOB

5 Perform spool intercept EOJ function

OUTPUT

Printer IOB

#SPOOL1

TCB

SFD, Disk IOB, Buffer [input to #SPCLO]

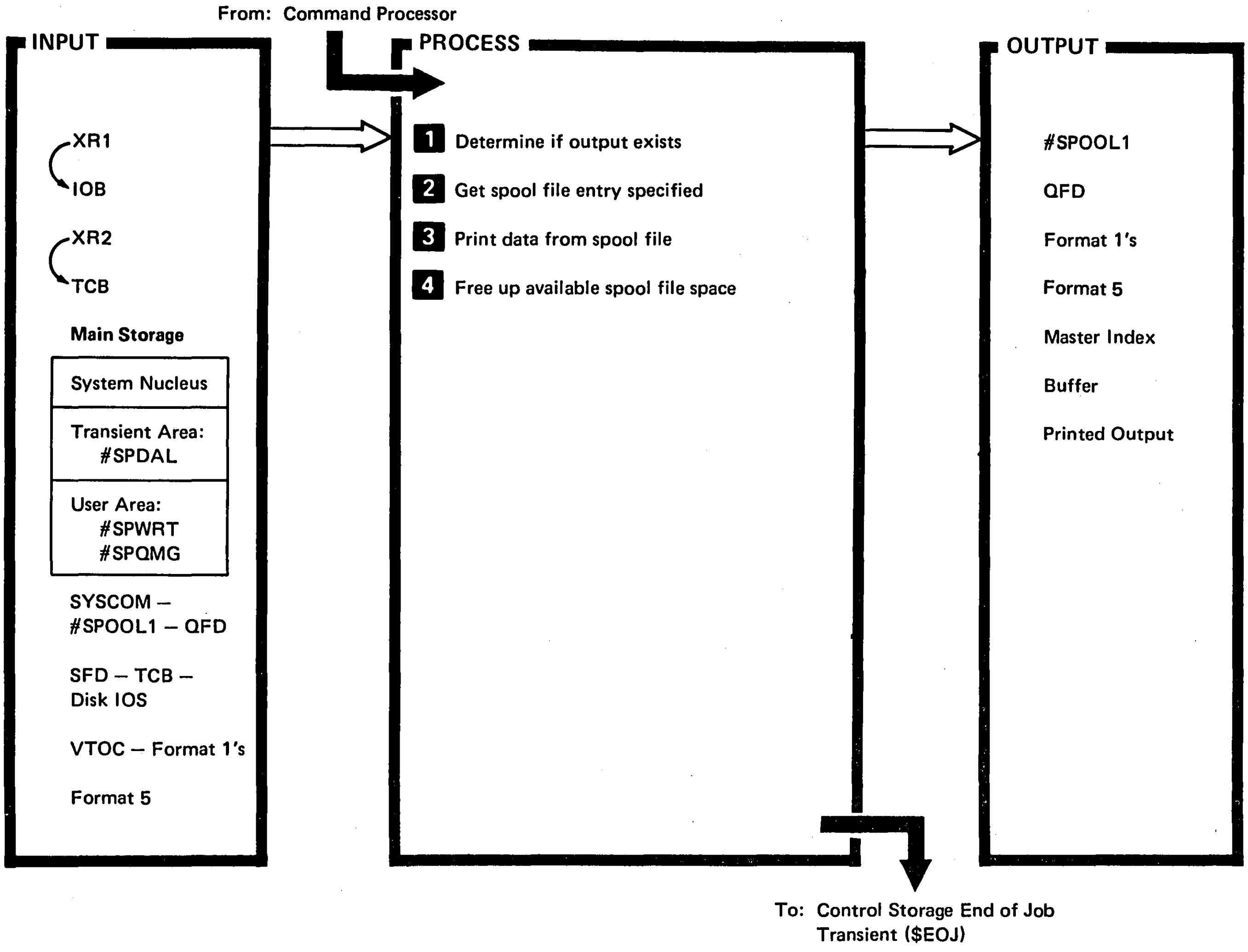
To: Calling Program

DESCRIPTION	MODULE/ ROUTINE
1 If first time call of this task, pass control to spool allocate (SPALC).	#SPINT
Check if first time for this task to call spool allocate. If first time, go to 1 A. If not first time, go to 1 B. A Assign and initialize task, spool file description (SFD), disk IOB, and buffer. Store SFD in task control block (TCB).	#SPALC

Diagram 4.8 (Part 1 of 2). Perform Spool Intercept Function

DESCRIPTION	MODULE/ ROUTINE
B Search spool file master index for available spool file space.	Disk IOS
• If block groups available in #SPOOL1-#SPOOL6, go to 1 E.	#SPALC
• If all extents are not allocated, go to 1 C.	
• Otherwise, issue SPOOL FILE FULL message with:	#CLXS
– Two option (control cancel) – call end of job routine (#CTEI).	#SPALC
– One option (retry) – try to find space again by going to 1 B.	
– Zero option (proceed) – call spool file close (#SPCLO), post writer, halt, and try again to find space by going to 1 B.	
C Attempt to allocate additional disk space for spool file.	#CAS1
D If no disk space available, issue NO DISK SPACE message and go to 1 C to try again.	#SPALC
E If first time, update data area sectors, master index, SFD, and disk IOB.	#SPDPQ
Chain index entry to queue.	Disk IOS
Return.	#SPDPQ
If not first time, update master index, SFD, and disk IOB.	#SPALC
Chain data-only index to queue.	Disk IOS
Return.	#SPALC
2 If buffer space available, move entire print record to buffer and go to 4.	#SPINT
3 Move as much of print record to buffer as space permits.	
Write buffer to spool file to free buffer space.	Disk IOS
If at end of spool block group, go to 1 B to allocate additional disk space.	#SPINT
Move remaining print record tu buffer.	
4 Mark intercepted print IOB complete (X'40').	#SPINT
Return to calling program.	
5 Call spool file close (#SPCLO). Terminator performs this function.	#CTEPR
Set end of file indicator.	#SPCLO
Write last record from buffer to disk.	Disk IOS
Update master index.	#SPCLO
If spool file close called by spool allocate, return to 1 B.	
Free SFD, disk IOB, and buffer.	
Return to calling program.	

Diagram 4.8 (Part 2 of 2). Perform Spool Intercept Function



From: Command Processor

INPUT

XR1 → IOB

XR2 → TCB

Main Storage

System Nucleus
Transient Area: #SPDAL
User Area: #SPWRT #SPQMG

SYSCOM – #SPOOL1 – QFD

SFD – TCB – Disk IOS

VTOC – Format 1's

Format 5

PROCESS

1 Determine if output exists

2 Get spool file entry specified

3 Print data from spool file

4 Free up available spool file space

OUTPUT

#SPOOL1

QFD

Format 1's

Format 5

Master Index

Buffer

Printed Output

To: Control Storage End of Job Transient ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Read master index to see if output exists.	#SPWRT Disk IOS
If queue is empty, call EOJ.	#SPWRT
If STOP PRT command issued, call end of job transient.	
Allocate printer if necessary.	#CAPT
Return.	#SPWRT
If printer not available, halt and allow options to either wait for the printer or cancel.	#CLXS
2 Set up queue file description (QFD) to indicate entry specified: ● First entry on queue. ● Next entry on queue. ● Stopped entry.	#SPWRT
Call spool queue manager (#SPQMG) to get entry specified.	#SPQMG

Diagram 4.9 (Part 1 of 2). Print Records From Spool File

DESCRIPTION	MODULE/ ROUTINE
Read master index to get address of queues using IOB provided.	Disk IOS
Based on QFD flag byte, search print queue for one of the following: ● First on queue (class). ● Next on queue (class). ● Stopped entry. If entry found: ● Place spool file address (SSSD) of entry in QFD. ● Return control and specific entry (in disk buffer) to spool writer (#SPWRT), go to 3. If specific entry not found: ● Set on end-of-file/not found indicator (QFDEF) in QFD. ● Return to spool writer (#SPWRT).	#SPQMG
● Call EOJ.	#SPWRT
3 Call spool file get.	#SPWRT
Retrieve first print line.	Disk IOS
Return.	#SPWRT
If forms change required, halt to allow operator to change forms.	#CSIM
Return.	#SPWRT
Issue separator page halt if first time or if forms change message was issued.	#CSIM
Return.	#SPWRT
Print separator pages if requested.	WSIOCH
Return.	#SPWRT
Perform page alignment if necessary.	#CLXS
Return.	#SPWRT
Fill 512-byte print buffer in system nucleus with print data.	Disk IOS
Call printer IOS.	#SPWRT
Print the line from the buffer.	WSIOCH
If multiple copies requested and entry printed out: ● Reset spool file description (SFD) to get first record in file. ● Return to start of 3.	#SPWRT
4 After all copies are printed, free up this entry and chain it from print queue to available queue.	Disk IOS
Return.	#SPWRT
Read spool file master index.	Disk IOS
If extent empty:	#SPWRT
● Update master index.	Disk IOS
● Return.	#SPWRT
● Free spool file extent.	#CAD1
Go to 1 to get next entry.	#SPWRT

Diagram 4.9 (Part 2 of 2). Print Records From Spool File

Program Organization

Figure 4-7 shows the main storage map for keysort.

Figures 4-8 through 4-14 show the control flow of functions used to run a job.

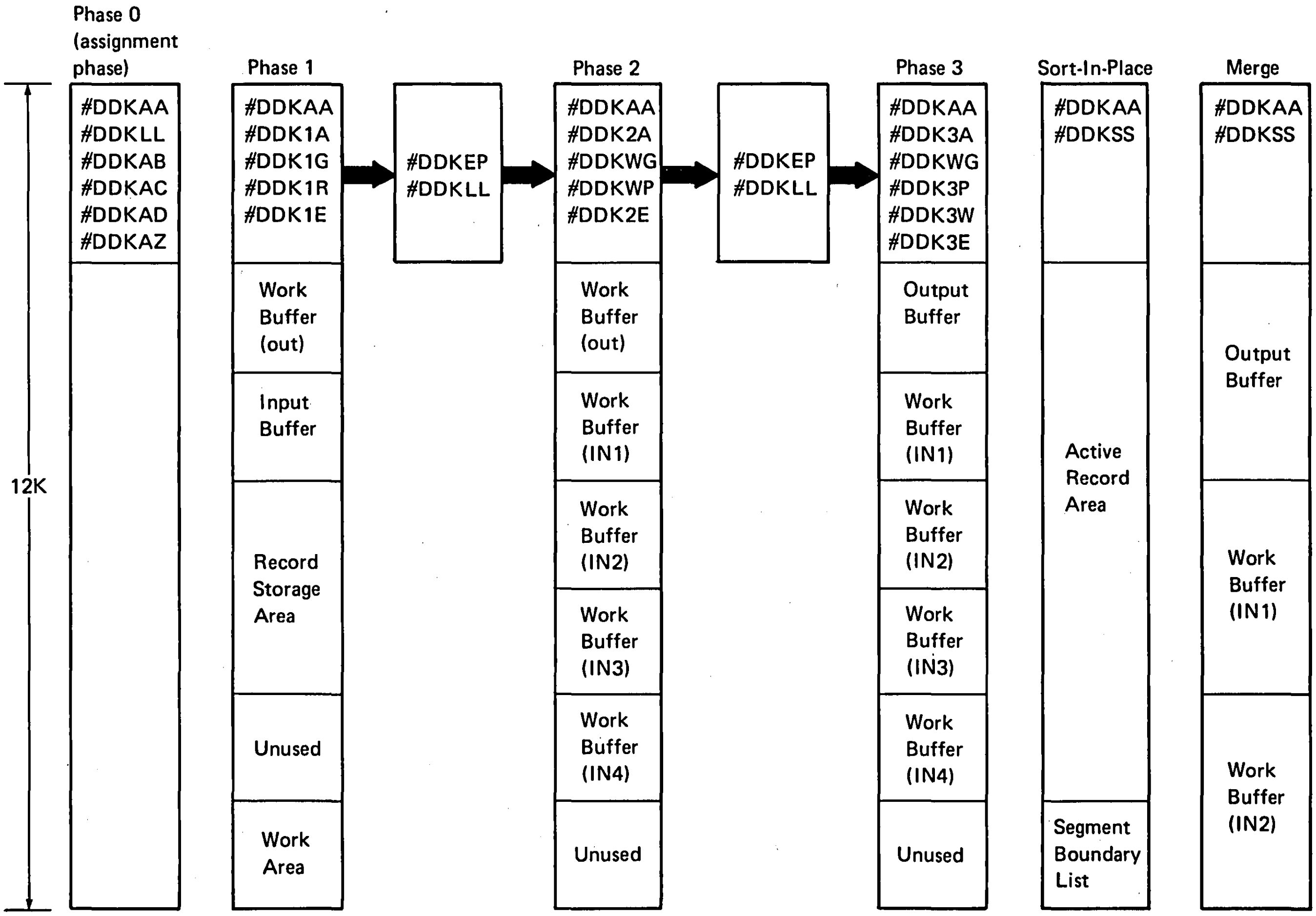


Figure 4-7. Keysort Main Storage Map

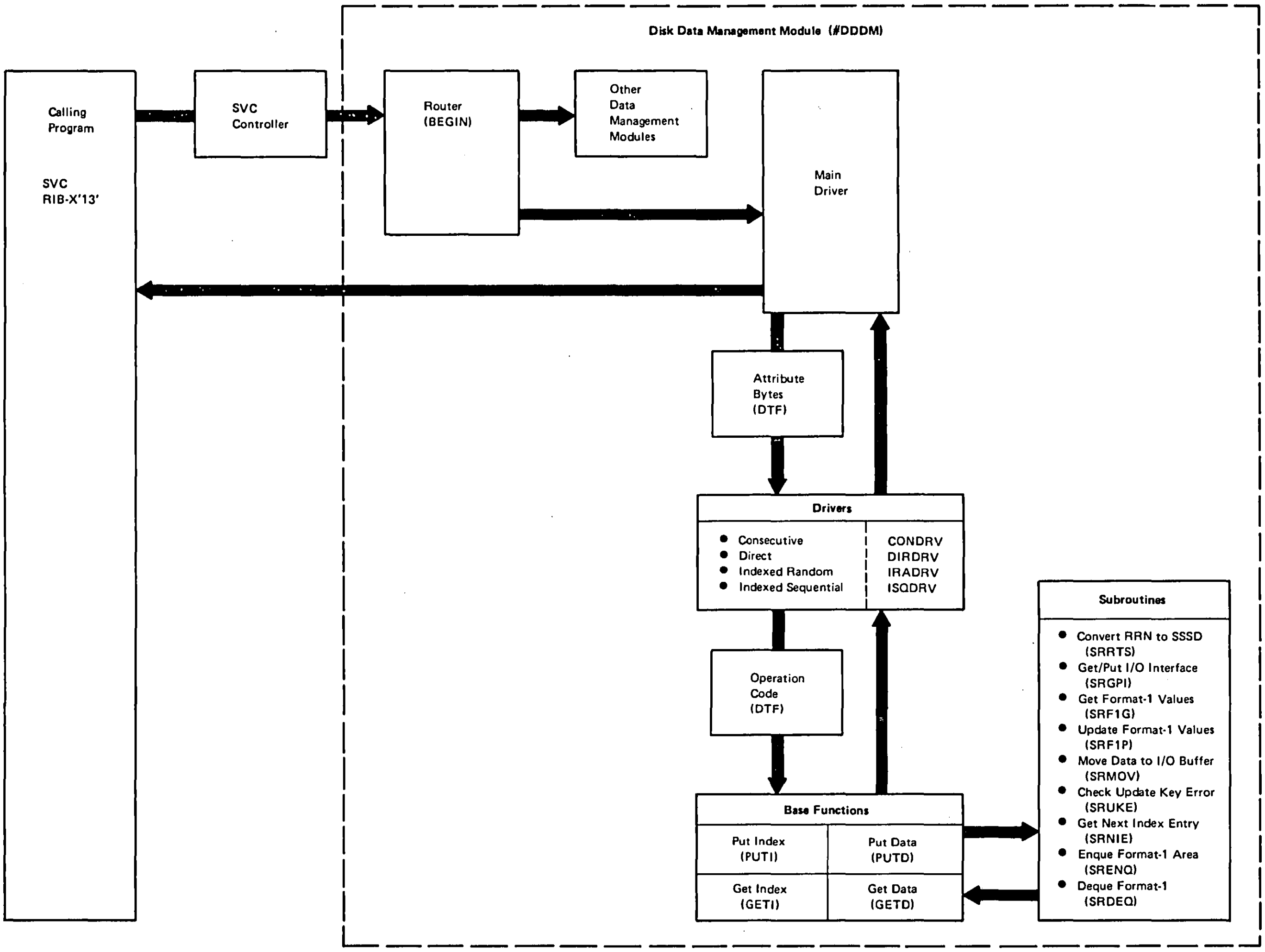

Disk Data Management Module (#DDDM)
Calling Program
SVC RIB-X'13'
SVC Controller
Router (BEGIN)
Other Data Management Modules
Main Driver
Attribute Bytes (DTF)
Drivers
● Consecutive
● Direct
● Indexed Random
● Indexed Sequential
CONDRV
DIRDRV
IRADRV
ISQDRV
Operation Code (DTF)
Base Functions
Put Index (PUTI)
Put Data (PUTD)
Get Index (GETI)
Get Data (GETD)
Subroutines
● Convert RRN to SSSD (SRRTS)
● Get/Put I/O Interface (SRGPI)
● Get Format-1 Values (SRF1G)
● Update Format-1 Values (SRF1P)
● Move Data to I/O Buffer (SRMOV)
● Check Update Key Error (SRUKE)
● Get Next Index Entry (SRNIE)
● Enque Format-1 Area (SRENQ)
● Deque Format-1 (SRDEQ)

Figure 4-8. Disk Data Management Control Flow

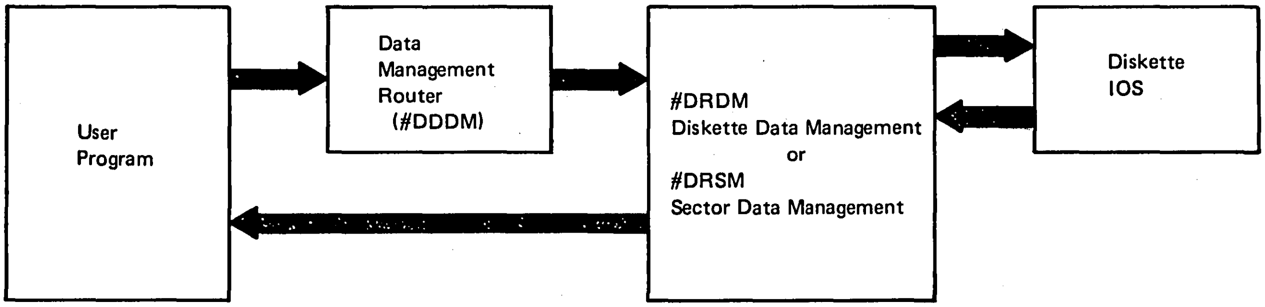


Figure 4-9. Diskette Data Management Control Flow

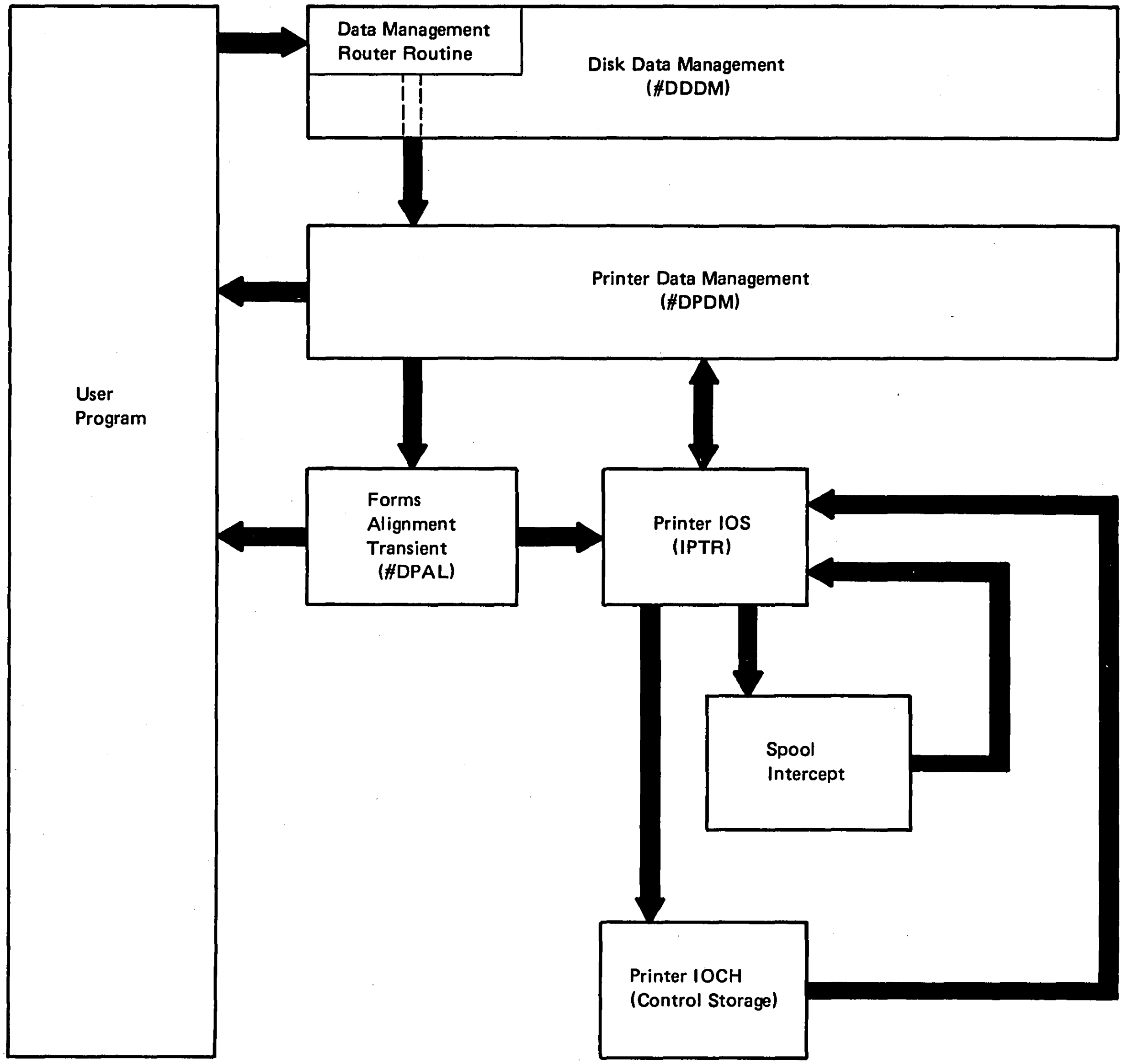

User Program
Data Management Router Routine
Disk Data Management (#DDDM)
Printer Data Management (#DPDM)
Forms Alignment Transient (#DPAL)
Printer IOS (IPTR)
Spool Intercept
Printer IOCH (Control Storage)

Figure 4-10. Printer Data Management Control Flow

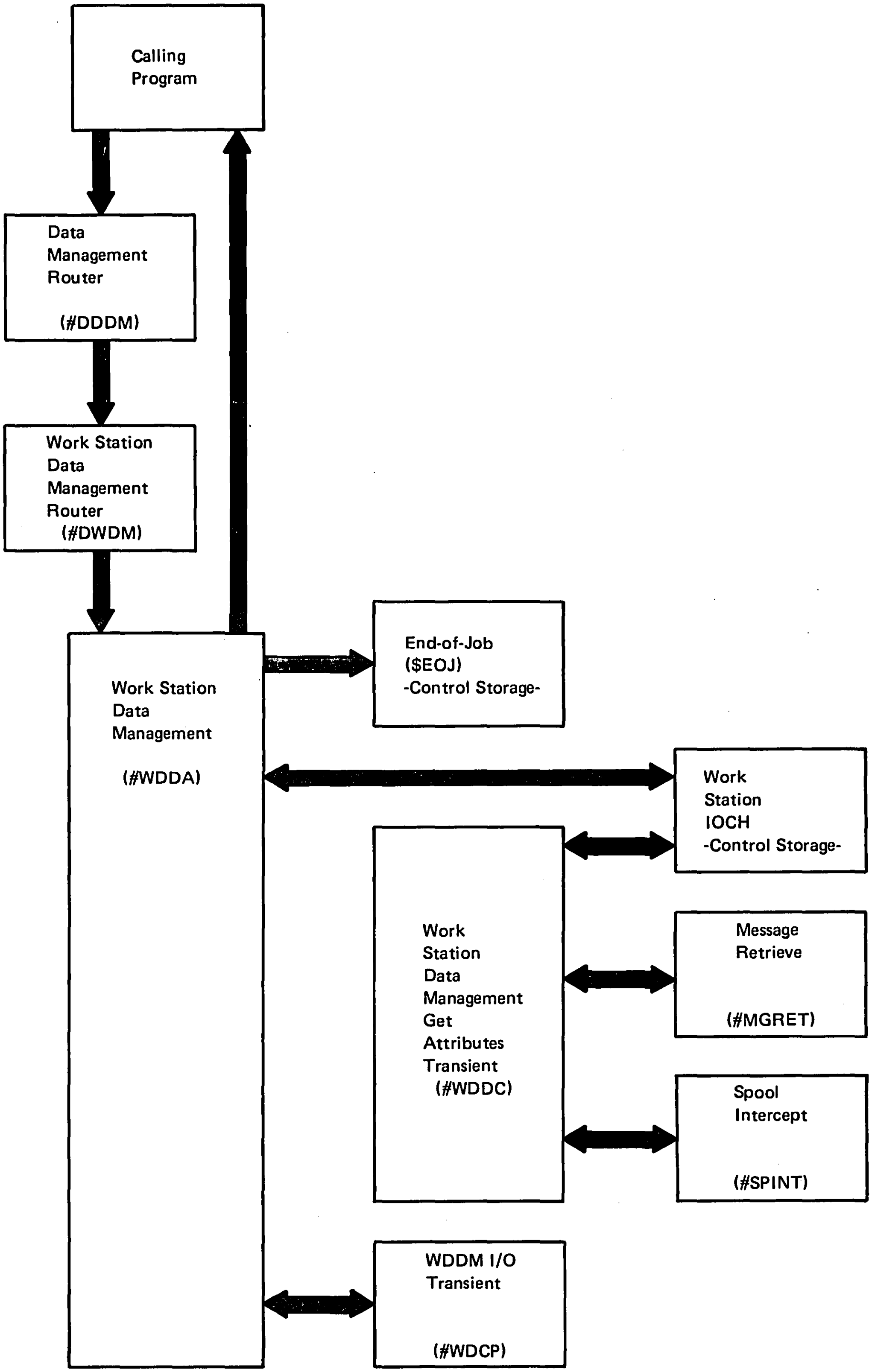

Calling Program
Data Management Router (#DDDM)
Work Station Data Management Router (#DWDM)
Work Station Data Management (#WDDA)
End-of-Job ($EOJ) -Control Storage-
Work Station IOCH -Control Storage-
Work Station Data Management Get Attributes Transient (#WDDC)
Message Retrieve (#MGRET)
Spool Intercept (#SPINT)
WDDM I/O Transient (#WDCP)

Figure 4-11. Work Station Data Management Control Flow

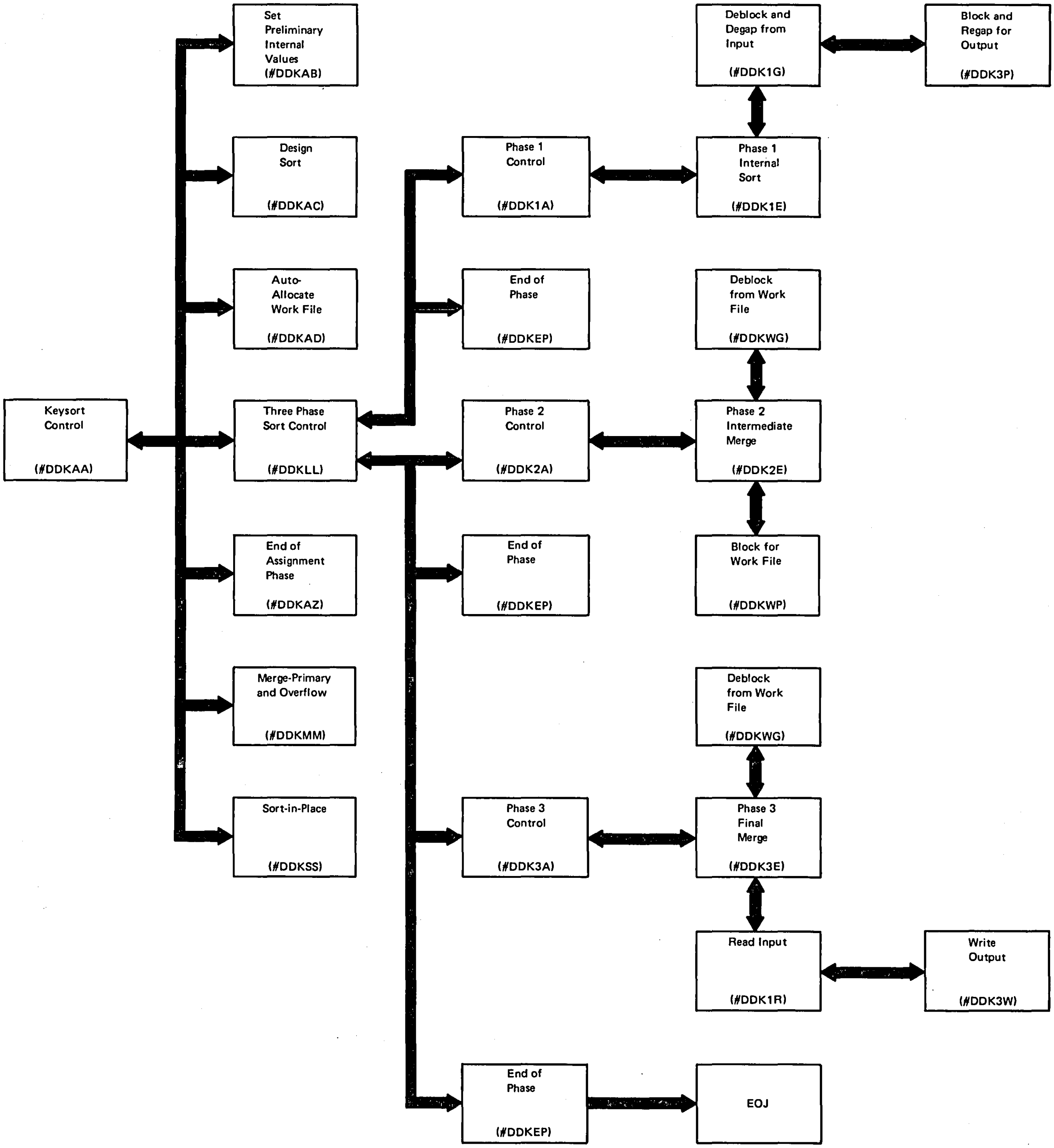

Set Preliminary Internal Values (#DDKAB)
Design Sort (#DDKAC)
Auto-Allocate Work File (#DDKAD)
Keysort Control (#DDKAA)
Three Phase Sort Control (#DDKLL)
End of Assignment Phase (#DDKAZ)
Merge-Primary and Overflow (#DDKMM)
Sort-in-Place (#DDKSS)
Phase 1 Control (#DDK1A)
End of Phase (#DDKEP)
Phase 2 Control (#DDK2A)
End of Phase (#DDKEP)
Phase 3 Control (#DDK3A)
End of Phase (#DDKEP)
Deblock and Degap from Input (#DDK1G)
Block and Regap for Output (#DDK3P)
Phase 1 Internal Sort (#DDK1E)
Deblock from Work File (#DDKWG)
Phase 2 Intermediate Merge (#DDK2E)
Block for Work File (#DDKWP)
Deblock from Work File (#DDKWG)
Phase 3 Final Merge (#DDK3E)
Read Input (#DDK1R)
Write Output (#DDK3W)
EOJ

Figure 4-12. Keysort Control Flow

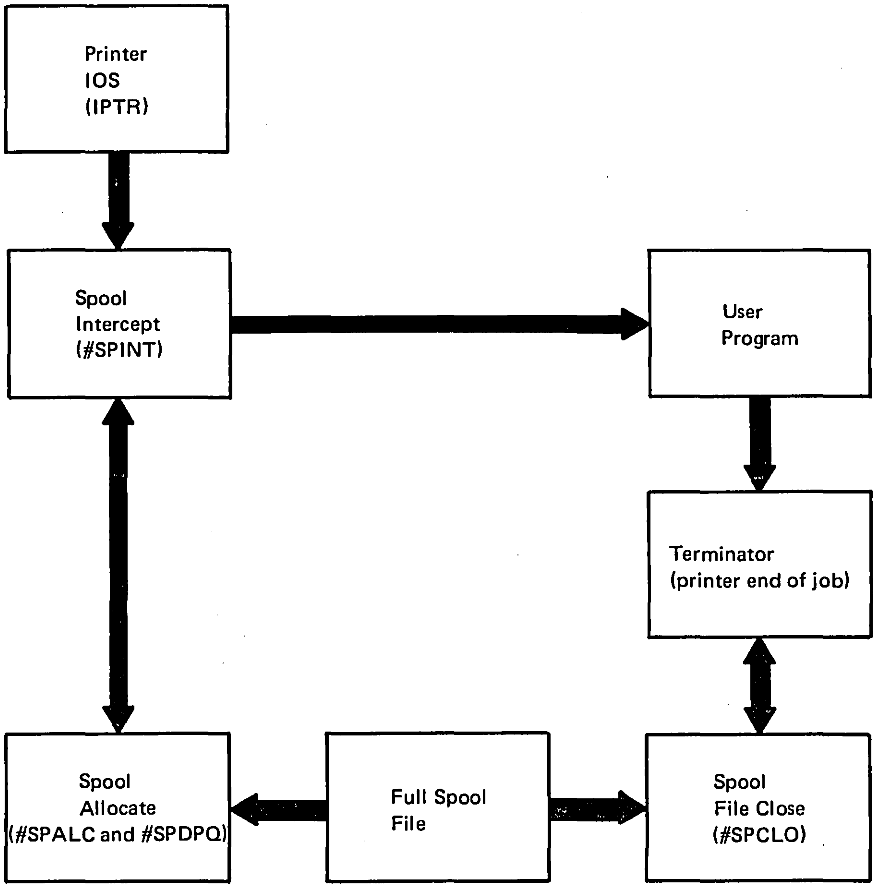


Figure 4-13. Spool Intercept Control Flow

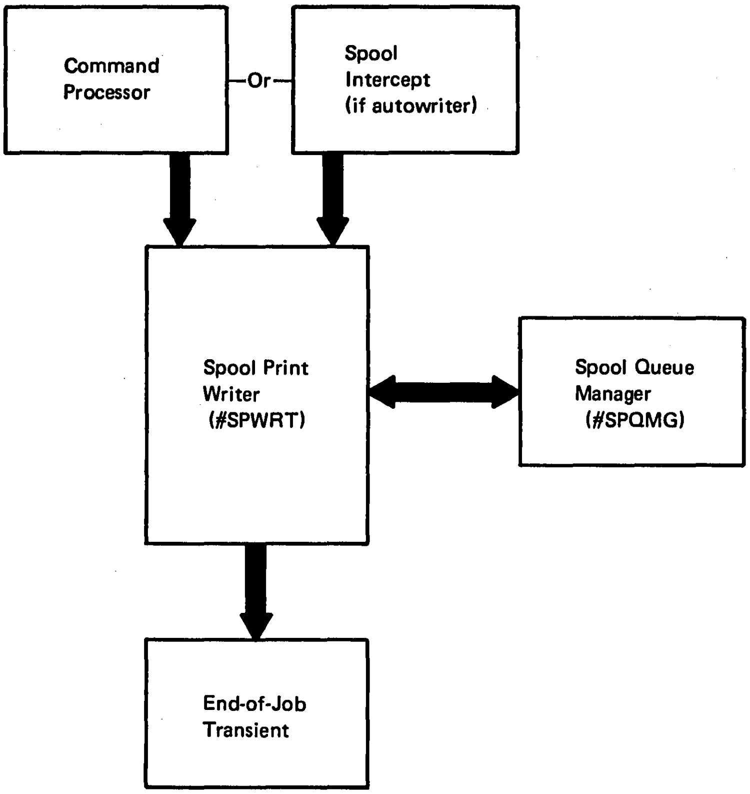

Command Processor
—Or—
Spool Intercept (if autowriter)
Spool Print Writer (#SPWRT)
Spool Queue Manager (#SPQMG)
End-of-Job Transient

Figure 4-14. Spool Print Writer Control Flow

Introduction

The functions that may be needed to terminate a job are:

- Close common
- Close printer
- Close diskette
- Close disk
- Step termination
- Job termination
- Abnormal termination

CLOSE

The purpose of close is to:

- Complete the processing of data in the output buffers.
- Extract data from DTF blocks so the data set label can be updated to reflect the current status of the file.
- Restore all opened DTFs to a preopen status.

The close function is performed by the common close transient (#DMCL) and the necessary device-oriented close transients:

Disk close	#DDCL
Diskette close	#DRCL

#DMCL removes the DTFs from the backward chain and then calls the appropriate device-oriented close transients.

All close modules are transients and reside as load modules in the system library. The close function is initiated by issuing an SVC with the close RIB (X'03') and with register 2 pointing to the DTF chain. The SVC processor will load the common close module (#DMCL) which, in turn, will call the appropriate device oriented close modules as needed.

To improve system performance, each of the device oriented modules are designed to be called only once for each DTF chain. Thus, each module will perform its functions for all required DTFs on the chain. It will search through the DTF chain to find all DTFs of its device type to be closed.

Common Close (#DMCL)

The common close transient (#DMCL) unchains the DTFs from the backward chain and then calls the appropriate device-oriented close transient. The disk close transient is called after all other DTFs are closed if a disk DTF is present.

Input to common close is a chain of postopen DTFs addressed by index register 2. XR2 contains the address of the first DTF to be closed.

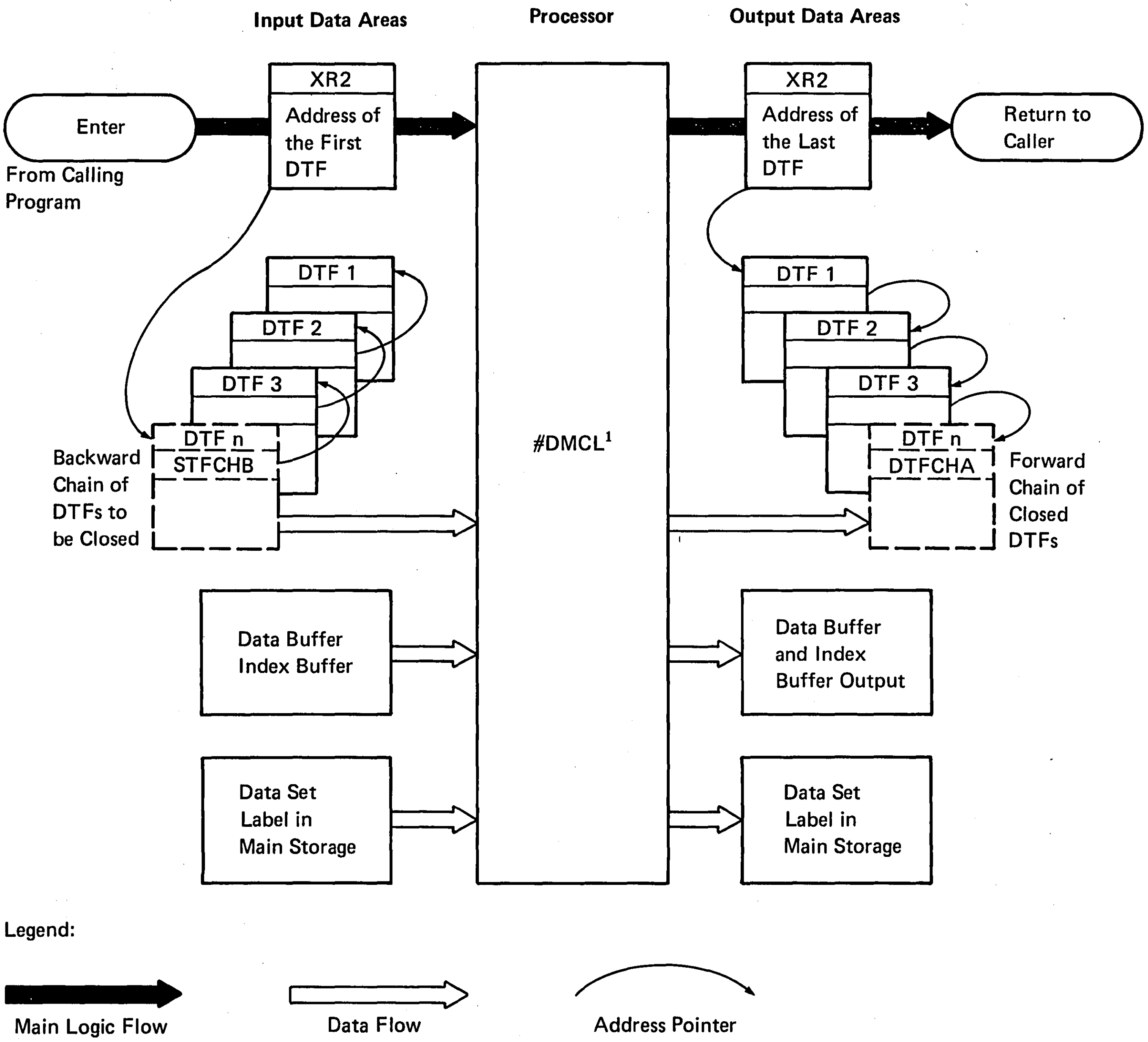
Output from common close is a preopen DTF for each file closed. Common close returns control to the calling program or to the disk close transient (see Figure 5-2).

Printer Close (within #DMCL)

The device oriented close function for the printers (within #DMCL) completes the processing of data in the print buffers, restores the DTF to a preopen status, and frees space for the printer IOB within the assign/free area. Control is passed to the next close transient or the calling program.

Work Station Close (within #DMCL)

The device-oriented close function for work stations (within #DMCL) indicates the work station file is closed by setting off the file allocated bit (X'04') in DTF attribute byte two ($DFAT2) and setting off the file opened bit (X'01') in DTF attribute byte two.



[1] Figure 5-2 shows the interrelationship of the transient close modules

Figure 5-1. Close Input and Output Data Areas

Binary Synchronous Communication Close (#BSCL)

When a binary synchronous communication (BSC) DTF is encountered in a call to common close, the following is performed:

- A call to close (X'22') is indicated in the BSC DTF operation code field, $DFOPC.
- A return code (X'00') is set in the communications specification block (CSB) return code filed CSBDRTNC.
- A task-to-task communication supervisor call is issued to indicate a close request to the BSC system task.

Control is then returned to common close (#DMCL) to close any other DTFs on the chain.

Diskette Close (#DRCL)

When a diskette DTF is encountered in a call to common close, the diskette close transient module is called. For output files, close updates the end of data pointers in the active format 1 image. The data set label is written into the diskette VTOC. The DTF is restored to its preopen state and may be allocated and opened again for further processing.

For input files, the data set label is rewritten in the diskette VTOC if the expiration date was changed.

Disk Close (#DDCL)

The device-oriented close function for disk completes the processing of data in the output buffers, updates the data set label (in main storage) to reflect the current status of the file, and restores the DTF to a preopen status. This transient must then loop through the entire DTF chain to locate any other disk DTFs that are open. Control is returned to the calling program when processing is complete.

TERMINATION

When a job step or job is ended, there is a considerable amount of cleanup activity to be performed before processing can continue. Termination performs these functions.

Step Termination

Step termination provides a means of terminating the current program at the end of a job step and preparing the system to accept another program for execution.

Step termination performs the following functions:

- Terminate the User Program
  Step termination gets control from the user program when the user program has finished executing. Since control is not returned, step termination ensures that all user files are closed. Termination is then loaded into the user area in main storage.
- Terminate the User Files
  All files used by the user program are processed. This may involve such operations as updating the disk VTOC and readying the format 1 for keysort.
- Reset the System
  Resources are freed up, pointers updated, switches updated, and necessary system data areas are reinitialized so the system is ready to accept another program for execution.
- Pass on Control
  Control is given to the initiator to enable the next step of the task to be processed.

Job Termination

Job termination is used to prepare the system to process new jobs.

Job termination performs the following functions:

- Terminate the User Files
  This involves such operations as updating the format 5 label and deleting the active format 1.
- Terminate the Task
  If required, the terminal associated with the task is freed. The task control block and job control block are then freed and reset respectively. Various pointers, switches, and necessary system data areas are reinitialized so the system is ready to execute more tasks.
- Pass on Control
  Control is given to the command processor to enable more tasks to be processed.

Abnormal Termination

Abnormal termination allows the user to stop processing at other than normal termination points.

Abnormal termination performs the following functions:

- Any remaining steps in the job are flushed.

- Files are not closed and, therefore, are left in one of the following conditions:
  - Shared files contain all updates or adds made prior to the abnormal termination.
  - Nonshared files contain all updates made prior to the abnormal termination.
  - Any adds made to nonshared files do not remain in the file (VTOC extents are not updated).
  - New files are removed from the VTOC.

Termination Interface

The termination interface transient (#CTEIF) is called in one of four ways (see Figure 5-3).

- Supervisor (abnormal termination with a dump request). The control storage end-of-job transient ($EJ1) calls #CTEIF.

- Command processor (abnormal termination due to a 2 or 3 option to an inquiry or CANCEL request). The command processor calls #CTEIF.

- User program (normal termination). The user program calls #CTEIF at step termination and job termination time.

- SYSLOG (3 option to halt).

Termination Interface (#CTEIF) is the main interface between the program requesting termination functions and the termination processor (#CTEPR). #CTEIF performs initial termination. Its primary job is to:

- Set termination indicators

- Call termination user interface (#CTEGU) to get user into main storage

- Call common close (#DMCL) to ensure user files are closed (normal termination)

- Wait for any active IOBs to complete

- Remove IOBs that have not started from the queue

- Call termination communication interface (#CTECM) if communication is supported

- Call the attach transient (#SVAU) if no communication interface is required

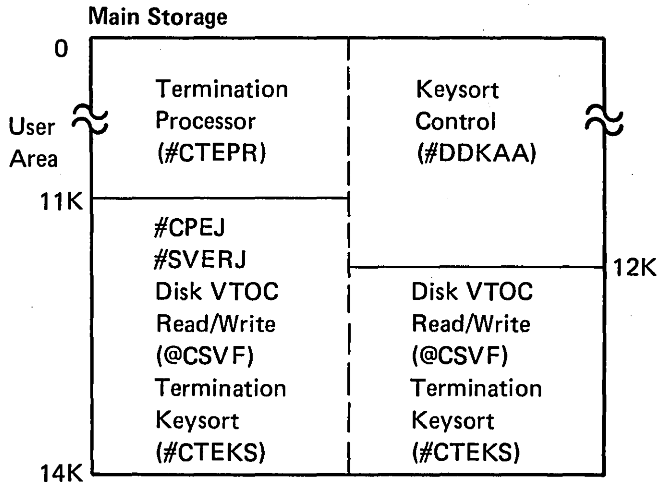
#SVAU loads and passes control to the termination processor (#CTEPR).

If a SYSLOG 2 option is taken, the user program terminates and continues on to the next job step.

Termination processor (#CTEPR) performs the three termination functions: step termination, job termination, and abnormal termination. #CTEPR is loaded (by #SVAU) into 14K of main storage starting at logical address X'C800'. The last 3K of #CTEPR contains the link edited modules, error recovery block cleanup for I/O SYSLOG (#SVERJ), command processor console SYSLOG EOJ message cleanup (#CPEJM), disk VTOC read/write (@CSVF) and termination keysort (#CTEKS).

The disk VTOC read/write routine is used by the termination processor to update the VTOC format 1's.

The termination keysort routine determines if keysort activity is required and, if needed, loads keysort control (#DDKAA) into main storage. #DDKAA is loaded over the first 12K of #CTEPR (#CTEKS and @CSVF are in the remaining 2K). When keysort completes, #CTEKS returns #CTEPR to its original 14K area. The left side of the following diagram illustrates main storage usage when termination is processing and keysort is not required. The right side illustrates main storage usage when termination is using keysort.


Main Storage
0
User Area
11K
14K
12K
Termination Processor (#CTEPR)
#CPEJ
#SVERJ
Disk VTOC Read/Write (@CSVF)
Termination Keysort (#CTEKS)
Keysort Control (#DDKAA)
Disk VTOC Read/Write (@CSVF)
Termination Keysort (#CTEKS)

The step termination function of #CTEPR is initiated at the end of each LOAD-RUN sequence within a procedure. Its primary job is to:

- Process files, utilizing disk and diskette VTOC read/ write and keysort control (#DDKAA) when needed
- Process library format 1's and file specification blocks
- Process source entry utility chain
- Performs clean up activity for spool function utilizing spool file close (#SPCLO)
- Free up allocated resources
- Free work station control blocks
- Release all data work stations for this task
- Reinitialize data areas as required
- RUN OXREF program (#MAXRF) as needed
- Clean up control storage SYSLOG halts (#SVERJ)
- Load the initiator (#CIML) into main storage

The job termination function of #CTEPR is initiated when the last step of a procedure completes processing or a load-run sequence completes processing outside a procedure. At job termination time, step terminate logic executes followed by job termination logic to:

- Process J type files
- Ensure work station control blocks are free
- Release all work stations for this task
- Free task work area control blocks
- Detach task control block
- Release user main storage
- Pass control to command processor

The abnormal termination function of #CTEPR is initiated when a 3 option is taken to a halt, the CANCEL command is received, or a program check with a dump request is encountered. The program requesting the abnormal termination function sets on the abnormal termination bit in its TCB. Abnormal termination executes step termination and job termination logic except that records added to files under certain conditions remain in the file. After the task is detached, the command processor is given control.

Termination User Interface (#CTEGU) is a main storage transient called by #CTEIF if the user program is pushed out of main storage and must be pulled back in.

Termination Communications Interface (#CTECM) is a main storage transient called by #CTEIF if a communications interface is required. Its main functions are:

- Call BSC to perform final cleanup for the user task
- Dequeue and free the communication specification blocks (CSBs)
- Call the attach transient (#SVAU) to load in, and pass control to, the termination processor (#CTEPR)

Method of Operation

This section contains functional diagrams for routines needed to terminate a job. They are:

- Close disk, diskette, printer, work station and data communications DTFs (Diagram 5.2)

- Termination function (Diagram 5.4)

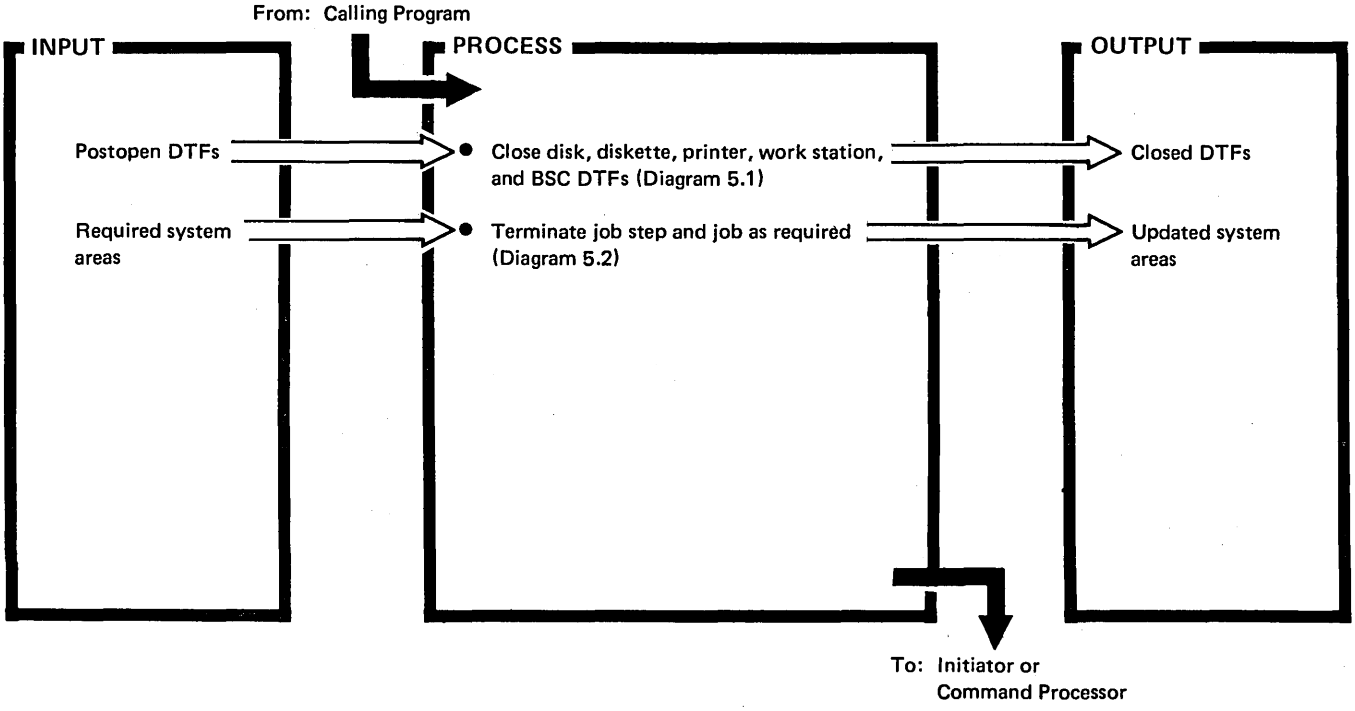
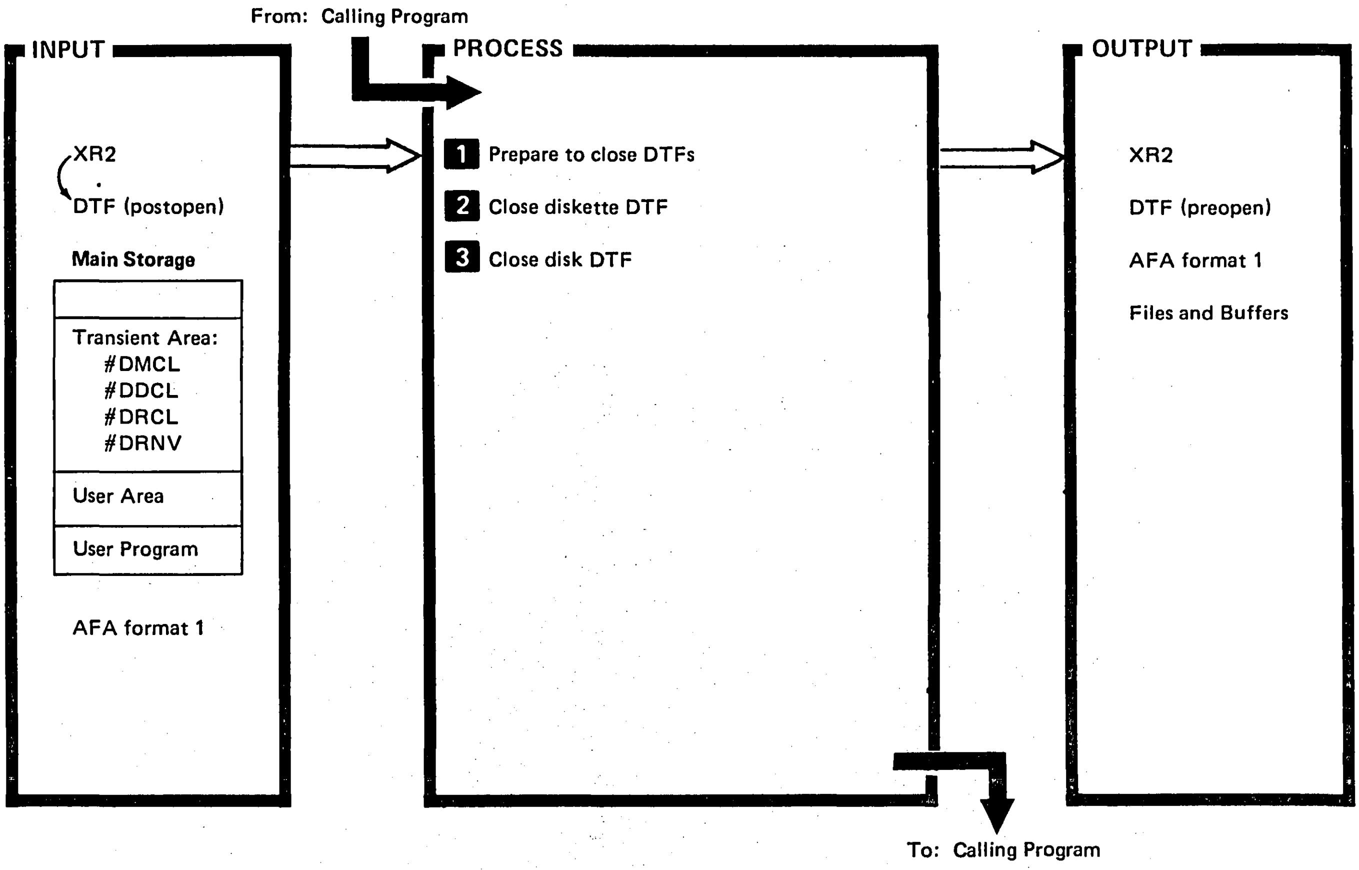

From: Calling Program
INPUT
PROCESS
OUTPUT
Postopen DTFs
Required system areas
Close disk, diskette, printer, work station, and BSC DTFs (Diagram 5.1)
Terminate job step and job as required (Diagram 5.2)
Closed DTFs
Updated system areas
To: Initiator or Command Processor

Diagram 5.0. Overview of Terminating a Job

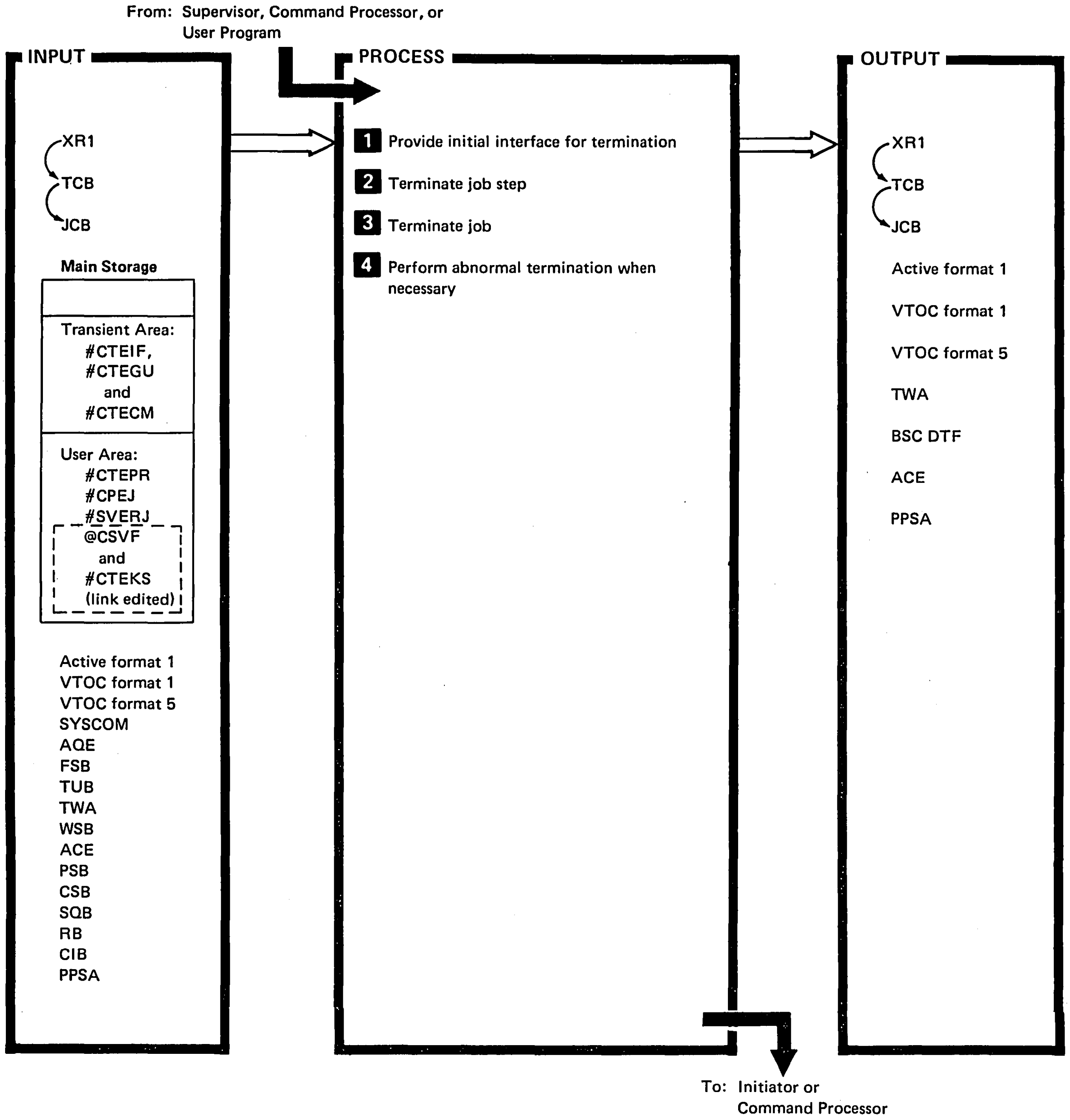


DESCRIPTION	MODULE/ ROUTINE
1 If diskette DTF, go to **2**.	#DMCL
Determine device type to unchain.	
Remove all DTFs from backward chain.	
A Determine device type to close.	
If work station DTF: ● Indicate DTF not allocated. ● Indicate DTF not open. ● Repeat steps until all work station DTFs are closed.	
If binary synchronous communication (BSC) DTF: ● Indicate a close call in BSC DTF operation code. ● Set return code in the communications specification block (CSB) return code field. ● Point XR1 at CSB. ● Issue task-to-task communication SVC to indicate a close call to the BSC system task. ● Repeat steps until all BSC DTFs are closed.	
If disk DTF, go to **3**.	

Diagram 5.1 (Part 1 of 2). Close Disk, Diskette, Printer, Work Station and Data Communications DTFs

DESCRIPTION	MODULE/ ROUTINE
Ⓑ If not last DTF on chain: ● Point to next DTF on chain. ● Return to Ⓐ. If printer DTF to close:	#DMCL
● Issue quiesce to printer to ensure completion of all I/O events.	WSIOCH
● Return DTF to preopen status. ● Free IOB space in assign free area (SVC 07). ● Return to 1 Ⓑ. If last DTF closed, return to caller.	#DMCL
2 Check file status to select appropriate close function: ● If input file or locate mode output file, go to Ⓔ. ● If final output move mode file, go to Ⓓ. ● If output error caused end of volume, continue with Ⓒ.	#DRCL
Ⓒ Issue permanent diskette output error message and initiate end of volume.	#CLXS
Ⓓ Call diskette IOS.	#DRCL
Write final output buffer to diskette if necessary.	Diskette IOS
If error, and error recovery can not be accomplished, go to Ⓒ. Ⓔ Update fields in active format 1 for last (or only) volume of file. Restore DTF to preopen status.	#DRCL
Write data set label into VTOC on diskette.	#CSVI
If end of volume, call open next diskette volume (#DRNV). If last (or only) volume, return to 1 to unchain diskette DTFs.	#DRCL
3 Close all disk DTFs:	#DDCL
● If data buffer must be written, write it to disk. ● If indexed access, flush index buffer.	Disk IOS
● Update format 1 in main storage. ● Restore DTF to preopen status. ● Repeat steps until all disk DTFs are closed. ● Return to calling program.	#DDCL

Diagram 5.1 (Part 2 of 2). Close Disk, Diskette, Printer, Work Station and Data Communications DTFs


From: Supervisor, Command Processor, or User Program
INPUT
XR1
TCB
JCB
Main Storage
Transient Area: #CTEIF, #CTEGU and #CTECM
User Area: #CTEPR #CPEJ #SVERJ
@CSVF and #CTEKS (link edited)
Active format 1
VTOC format 1
VTOC format 5
SYSCOM
AQE
FSB
TUB
TWA
WSB
ACE
PSB
CSB
SQB
RB
CIB
PPSA
PROCESS
1 Provide initial interface for termination
2 Terminate job step
3 Terminate job
4 Perform abnormal termination when necessary
OUTPUT
XR1
TCB
JCB
Active format 1
VTOC format 1
VTOC format 5
TWA
BSC DTF
ACE
PPSA
To: Initiator or Command Processor

DESCRIPTION	MODULE/ ROUTINE
1 Indicate in the task control block (TCB): ● Task is in termination. ● Task is not cancelable. ● Task is not inquirable.	#CTEIF

Diagram 5.2 (Part 1 of 3). Perform Termination Function

DESCRIPTION	MODULE/ ROUTINE
Pull the user program into main storage when required.	#CTEGU
Return.	#CTEIF
▶ Close all user files when required.	#DMCL ◀
Purge/quiesce ACEs from system queues.	#CTEIF
If communication systems in use: ● Terminate tasks as required. ● Dequeue and free communication specification blocks (CSB) (use control storage dequeue and free functions).	#CTECM
Load and pass control to termination processor (#CTEPR) (use attach transient).	#SVAU
2 For job step termination, determine if keysort required (#CTEKS link edited with #CTEPR). ● Disk file (not diskette). ● Indexed unordered load file. ● T, P, or J type file. ● Keysort bit on in format 1. ● Keysort not previously run for this task. If keysort is required: ● Update AFA format 1 to indicate keysort running. ● Load keysort control (#DDKAA) over first 12K of #CTEPR and pass control to it. ● After keysort completes, set off keysort running indicator in AFA format 1, reload #CTEPR, and pass control to it.	#CTEKS
● Return.	#CTEPR
Clean up console SYSLOG queue (#CPEJ link edited with #CTEPR). Perform error recovery block cleanup for I/O SYSLOG (#SVERJ link edited with #CTEPR).	#CPEJ
Process diskette files: ● Free active format 1 block (use control storage free function). ● Dequeue FSB from chain (use control storage free function). ● Free FSB in assign/free area (use control storage free function). Process disk files:	#CTEPR
● Update VTOC format 1's and format 5s as needed (use @CSVF – link edited with #CTEPR).	@CSVF
● Maintain latest date indicator for T and P type files. ● Dequeue active format 1's as required (use control storage dequeue function). ● Free active format 1 block in assign/free area if required (use control storage free function). ● Free high key bucket area in assign/free area for indexed files (use control storage free function). Process SEU member chain. Process work stations:	#CTEPR
● Release work stations (except requestors)	#DWDM
● Dequeue allocation queue element (AQE) for printer terminal unit blocks (TUB). ● Free work station specification blocks (WSB) (use control storage free function). Process printer specification blocks (PSB). Update job control block (JCB) as required. Reinitialize data areas as required. Deallocate all devices in device allocate table owned by task.	#CTEPR
If spool intercept being used, perform necessary spool cleanup.	#SPCLO

Diagram 5.2 (Part 2 of 3). Perform Termination Function

DESCRIPTION	MODULE/ ROUTINE
Return.	#CTEPR
Update spool file extents if necessary (SCA byte SCADCFG1).	#CTES
Return.	#CTEPR
Run cross reference resolver (#OXREF) program if needed.	#MAXRF
Load initiator mainline (#CIML) over #CTEPR in main storage and pass control to it.	#CTEPR
3 During normal job termination, perform the following:	
● Update format 5 and delete active format 1 for J type files.	Disk IOS
● Free control blocks for file FSBs and active format 1's for J type files.	#CTEPR
● Ensure control blocks for terminal WSB's and PSB's are free.	#DWDM
● Free task work area (TWA) control blocks.	#CTEPR
● Detach TCB, release users main storage, and reset swap area (control storage detach function.	#SVAU
● Release the requester's terminal.	#DWDM
● Free compiler information block (CIB) if necessary.	#CTEPR
● Pass control to command processor.	
4 If abnormal termination, perform job termination logic and:	
● For old nonshared disk files (P or T type):	
– Reset to zero, data area where new records added.	
– Put X'FF's in index overflow area where new record indexes were added.	
● For S type files, file remains on VTOC.	
Pass control to command processor.	

Diagram 5.2 (Part 3 of 3). Perform Termination Function

Program Organization

Figures 5.2 and 5.3 show the control flow required to terminate a job.

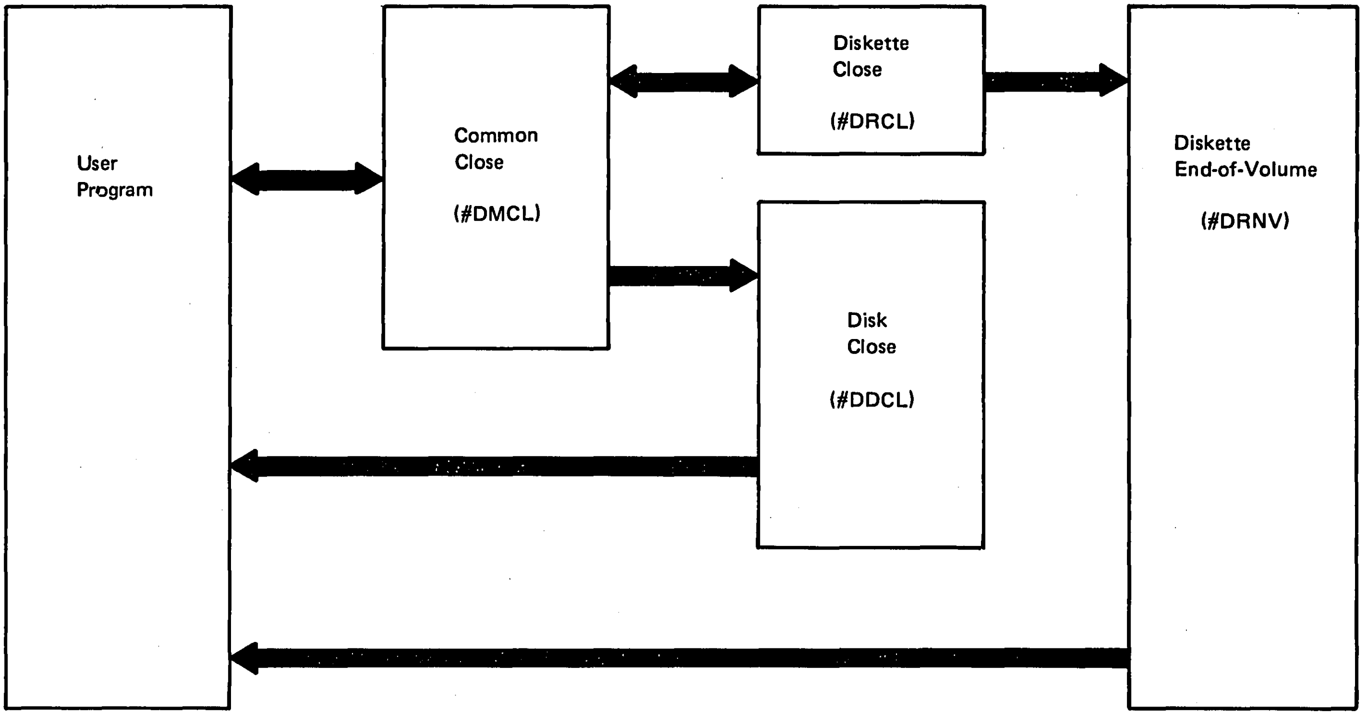


Figure 5-2. Close Control Flow

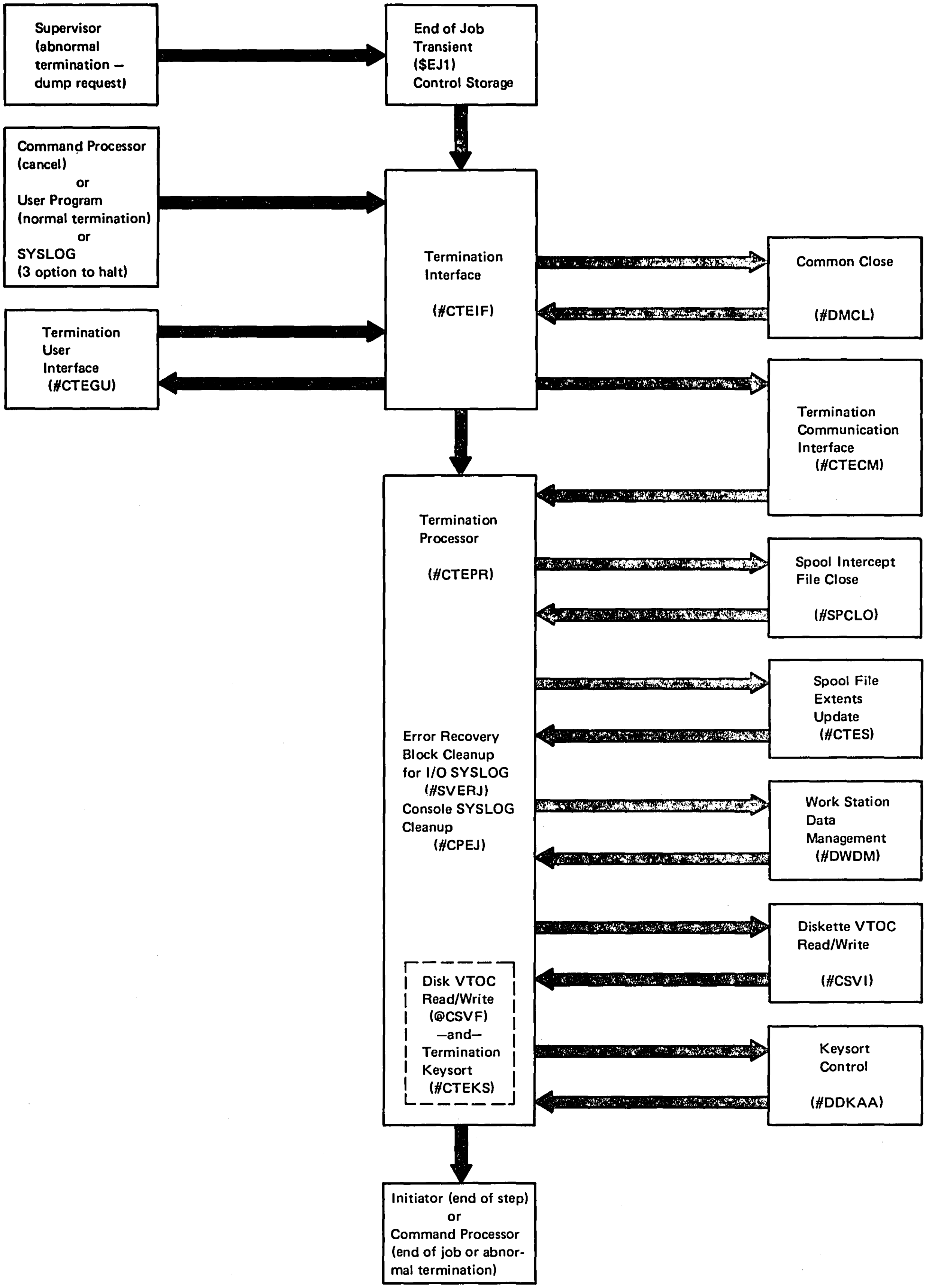

Supervisor (abnormal termination – dump request)
End of Job Transient ($EJ1) Control Storage
Command Processor (cancel) or User Program (normal termination) or SYSLOG (3 option to halt)
Termination Interface (#CTEIF)
Common Close (#DMCL)
Termination User Interface (#CTEGU)
Termination Communication Interface (#CTECM)
Termination Processor (#CTEPR)
Spool Intercept File Close (#SPCLO)
Spool File Extents Update (#CTES)
Error Recovery Block Cleanup for I/O SYSLOG (#SVERJ) Console SYSLOG Cleanup (#CPEJ)
Work Station Data Management (#DWDM)
Diskette VTOC Read/Write (#CSVI)
Disk VTOC Read/Write (@CSVF) —and— Termination Keysort (#CTEKS)
Keysort Control (#DDKAA)
Initiator (end of step) or Command Processor (end of job or abnormal termination)

Figure 5-3. Termination Control Flow

Part 3. Special Function Programs

Chapter 6. System Service Programs

Introduction

The system service functions are:

- Librarian facilities
- Library member protection
- Active format 1 area access routine
- Cross reference resolver
- Duplicate key display routine
- Print image verify routine
- Disk VTOC read/write
- Diskette VTOC read/write
- Message retrieve
- System input (SYSIN)
- System list (SYSLIST)
- System log (SYSLOG)
- History file put
- Supervisor task attach
- Supervisor task detach
- Syntax checker
- Information Retrieval
- Data management task transfer control
- Snap dump

LIBRARIAN FACILITIES

System programs and user programs are stored in a manner that makes them readily accessible to the system user. The programs are stored in libraries on disk and are called library members. Library members can be executable load modules (O-modules), procedures (P-modules), subroutines (R-modules), and source statements (S-modules).

The librarian facilities provide a way to locate and access library members. The librarian facility programs are:

- Find a library routine (#MAFLB)
- Single name find routine (#MASFN)
- Librarian find routine ($MAFND)
- Source library get routine (#MASYG) or (#MASYL)
- Library record put routine ($MAPUR)
- Library sector get/put routine ($MAPGS)

Find a Library Routine

The find-a-library transient (#MAFLB) finds a library by name. Given an 8-byte library name, it returns the 2-byte address of the format 1 for the named library. Zero is returned if the library does not exist. The library format 1 address must be passed to librarian access routines when accessing members in libraries other than the system library or the designated user library.

#MAFLB finds or builds the format 1 for the requested library in the active format 1 area (AFA). #MAFLB does not enqueue the requested library. However, if the requested library is not being used by the calling task, #MAFLB does chain the requested library's format 1 to the caller's job control block (JCB) with a library file specification block (FSB) and increment the use count in the format 1. This ensures the existence of the requested library for the duration of the job step.

XR2 must contain the address of the leftmost byte of a 10-byte parameter list (see Figure 6-27). The first 8 bytes must contain the library name. On input, the last 2 bytes must be zero, or must contain the address of the JCB to be used by the command processor when building the JCBs.

Single Name Find Routine

The single name find transient (#MASFN) finds a specified library member by searching first in a user library, then in the system library. The user library may be the designated user library (from the LIBRARY statement) or another user library. The search may be restricted to only the user library or only the system library.

On a regular call, #MASFN returns 17 bytes of the directory entry (bytes 10 through 26) and indicates if the member was found in the user library or the system library. When #MASFN is called with a request to build the loader parameter list, the first 10 bytes of the loader parameter list are returned. The user must set the last 2 bytes (the load address) of the loader parameter list. The load address can be set with the system find parameter list macro ($FNDP).

#MASFN enqueues the library directory for the duration of the search.

When the requested member is not found, #MASFN returns the parameter list unchanged or issues an error message and terminates if building the loader parameter list.

The parameter list is 18 bytes for a regular call or 12 bytes for a loader call. XR2 must point to the leftmost byte.

If the system find include version (@MASFN) is link-edited to another module, the load request is not supported. The link-edited module must have addressability to the nucleus and must provide a buffer.

Librarian Find Routine

The librarian find routine ($MAFND) locates directory entries by full or partial name. The caller provides a buffer, specifies the library to search, gives the member type or types, and gives the member name or partial name and length. On each call, $MAFND returns the address in the buffer of the next directory entry meeting the criteria or indicates that no more members meet the specified criteria. The caller may update the found directory entry and then have $MAFND write the updated directory entry back to disk.

The library to search is specified by giving the library format 1 address (returned by #MAFLB) or by giving zero and indicating the designated user library (library from the LIBRARY statement) or the system library or both. If both are given, the designated user library is searched first for any member meeting the specified criteria. Only if no members are found is the system library searched. The searched return indicator is set to indicate if the member was found in the designated user library or the system library. $MAFND enqueues the library directory on the first call and dequeues it on the last call.

$MAFND may be executed as a transient version ($MALFN) or loaded into the caller's area. XR2 must contain the address of the leftmost byte of a 24-byte parameter list (see Figure 6-30). The transient version requires a 25-byte work area following the parameter list.

Source Library Get Routine

The source library get transient (#MASGT) retrieves source or procedure members from a library one record per call. If requested, #MASGT finds the member and updates the parameter list for a get or returns a not found indicator. The caller specifies the library to search.

For each call, the next record is expanded into the caller's record buffer. If the actual record length is less than the record buffer, the record is transferred intact and the buffer is padded with blanks. If the actual record is longer, it is truncated and a truncate indicator is set. If requested, the last nonblank character in the record will be indicated. An EOF indicator is set when the last record is returned.

The caller must provide a record buffer and an I/O buffer of at least one sector. XR2 must contain the address of the leftmost byte of a 19-byte parameter list (see Figures 6-31 and 6-32). The source library get transient requires a 15-byte work area following the parameter list. The source library get routine include version (#MASYL), may be link-edited with or loaded by other modules to retrieve requested library members. It does not, however, support the find request function.

Library Record Put

The library record put routine ($MAPUR) places source or procedure records into a specified library in compressed format. The records must be from 40 to 120 bytes in length.

The library control block (LCB) must be supplied by the caller as a parameter list. XR2 must point to the leftmost byte. (See the *Data Areas Handbook* for the LCB format).

The caller's first call to $MAPUR must be an open request for the output library. If the open is successful, the caller makes one put request call for each record. After the last record, the caller must call $MAPUR with a close request.

For the open request, $MAPUR calls the library open/close routine ($MACOM or $MALCO). For each put request, $MAPUR moves the record to the caller-supplied I/O buffer, compressing the record in the process. When the I/O buffer is full, it is written to the library. If available library space is exceeded, SYSLOG (#CLSG) is called to issue a message. In responding to the message, the operator can either cancel the job, or accept the partial member that was written. If the operator accepts the member as written, $MAPUR closes the library with the partial member and indicates to the caller that the member has been closed. For the close request, $MAPUR closes the output library by calling library open/close ($MACOM or $MALCO).

The library control block (LCB) must be supplied by the caller as a parameter list. XR2 must point to the leftmost byte. (See the *Data Areas Handbook* for the LCB format.)

Library Sector Get/Put

The library sector get/put routine ($MAPGS) retrieves members from or places members into a specified library. Members are accessed in block or sector mode without checking content.

When $MAPGS is called, the user must request either a get or a put operation.

If a get is requested, the user may first request that the single name find routine (#MASFN) locate the specified member. If #MASFN cannot find the member in the system or user libraries, the completion code (LCBCOMP) is set to indicate this fact and control returned to the calling program. If the find is successful, $MAPGS calls disk IOS to read the requested sectors into a user provided I/O buffer. Several calls to disk IOS may be required to read the entire member unless the I/O buffer is large enough for a single read operation. The successful completion bit in LCBCOMP is set after each read. After the entire member is retrieved, a bit in the completion code (LCBCOMP) is turned on to notify the user.

If a put is requested, the caller of $MAPGS can request the library open/close routine ($MACOM or $MALCO) to open the specified library. When the library is open, $MAPGS is called with a put request and then $MAPGS calls disk IOS to write the member sectors from the user supplied I/O buffer into the specified library. On the last put call, after the entire member is written to the library, $MACOM or $MALCO is called to close the library. Control is then returned to the calling program.

The library control block (LCB) must be supplied by the calling program as a parameter list. XR2 must point to the leftmost byte. (See the *Data Areas Handbook* for the LCB format.)

LIBRARY MEMBER PROTECTION

The library member protection routine (#MAMPM) is a refreshable transient. #MAMPM has two functions:

- Prevent source entry utility (SEU) tasks from updating a library member if that member is currently being updated.
- Prevent initiation of an SEU task when an SEU user is inquiring into SEU

To perform these functions, #MAMPM builds an SEU member chain. If either condition occurs, the SEU task is immediately canceled. Otherwise, an element is added to the chain.

When the library member protection routine (#MAMPM) is called, it enqueues the SEU member chain queue header at level 3. This provides #MAMPM exclusive use of the member chain for the entire duration of the program. #MAMPM then assigns space to the assign/free area for the member chain element it plans to build (see Figure 6-1). This area is then added at the end of the member chain.

Next, the member chain element is built by retrieving the fields that are contained in the element. The task control block (TCB) address of the user task is obtained and moved to the member chain element. Also moved into the member chain element is the terminal ID, obtained from the terminal unit block (TUB). The terminal ID is also put into the 2-byte field at the end of the single name find parameter list. Single name find (#MASFN) is then called to find the member from the specified library. If the find is successful, the sector address of the member is moved into the member chain element. If the find is not successful, a null member address of zeros is moved into the member chain element. This indicates that a new member is being created by the user. The chain is then searched. The terminal ID of the task is compared with the terminal ID of each element on the chain. If a match occurs, it indicates that the same user has made an inquiry back into SEU. A message is then displayed to the user through SYSLOG, and the user task is cancelled. Next, the member address in the member chain element is compared with the member address in each block of the chain. The only valid match is if the member address is zero, otherwise, it indicates that another SEU user is updating that member. The type and name of the member along with a message is displayed to the user through SYSLOG, and the user task is terminated. If an error occurs, the member chain element is freed from the member chain during termination. If the chain is searched and the end is reached without error, the member chain queue header is dequeued and control is returned to the user.

The single name find parameter list (see Figure 6-28) with an additional 2-byte area for terminal ID must be supplied by the calling program. XR2 must point to the leftmost byte of the list.

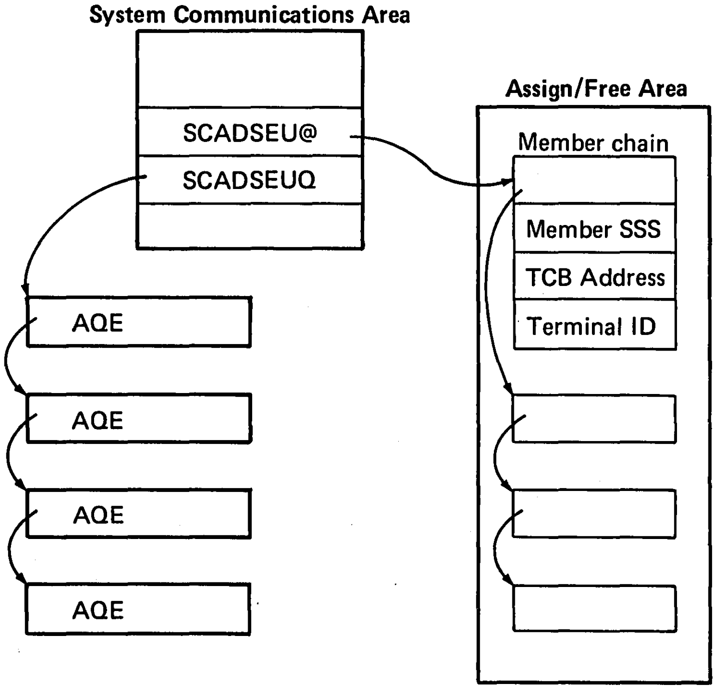


Figure 6-1. SEU Member Chain

ACTIVE FORMAT 1 AREA ACCESS ROUTINE

The active format 1 area access routine (#CSAF) is a transient used to manage requests for get and put of format 1 blocks in the active format 1 area (AFA).

#CSAF supports the following functions:

- Get by label
- Get by name
- Get by address
- Put

#CSAF examines the function byte in the user provided AFA access parameter list to determine the service requested (see Figure 6-33).

If the get label request is for unit F1, #CSAF searches the AFA chain for a format 1 with the specified label. When found, #CSAF places the format 1 address in the parameter list. The request may be further qualified by date and ID verify. Date verify compares the creation date in the format 1 to the date in the caller's parameter list. ID verify causes the allocation queue element (AQE) chain associated with the format 1 to be searched for at least one AQE containing the current task control block (TCB) address. If the caller requests a move, #CSAF places the format 1 in the caller's I/O area. The parameter list return code is updated when the operation is completed. Control is returned to the calling program.

If the request is get label for unit I1, #CSAF searches the file specification block (FSB) chain pointed to by the job control block (JCB). The FSB contains a pointer to an associated format 1. #CSAF examines each format 1 for the specified label and unit. The request may be further qualified by date verify. If the caller requests a move, #CSAF places the format 1 in the caller's I/O area. The parameter list return code is updated and control is returned to the calling program.

If the request is for get name, #CSAF searches the file specification block (FSB) chain pointed to by the job control block (JCB) for the specified name. The FSB contains the format 1 address. If the format 1 contains the correct unit field, #CSAF places the format 1 address in the caller's parameter list. If the caller requests a move, #CSAF places the format 1 in the caller's I/O area. The parameter list return code is updated and control returned to the calling program.

If the request is for get by address, #CSAF moves the format 1 at the specified address to the caller's I/O area.

If the request is for put format 1, #CSAF replaces the format 1 in the AFA, at the address specified in the caller's parameter list, with the format 1 in the caller's I/O area.

XR2 must point to the leftmost byte of the user-provided AFA access parameter list (see Figure 6-33).

CROSS-REFERENCE RESOLVER

The cross-reference resolver routine (#MAXRF) places loader information in the where-to-go (WTG) table. Without a WTG table, the caller of an SSP module must first call system find to build a loader parameter list, then call the loader to load the module. If the caller supplies a WTG table, the call to system find is eliminated. #MAXRF also fills in format index tables in SSP modules.

#MAXRF is run any time SSP load members in the system library are moved. This may be after RELOAD, system library compress, or replacement of an SSP load member.

#MAXRF reads the system library directory and builds a resident table of all SSP load modules. Each entry contains the second through fifth character of the name, the disk address, the number of text sectors, the RLD displacement, a WTG table indicator, and a format index table indicator. #MAXRF builds a main storage format index from the index sectors of the command processor screen format modules.

#MAXRF then uses information from the resident table to find and read the last four text sectors of each module with a WTG or format index table. #MAXRF calculates the end of the module by using the number of text sectors and the RLD displacement. For each module in the WTG table, #MAXRF searches the resident table for a matching entry. If a match is found, the loader information is moved from the resident table to the module's WTG table. If a match is not found, loader data for the no-op routine (#MANOP) is placed in the WTG table. When executed, #MANOP issues the error message. For each module with a format index table, #MAXRF uses the module's format index table displacement to get information from the main storage format index to the format index table. The last four text sectors are then written back to their original location on disk.

The format index table and WTG table must be the last bytes in the module. Each must be preceded by X'FFFF'. If both are present, the format index table must precede the WTG table.

Each entry in the WTG table is 9 bytes, in the format CCCCSSSnR:

```
CCCC = second through fifth character of called
         module name
SSS  = disk address of called module  )   #MAXRF
n    = number of text sectors          }   loads these
R    = RLD displacement                )
```

Each entry in the format index table is 9 bytes, in the format DDSSSfnll:

```
DD  = displacement into format index sectors
SSS = disk address of screen format  )
f   = number of FDT                  }   #MAXRF
n   = number of text sectors         }   loads these
ll  = input length of screen         )
```

DUPLICATE KEY DISPLAY ROUTINE

The duplicate key display routine (#CSDK) is a transient that display the duplicate key and returns the operator-selected option to the calling program.

When #CSDK is called, the user must supply a where-to-go table specifying #CSDK, an 80-byte message buffer area, and the duplicate key halt parameter list. The calling program, by way of the parameter list, provides pointers to the key and message build buffer (see Figure 6-34 for the duplicate key display parameter list format). #CSDK calls SYSLOG (#CLSG) to output a message indicating a duplicate key was found in the specified file. The operator must indicate whether the key is to be displayed in EBCDIC or converted to hexadecimal notation. After the operator responds to the message, #CSDK moves the duplicate key to the message build buffer and calls #CLSG to display the key and the options specified in the duplicate key display parameter list. When the operator responds, the requested option is placed in the parameter list and control returned to the calling program.

XR1 must point to the leftmost byte of the user-provided parameter list for duplicate key display.

PRINT IMAGE VERIFY ROUTINE

The printer image verify routine (#CSIM) is a transient program that has four functions:

- Set the print image
- Set the forms number
- Indicate page separator information for spool
- Set lines per page

#CSIM processes all printer specification blocks (PSBs) associated with the calling task. Multiple printers may be processed with one call to #CSIM.

The PSB contains a flag byte, forms number, and lines-per-page fields used by #CSIM. #CSIM scans the PSB chain associated with the calling task and performs the following functions:

- *Image processing:* If the PSB flag byte indicates processing is required, #CSIM compares the current print image to the work station configuration record. If they do not match, SYSLOG displays a message to the operator. The operator, after changing the print belt, replies to the message. #CSIM then sets the new print image into the printer control unit.
- *Forms processing:* If the PSB flag byte indicates forms processing is required, #CSIM compares the current forms number in the TUB to the PSB forms number. If they do not match, SYSLOG displays a message to the operator. The operator, after changing the forms, replies to the message. #CSIM then sets the new forms number into the printer terminal unit block (TUB).
- *Spool separator page processing:* If the calling task is spool, #CSIM prompts the operator to indicate if separator pages are required. If the operator's reply is option 1, #CSIM updates the PSB flag byte to indicate separator pages are required.
- *Lines per page processing:* If the PSB flag byte indicates lines per page processing is required, #CSIM sets the lines per page from the PSB into the printer control unit.

No caller parameter list is required to execute #CSIM. The PSB, however, must be initialized when #CSIM is called (see *Data Areas Handbook* for PSB format).

DISK VTOC READ/WRITE

The disk VTOC read/write routine is supplied in two versions. #CSVF is a transient, and @CSVF is designed to be link-edited with system routines that address the nucleus.

Disk VTOC read/write performs three major functions:

- Format 1 read
- Format 1 write
- Existence test

The program calling disk VTOC read/write must supply a parameter list with leftmost byte address in XR2 (see the *Data Areas Handbook* for disk VTOC read/write parameter list format).

Disk VTOC read/write examines the function byte in the parameter list to determine the service requested:

- *Format 1 read:* disk VTOC read/write uses the scan function of disk IOS. A scan mask containing the requested label, and the date as an option, is passed to disk IOS by way of a pointer in the IOB. When the format 1 is located in the VTOC, disk IOS places the sector number in the IOB and disk VTOC read/write reads the sector containing the format 1 into the data area. The format 1 is then moved from the data area to the calling program's I/O area and control returns to the caller.

  There are four types of format 1 read requests:
  - Read next
  - Read next same label
  - Read specific
  - Read by sector displacement

  Format 1 read request processing varies depending on the type of request. The caller also has the option of requesting a date verify. Date verify uses a scan mask containing the label and date. This allows the caller to select a specific file from a group of files with the same label. Without date verify specified, the format 1 with the latest date is selected. If the format 1 read request is:
  - Read next, the parameter list displacement byte is tested for a first request. If it is the first request, the scan starts at the first sector in the VTOC format 1 area. If it is not the first request, the parameter list sector number is incremented and the scan started in the next sector. The scan is for the next logical format 1 in the VTOC and no compare for label or date is made.
  - Read next-same label, processing is the same as a read next request except the scan mask is set up to compare on a specified label.
  - Read specific, the scan starts at the VTOC format 1 area beginning. The scan is for a format 1 containing the specified label and optional date.
  - Read by sector displacement, the disk sector specified in the parameter list is read and the format 1 at the specified displacement moved to the caller's I/O area.

- *Format-1 write:* disk VTOC read/write uses disk IOS to read the sector specified in the parameter list. The format 1 in the caller's I/O area is moved into the sector just read from disk. Disk IOS is called again to write the updated sector back to disk.

- *Existence test:* processing is the same as format 1 read-read specific processing except the format 1 is not moved into the caller's I/O area. The sector/displacement and the return code are updated in the parameter list.

Figure 6-13 shows disk VTOC read/write control flow.

DISKETTE VTOC READ/WRITE

Diskette VTOC read/write is transient and consists of three modules. The mainline module (#CSVI) process the requests and routes control to the two conversion modules (#CSVJ and #CSVK) as required.

Diskette VTOC read/write performs three major functions:

- Convert format 1's and header 1's
- Prepare diskettes for processing
- Format 1 read/write

The program calling diskette VTOC read/write must supply a parameter list with the leftmost byte address in XR2 (see the *Data Areas Handbook* for diskette VTOC read/write parameter list format).

#CSVI examines the function byte in the parameter list to determine the service requested:

- *Convert header 1's:* #CSVJ reads the diskette VTOC, converts the header 1's to format 1's and places the format 1's into a diskette VTOC area on disk. Subsequent diskette format 1 read/write requests access this area on disk.
- *Convert format 1's:* #CSVK reads the diskette VTOC area on disk, converts the format 1's to header 1's and places the header 1's into the diskette VTOC on diskette.
- *Prepare diskette:* #CSVI issues a recalibrate request for the diskette. The volume label is read and verified. The volume label and physical attributes are placed into the system communication area (SCA). A 4-byte lock number is also placed into the SCA and written on the diskette. #CSVJ is then called to create the diskette VTOC area on disk (convert header 1's function).
- *Format-1 read:* The diskette format 1 in the diskette VTOC area on disk (placed on disk by #CSVJ) is read the same as by the format 1 read function of disk VTOC read/write. Latest date processing, however, is not supported. (See format 1 read function in Diagram 6.12.)
- *Format-1 write:* This function is identical to disk VTOC read/write, #CSVF. (See format 1 write function in Diagram 6.12.)
- *Existence test:* The check for the existence of a specific format 1 is performed the same as by disk VTOC read/write, #CSVF. (See existence test function in Diagram 6.12.)

Figure 6-14 shows diskette VTOC read/write control flow.

MESSAGE RETRIEVE

The message retrieve routine (#MGRET) is a transient that locates the message text of a specified message identification code (MIC).

#MGRET ensures a valid message member specification by checking the message retrieve parameter list for valid indicators and the message member pointer in the appropriate communication region for a nonzero SSS (see the *Data Areas Handbook* for message retrieve parameter list format). Disk IOS is then called to locate the message member sector by scanning the message member for the sector identified by a MIC greater than or equal to the one requested. #MGRET locates the desired message by reading the message member sector, looking for the specified MIC. When the message is found, #MGRET blanks the caller's buffer, places the message in the buffer, and places the message text length in the parameter list. If an error is detected, the error MIC is placed in the parameter list. Upon completion, control is returned to the calling program.

Figure 6-15 shows the control flow for the message retrieve routine.

SYSIN

SYSIN performs two major functions:

- Retrieve records from the keyboard
- Retrieve records from library procedure members

Records retrieved from the keyboard are returned directly to the calling program. If records are retrieved from a procedure member, SYSIN performs:

- Substitution (#CLPR or #CLSB)
- If statement processing consisting of:
  - Existence testing (#CLFX, #CLSM, #CLAC, or #CLBL)
  - Character string comparisons (#CLSS)

The SYSIN mainline module (#CLSS) uses the above routines to perform the various functions performed when SYSIN retrieves library procedure members. (See Figure 6-16 for SYSIN control flow.)

When a user program requests SYSIN services, the SYSIN Load Transient (#CLSN) is loaded into the transient area. #CLSN then loads and passes control to #CLSS at location X'DD00' (logical address).

When the initiator is processing, #CLSS is link-edited with the initiator. A 2K block of main storage starting at X'F800' (logical address) is reserved by #CLSS to handle the SYSIN routines used to process procedure members. This 2K block of main storage may also be used by the initiator to handle OCL statements, and history file put (@HFPUT).

The SYSIN calling program must place the leftmost byte address of the SYSIN parameter list in XR2 (see *Data Areas Handbook* for SYSIN parameter list format).

The history file put routine (@HFPUT) and the source library get routine (#MASYL) are link-edited with the SYSIN mainline module. #MASYL retrieves procedure records and @HFPUT places records in the history file.

SYSLIST

SYSLIST provides a means of printing or displaying system output to the user. SYSLIST executes as either a transient or a loadable module.

The SYSLIST transient interface consists of the following:

- *#CLST:* SYSLIST printer transient. This module is always loaded first on every transient call to SYSLIST.

- *#CLSC:* SYSLIST work station transient.

- *SLIST:* SYSLIST macro that provides linkage to the transient module (#CLST).

The loadable SYSLIST interface consists of the following:

- *#CLSP:* Loadable SYSLIST printer module.

- *#CLSW:* Loadable SYSLIST work station module.

- *SLSTL:* SYSLIST load macro that loads either #CLSP or #CLSW into the specified user load area.

The SYSLIST printer modules (#CLST and #CLSP) list system output on the system or specified work station printer. The SYSLIST work station modules (#CLSC and #CLSW) list system output on the work station display screens.

Two types of system output are listed by SYSLIST: Type 1 output comes from a message member and Type 2 output comes from a system program.

When the SYSLIST transient is called, the user must supply a SYSLIST parameter list with the leftmost byte address in XR2. Control is then passed to #CLST via the SLIST macro. If the display screen is to be the SYSLIST device, #CLST transfers control to #CLSC, which is loaded in the transient area.

When loadable SYSLIST is called, the user must supply a SYSLIST parameter list with the leftmost byte address in XR1. Control is passed via a branch to the specified user load area. Prior to passing control, the user must have executed macro SLSTL, to load the proper module into the load area.

The SYSLIST parameter list may be in one of two formats: TYPE 1 or TYPE 2. (See the *Data Areas Handbook* for SYSLIST parameter list formats.)

SYSLOG

SYSLOG provides a method for printing or displaying messages.

Messages are printed only if the system is in single program mode and the printer is the SYSLOG device and not busy.

Six types of messages may be displayed on the work station or operator's display screen. The six message types are:

- *Type 1:* Messages from a message member without a response

- *Type 1R:* Messages from a message member with a data response

- *Type 2:* Messages from a user program without a data response

- *Type 2R:* Messages from a user program with a data response

- *Type 3:* Messages from a user program with a format line request

- *Type 4:* Messages from a message member with 8 bytes of data inserted at the beginning of the message

The SYSLOG mainline module (#CLSG) can be called by way of the SYSLOG push/pull transient (#CLXS).

When loaded into the transient area, #CLXS pushes 10K of user program from main storage to disk. It then loads #CLSG into main storage at location X'C900' (logical address). #CLXS also moves the parameter list, ATR, SSN of #CMWO and, if the message type is 2, 2R, or 3, the message from the transient area to the first sector of the user area just cleared.

#CLSG uses printer IOS (IPTR) to output messages to the printer and uses the command processor (#CMWO) to output messages to the display station display screen.

History file put (@HFPUT) and message retrieve (@MGRET) are link edited with #CLSG. @MGRET is used to retrieve type 1, 1R, and 4 messages from the proper message member. @HFPUT is used to log messages and responses to the history file when requested.

The caller of SYSLOG must supply a parameter list with XR2 containing the leftmost byte address (see the *Data Areas Handbook* for the SYSLOG parameter list format).

If option 3 (cancel request) is returned to SYSLOG in response to a message, #CLSG calls the end-of-job transien (#CTEI) to terminate the job. #CLSG returns control to #CLXS. If #CLXS is called by #CLSG, it is to move the SYSLOG parameter list back to the transient area and to pull the 10K of user main storage previously pushed to disk back into the user area. If an option response or data response was received, #CLXS passes the response to the caller. If the option is D, a main storage dump is performed.

HISTORY FILE PUT

History file put documents information such as OCL and utility control statements entered by the system operator, error messages, and operator responses.

History file put exists either as a transient (#HFPUT) which executes in the transient area or as a link-edit module (@HFPUT) along with the user program and branched to by user program request.

The history file is not a data file but is located in the system area on disk. (See the *Data Areas Handbook* for history file description and format.) Entries are placed in each sector of the history file, one after another, until the point is reached where the next entry would extend beyond the sector. In this case, the next entry is placed in the following sector and the current sector pointer is updated. When the point at which there is no following sector is reached, the entry is placed in the sector at the beginning of the file. This condition is known as wrap-around. The entry last placed in the history file is the current entry.

The system communication area (SCA) contains the history file status:

- *SCAHIST:* history file beginning sector address
- *SCAHFSIZ:* history file size in sectors
- *SCAHFCUR:* history file sector containing current entry
- *SCASYS1:* error condition if SCAHFERR set

If the SCAHFERR bit is not set in SCASYS1, #HFPUT (or @HFPUT) reads the history file status from the SCA to determine the history file start address, the history file size, and the address of the current entry. #HFPUT uses this information to read the sector containing the current entry from disk and places it into the history file I/O buffer. Next, #HFPUT removes all trailing blanks from the input text, and moves the text into the history file data buffer. Additional information about the entry is also placed in the history file data buffer: the terminal identifier is read from the terminal unit block (TUB), the user identification is read from the job control block (JCB) or terminal unit block (TUB), the job identification is read from the JCB, the current time is extracted from the system timer routine ($TOD), and control bits are set indicating if the entry was broadcast and if the entry was displayed to the operator.

The completed entry is moved into the history file I/O buffer immediately following the previous current entry. If the new entry will not fit into the current sector, it is placed into the next sector, and the SCA current sector address is updated. A maximum of four lines of equal length may be placed into the history file on each call. Disk IOS writes the entry from the history file buffer to the history file on disk.

The history file put routine caller must provide a 10-byte parameter list with XR2 containing the leftmost byte address. (See the *Data Areas Handbook* for the history file put parameter list format).

SUPERVISOR TASK ATTACH TRANSIENT (#SVAT)

The supervisor task attach routine (#SVAT) is a transient used to attach a new task to the system.

$SVAT assigns the task control block (TCB) to run the new system task. The number of 2K main storage blocks required to start the new task is then calculated. This is the largest program size required to load the program mainline or the size required to execute the program.

If enough main storage and swap area are available to run the task, #SVAT assigns a request block (RB), action control element (ACE) — if required, and swap area. All necessary fields in the TCB are initialized to start the task. The new TCB is then placed on the swap-in-queue with the dispatching address pointing to the bootstrap code in the RB.

Input to the supervisor attach transient is the attach parameter list supplied by the calling program. XR1 contains the address of the leftmost byte of the attach parameter list (see Figure 6-35 for the format and content of the parameter list).

Output from #SVAT is the new TCB placed on the swap-in queue. XR1 contains either the address of the new task's TCB or the value of the error return code.

After execution, #SVAT returns control to the calling program or to a transient pointed to by the attach parameter list.

SUPERVISOR TASK DETACH OR CHANGE ORIGIN POINT (#SVAU)

The supervisor task detach or change origin point routine (#SVAU) is a transient used to:

- Change a task's point of origin
- Detach a task from the system

Change Point of Origin

A task's point of origin within main storage may be changed upon request. When control is passed from the scheduler to the user program or from the user program to the scheduler, a task's location in main storage may require changing.

#SVAU determines if enough main storage space is available to load the requested program. The old swap area on disk is freed if enough main storage area exists and the old swap area is larger than the required swap area. The old swap area is also freed if the required swap area is smaller than the old swap area and the task is in termination.

All main storage assigned to the task, except the first 2K bytes, is then freed. The bootstrap code used to start the task is moved to the first 2K byte area and control passed to the program indicated in the attach parameter list.

Detach Task

The task detach function is called when a task goes to end of job.

#SVAU frees the task's main storage area and swap area on disk. The task's request blocks (RBs) and job control block (JCB) are then freed if required. Finally, the task control block (TCB) is freed. When the detach function is complete, control is returned to the dispatcher (control storage) and the transient area is freed.

Input to the supervisor detach and change origin point transient is the attach parameter list supplied by the calling program. XR1 contains the address of the leftmost byte of the attach parameter list. (See Figure 6-35 for the format and content of the parameter list.)

Output from #SVAU is a new point of origin for the task or, if requested, the task is detached from the system.

If an error condition is encountered while executing #SVAU, the error return code is placed in XR1 and returned to the calling program.

SYNTAX CHECKER (#USYX)

The syntax checker (#USYX) performs the following functions:

- Checks for a valid verb in a control statement
- Indicates in the communication table the parameters specified in the control statement
- Checks that parameter values are valid
- Places parameter values (or values that are to be substituted for parameter values) in the communication table
- Checks that parameters are used in valid combinations
- Indicates in the communication table any syntax errors

The syntax checker consists of a single phase, #USYX, that resides in the system library.

Input to the syntax checker consists of:

- Syntax checker parameter list
- Verb list
- Communication table
- Syntax specification module
- Control statement

Note: The syntax checker parameter list, communication table, and syntax specification module are described in the *Data Areas* section of this chapter.

When the syntax checker is called:

- The calling utility places the address of the syntax checker parameter list in register 2
- The syntax checker parameter list indicates the address of the verb list and communication table
- The communication table specifies the name of the syntax specification module to load from the system library

Output consists of the communication table returned to the calling utility with the following information:

- Parameters found in the control statement
- Parameter values
- Syntax errors that were detected

The syntax checker requires 4.5K bytes of main storage for program execution plus space for the syntax specification module, if required.

INFORMATION RETRIEVAL (#SVINF)

The information retrieval transient (#SVINF) is invoked by the $INFO macro instruction in the user's program. #SVINF provides the user with the ability to access certain fields within privileged control blocks (JCB or TCB) or the local area on disk.

The user provides a parameter list which specifies the operation to be performed (get or put) and the data area to be used for communication between the user and the transient.

When #SVINF receives control, XR2 contains the address of the parameter list. #SVINF determines whether the user is permitted to access the field. If the user is not permitted to access the field, a 3 option only SYSLOG message is issued.

DATA MANAGEMENT TASK TRANSFER CONTROL (#SVTTC)

Data management task transfer control (#SVTTC) controls communication between user programs and the communications data management task.

SNAP DUMP (#SVDMP)

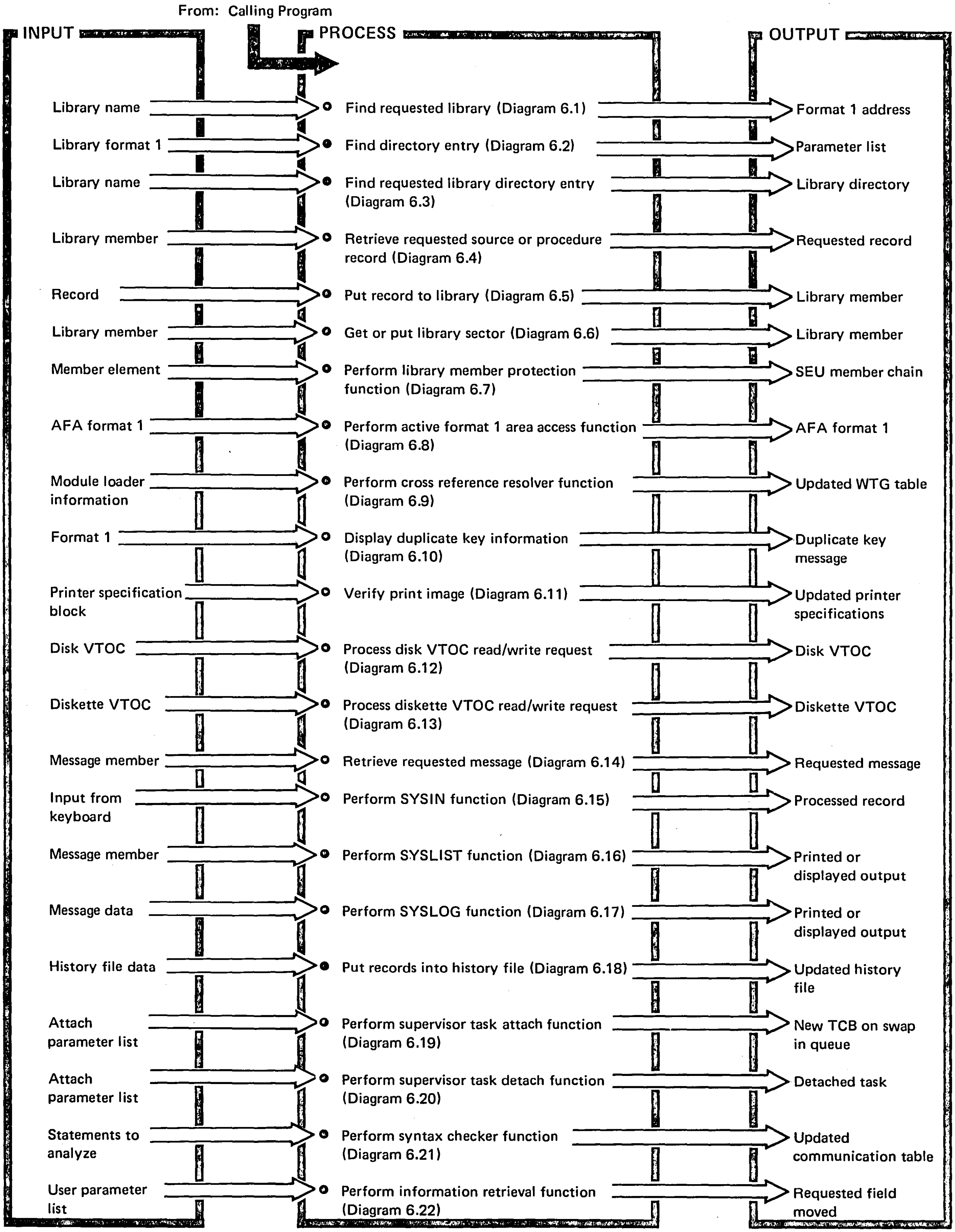
The snap dump transient (#SVDMP) provides a formatted main storage dump when it is invoked by the $SNAP macro in the user program. The user either dumps the entire region of main storage, or specifies storage limits for the dump in a parameter list:

Byte	Description
0	Flag
1-2	Low storage address
3-4	High storage address
5-8	Dump identifier

Method of Operation

This section contains functional diagrams of the system service functions. They are:

- Find a library routine
- Single name find routine
- Librarian find routine
- Source library get routine
- Library record put routine
- Library sector get/put routine
- Library member protection
- Active format 1 area access routine
- Cross reference resolver
- Duplicate key display routine
- Print image verify routine
- Disk VTOC read/write
- Diskette VTOC read/write
- Message retrieve
- SYSIN
- SYSLIST
- SYSLOG
- History file put
- Supervisor task attach
- Supervisor task detach
- Syntax checker
- Information retrieval
- Data management task transfer control
- Snap dump



From: Calling Program

INPUT	PROCESS	OUTPUT
Library name	Find requested library (Diagram 6.1)	Format 1 address
Library format 1	Find directory entry (Diagram 6.2)	Parameter list
Library name	Find requested library directory entry (Diagram 6.3)	Library directory
Library member	Retrieve requested source or procedure record (Diagram 6.4)	Requested record
Record	Put record to library (Diagram 6.5)	Library member
Library member	Get or put library sector (Diagram 6.6)	Library member
Member element	Perform library member protection function (Diagram 6.7)	SEU member chain
AFA format 1	Perform active format 1 area access function (Diagram 6.8)	AFA format 1
Module loader information	Perform cross reference resolver function (Diagram 6.9)	Updated WTG table
Format 1	Display duplicate key information (Diagram 6.10)	Duplicate key message
Printer specification block	Verify print image (Diagram 6.11)	Updated printer specifications
Disk VTOC	Process disk VTOC read/write request (Diagram 6.12)	Disk VTOC
Diskette VTOC	Process diskette VTOC read/write request (Diagram 6.13)	Diskette VTOC
Message member	Retrieve requested message (Diagram 6.14)	Requested message
Input from keyboard	Perform SYSIN function (Diagram 6.15)	Processed record
Message member	Perform SYSLIST function (Diagram 6.16)	Printed or displayed output
Message data	Perform SYSLOG function (Diagram 6.17)	Printed or displayed output
History file data	Put records into history file (Diagram 6.18)	Updated history file
Attach parameter list	Perform supervisor task attach function (Diagram 6.19)	New TCB on swap in queue
Attach parameter list	Perform supervisor task detach function (Diagram 6.20)	Detached task
Statements to analyze	Perform syntax checker function (Diagram 6.21)	Updated communication table
User parameter list	Perform information retrieval function (Diagram 6.22)	Requested field moved

Diagram 6.0 (Part 1 of 2). Overview of System Service Programs

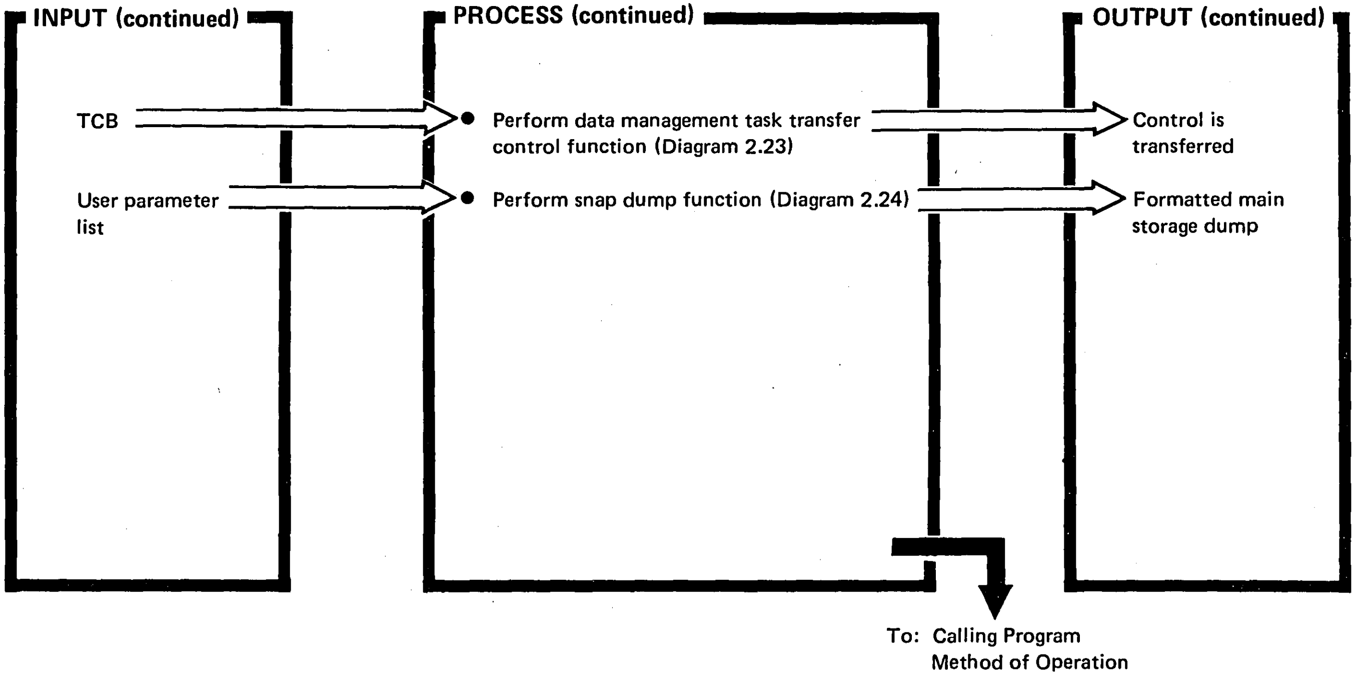
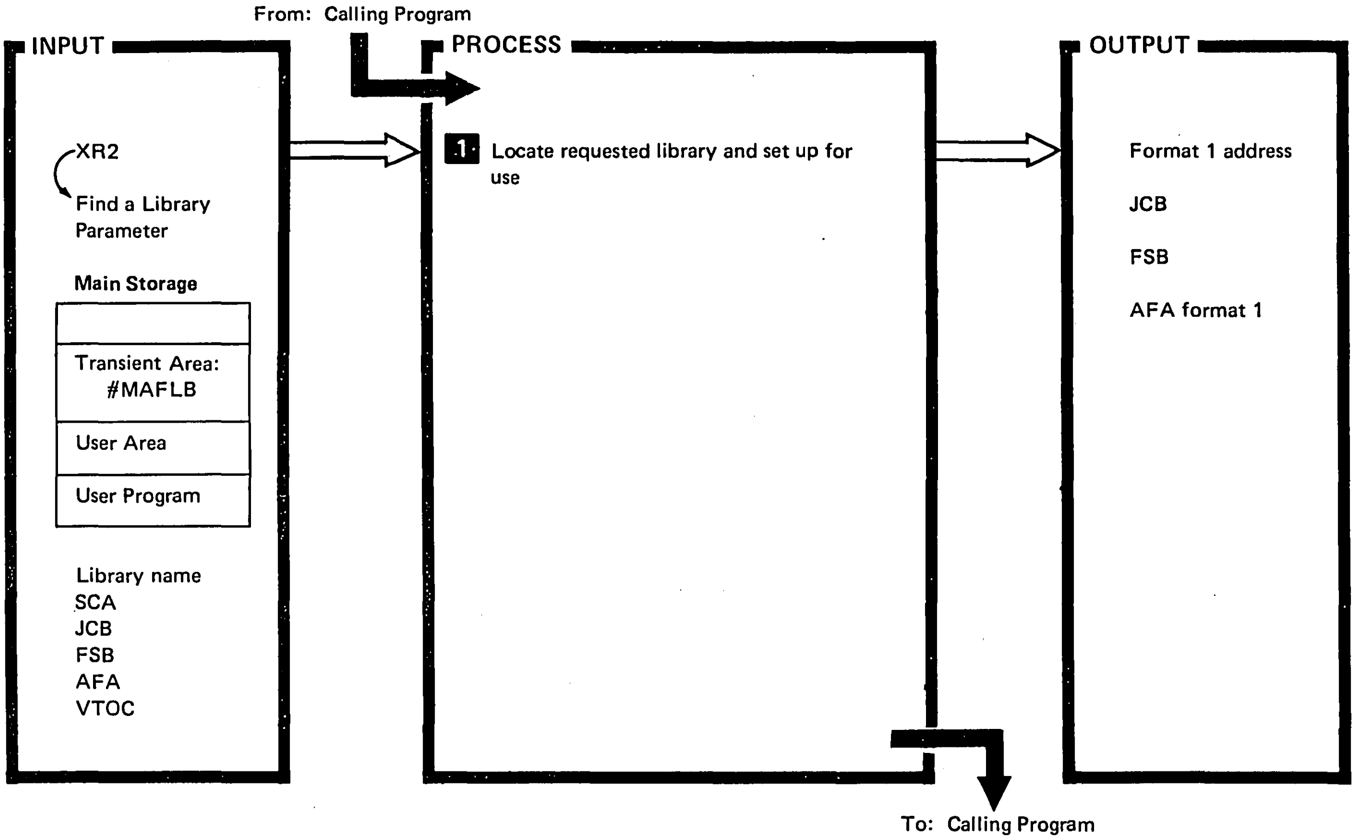
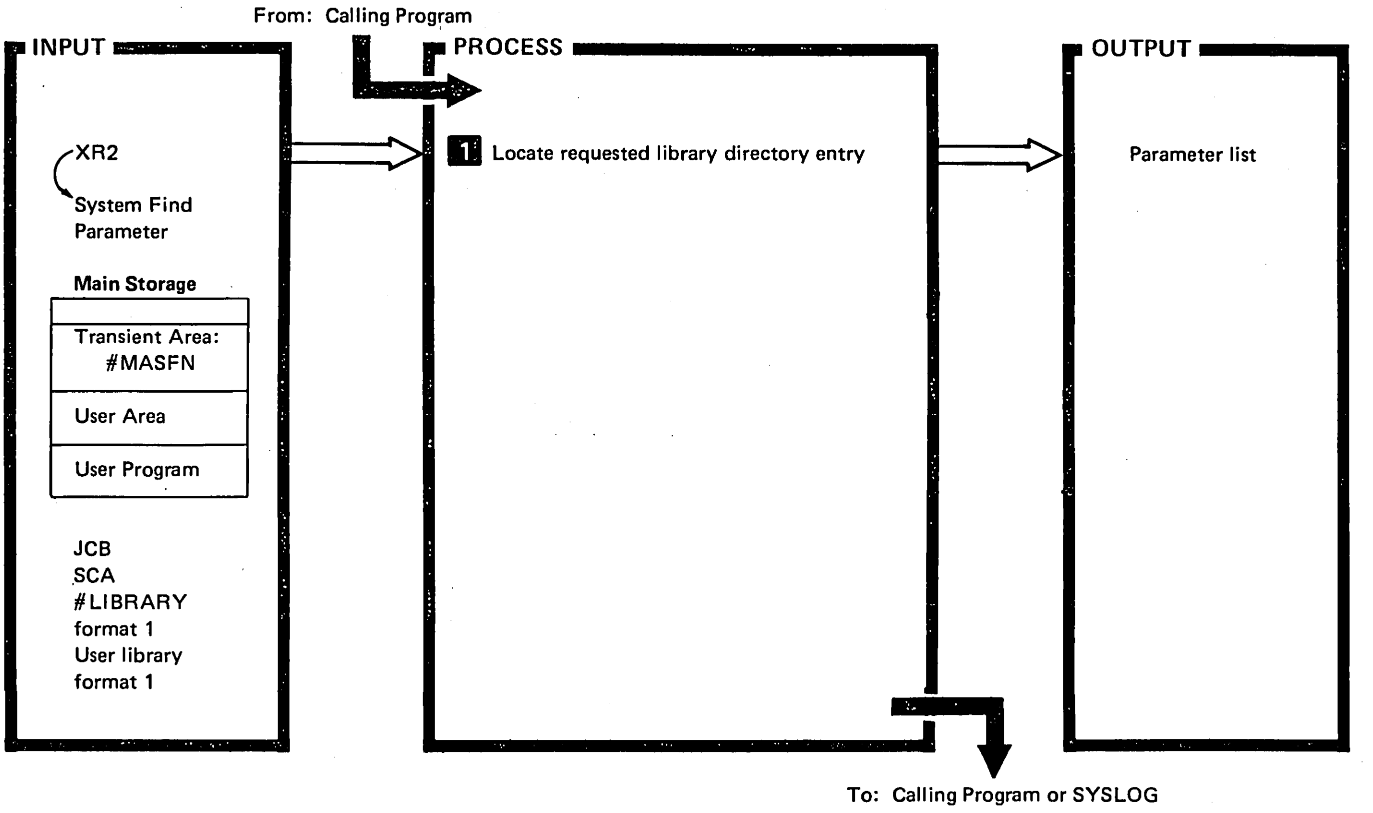

INPUT (continued)
TCB
User parameter list
PROCESS (continued)
● Perform data management task transfer control function (Diagram 2.23)
● Perform snap dump function (Diagram 2.24)
OUTPUT (continued)
Control is transferred
Formatted main storage dump
To: Calling Program Method of Operation

Diagram 6.0 (Part 2 of 2). Overview of System Service Programs



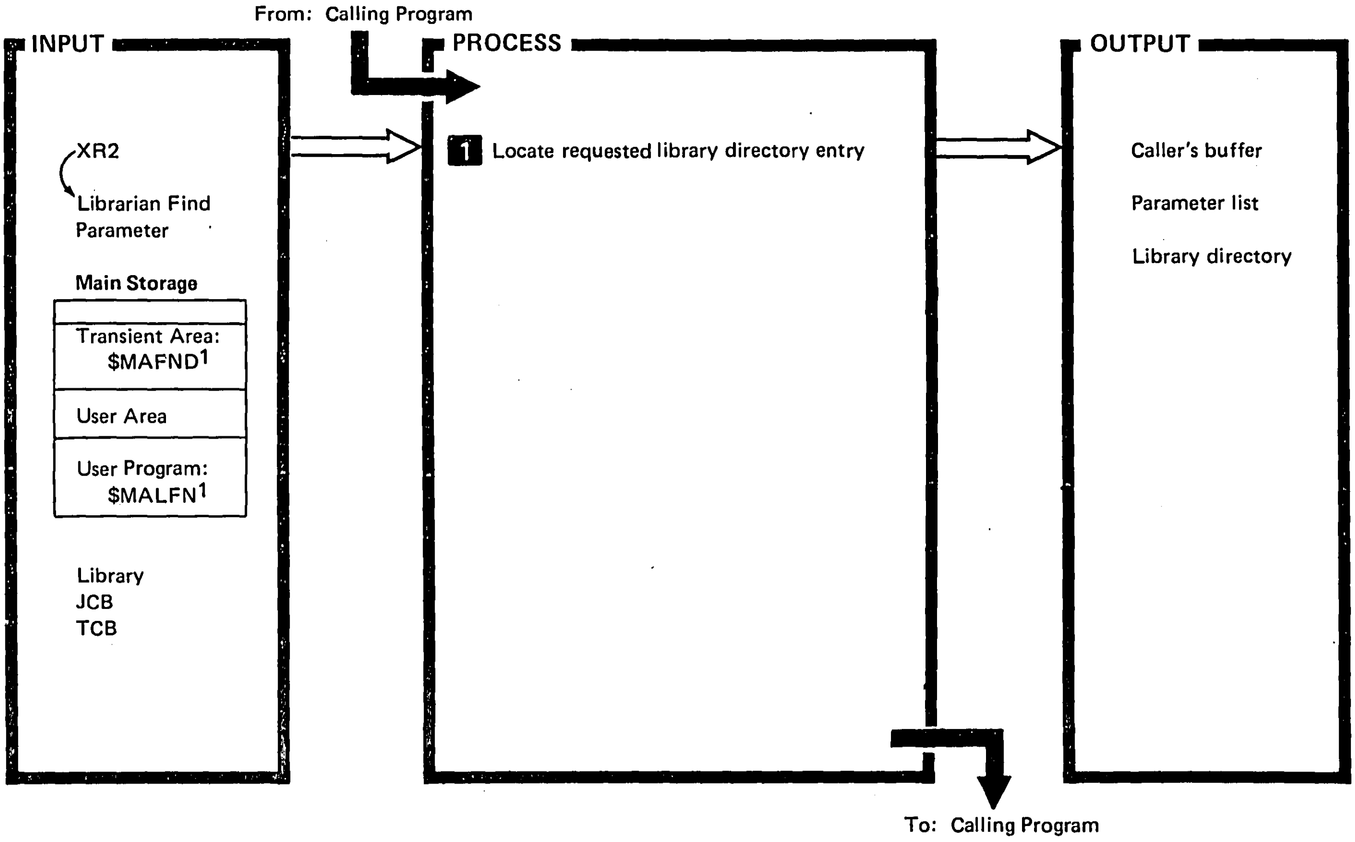
DESCRIPTION	MODULE/ ROUTINE
1 Check if requested library name is #LIBRARY. If yes: ● Get #LIBRARY format 1 address from system communications area (SCA). ● Return to calling program. Search active format 1 area (AFA) for requested library. If requested library in AFA: ● If library format 1 on job control block (JCB) library file specification block (FSB) chain, only return format 1 address. ● If library format 1 not on library FSB chain, build library FSB, chain FSB to JCB, increment use count in format 1, and return format 1 address. ● Return to calling program.	#MAFLB
Search VTOC for requested library (disk VTOC read/write, @CSVF, is link edited with #MAFLB).	@CSVF
If requested library in VTOC: ● Build library format 1 in AFA. ● Build library FSB and chain to JCB. ● Set format 1 use count to 1. ● Return format 1 address to calling program. If requested library can not be found: ● Set format 1 AFA address in parameter list to zero (calling program must handle error). ● Return to calling program.	#MAFLB

Diagram 6.1. Find Requested Library



DESCRIPTION	MODULE/ ROUTINE
1 Move parameter to transient area. Determine if user library to be searched.	#MASFN
If user library, search user library directory for requested entry (scan read). Search system library directory if 1 not to be skipped, 2 entry not in user library, 3 user library skipped, or 4 user library not designated or given (scan read).	Disk IOS
If user call: ● If requested entry found: – Move requested directory data from scan buffer to output parameter. – If requested move format 1 address of library member is found in parameter list start address field ($FNDDF1F): a. Move output parameter to caller's area. b. Return to calling program. ● If requested entry not found: – Leave unchanged parameter list in caller's area. – Return to calling program. If loader call: ● If requested entry found: – Move loader information from directory entry to parameter list. – Move parameter to caller's area. – Return to calling program. ● If requested entry not found:	#MASFN
– Display name of member not found.	#CLXS
– Issue halt.	#CLSG

Diagram 6.2. Find Directory Entry



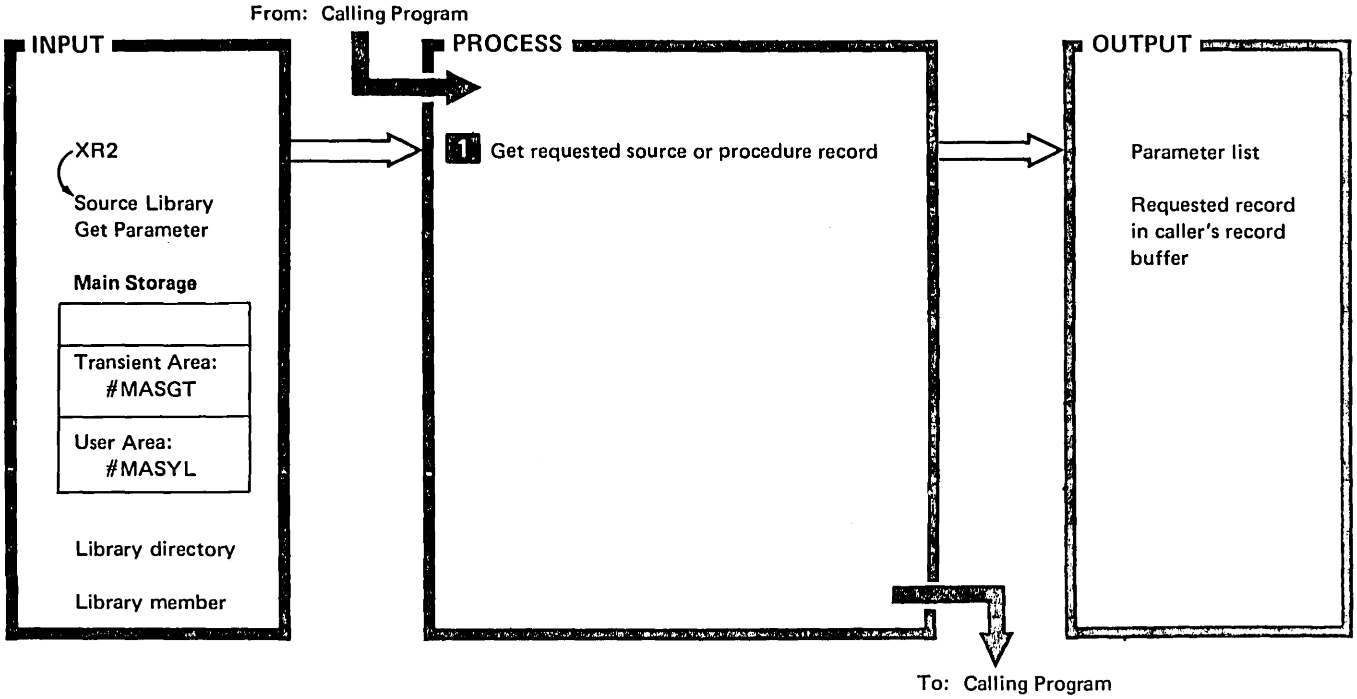
DESCRIPTION	MODULE/ ROUTINE
1 Determine if $MAFND executing in user area or transient area and set up code accordingly. Determine library to search: ● Specified library. ● Designated user library. ● System library. ● Designated user library, then system library.	$MAFND or MALFN
Enqueue library directory to be searched on the first call scan library directory to first requested member. Read library directory into caller's buffer.	Disk IOS
Search library directory for match on type and name (full or partial name). If requested entry found, update parameter list to point to found entry in caller's buffer. When at buffer end:	$MAFND or $MALFN
● Write caller's buffer back to directory on disk if write switch on in parameter list.	Disk IOS
● Go to **1** **A**. If directory entry not found: ● Update parameter list to indicate requested library directory not found. ● Set end of file (EOF) switch. ● Go to **1** **A**.	$MAFND or $MALFN

[1] $MAFND may execute in transient area or user area ($MALFN).

Diagram 6.3 (Part 1 of 2). Find Requested Library Directory Entry

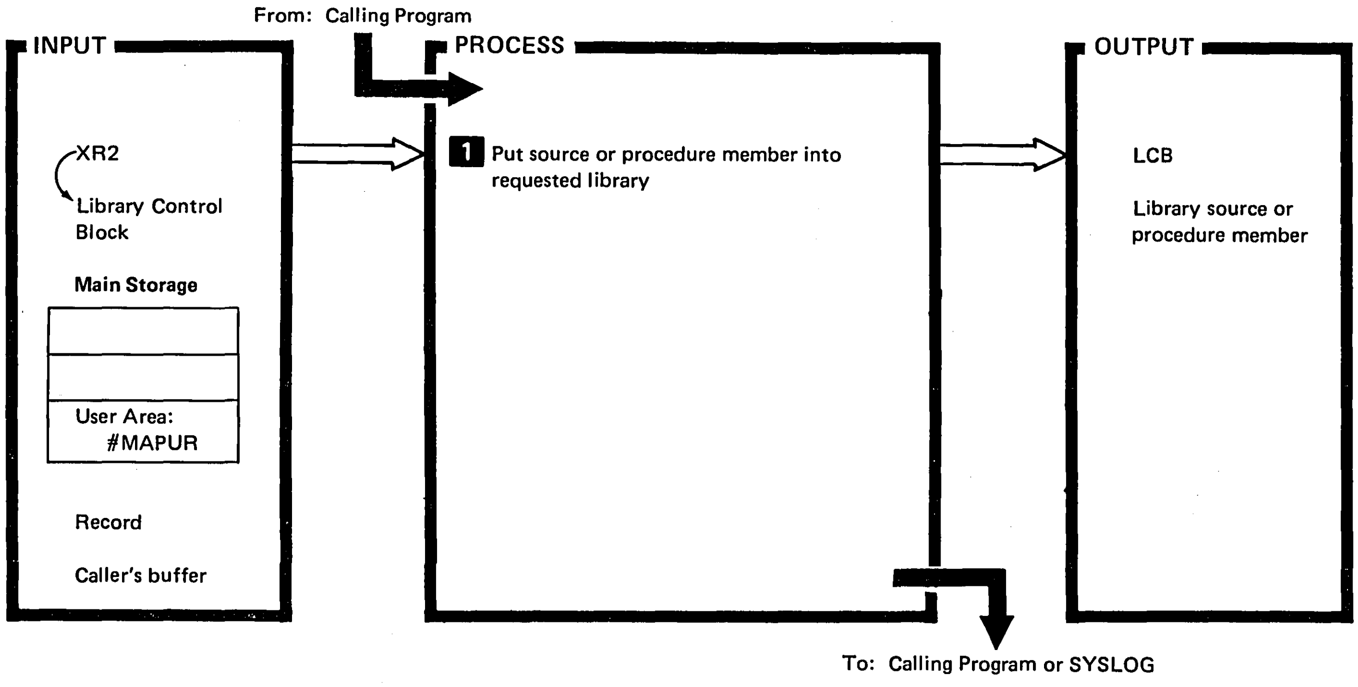
DESCRIPTION	MODULE/ ROUTINE
A Terminate program: ● Perform cleanup. ● Dequeue library directory on last call (EOF on single name call without write). ● Move parameter back to user area. ● Return to calling program.	$MAFND or $MALFN

Diagram 6.3 (Part 2 of 2). Find Requested Library Directory Entry



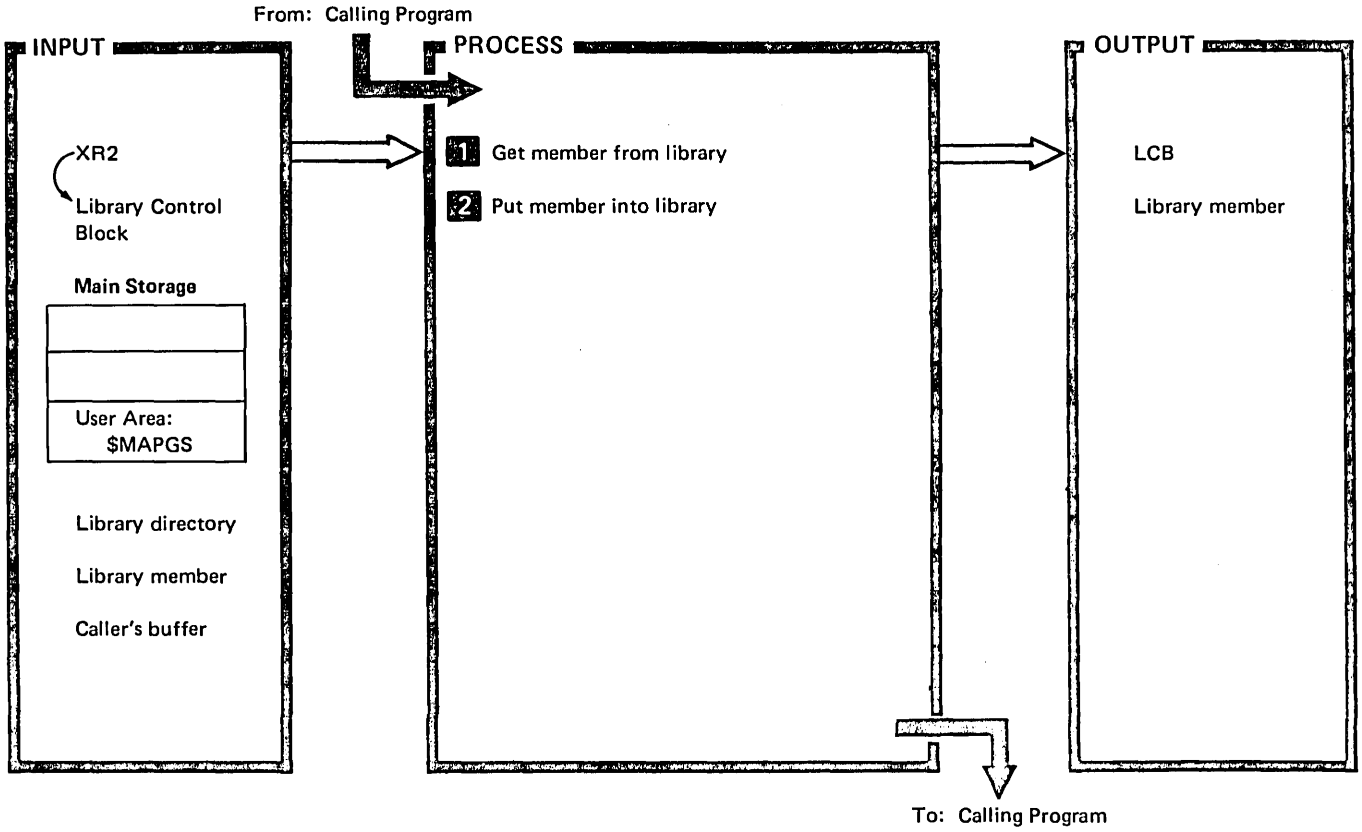
DESCRIPTION	MODULE/ ROUTINE
1 If get first request, perform first time processing. If find request (#MASGT only):	#MASGT or #MASYL
• Find requested library member.	#MASFN
• If not found, return not found indicator. • Set parameter list for get first.	#MASGT or #MASYL
If get first request or reprime request, read requested record into I/O buffer.	Disk IOS
If next record to be retrieved: • Blank record buffer. • Read another buffer if at end present buffer. • Expand record and move record from I/O buffer to caller's record buffer. If record larger than caller's record buffer: • Truncate record. • Set truncated record bit in parameter list. If record smaller than caller's record buffer, pad remaining space with blanks: If requested, indicate start of blanks with X'FF'. When last record returned, set on EOF bit. When finished processing present record: • Update parameter list. • Return to calling program.	#MASGT or #MASYL

Diagram 6.4. Retrieve Requested Source or Procedure Record


From: Calling Program
INPUT
XR2
Library Control Block
Main Storage
User Area: #MAPUR
Record
Caller's buffer
PROCESS
1 Put source or procedure member into requested library
OUTPUT
LCB
Library source or procedure member
To: Calling Program or SYSLOG

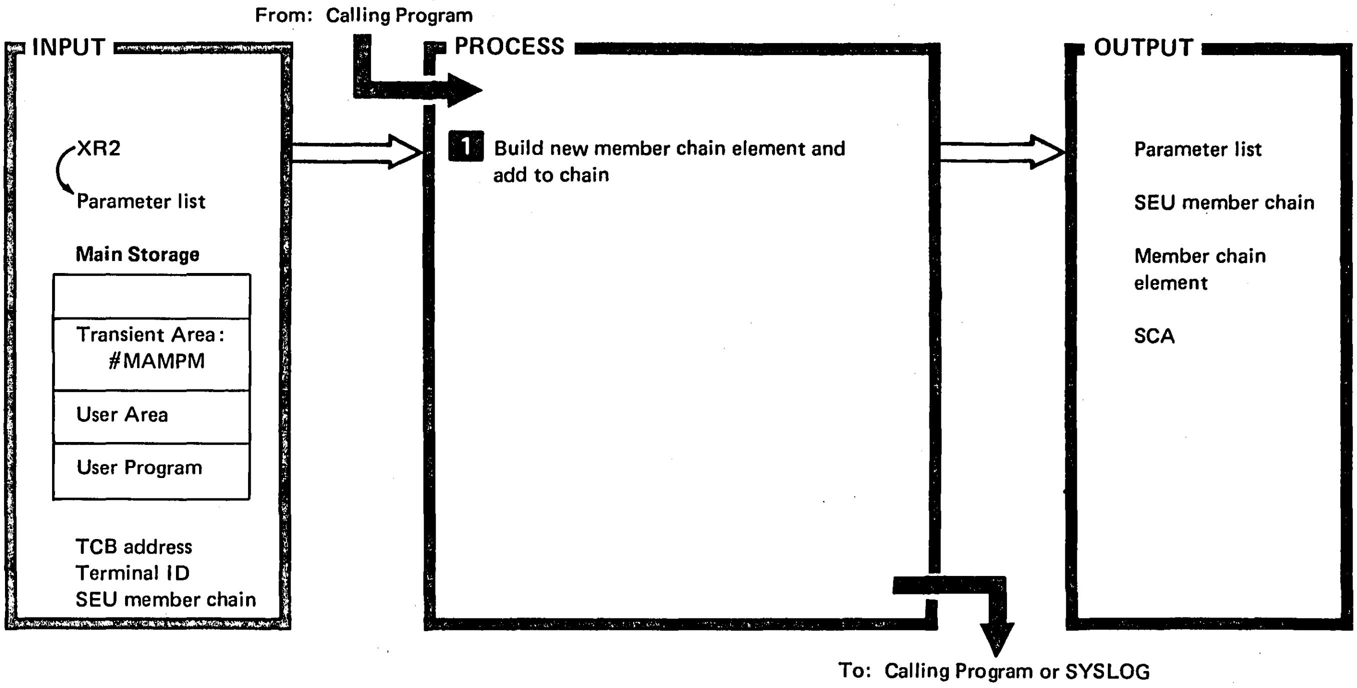
DESCRIPTION	MODULE/ ROUTINE
1 Check library control block (LCB) for library open request.	$MAPUR
If open request, open requested library and return to caller.	$MACOM or $MALCO
Compress record and move record from user's buffer to I/O buffer.	$MAPUR
When I/O buffer full, write buffer to requested library.	Disk IOS
Update LCB to indicate number library member sectors available.	$MAPUR
If not enough library member sectors available, issue appropriate message.	#CLXS
If close request, close library to update library directory and return to caller.	$MACOM or $MALCO

Diagram 6.5. Put Record to Library



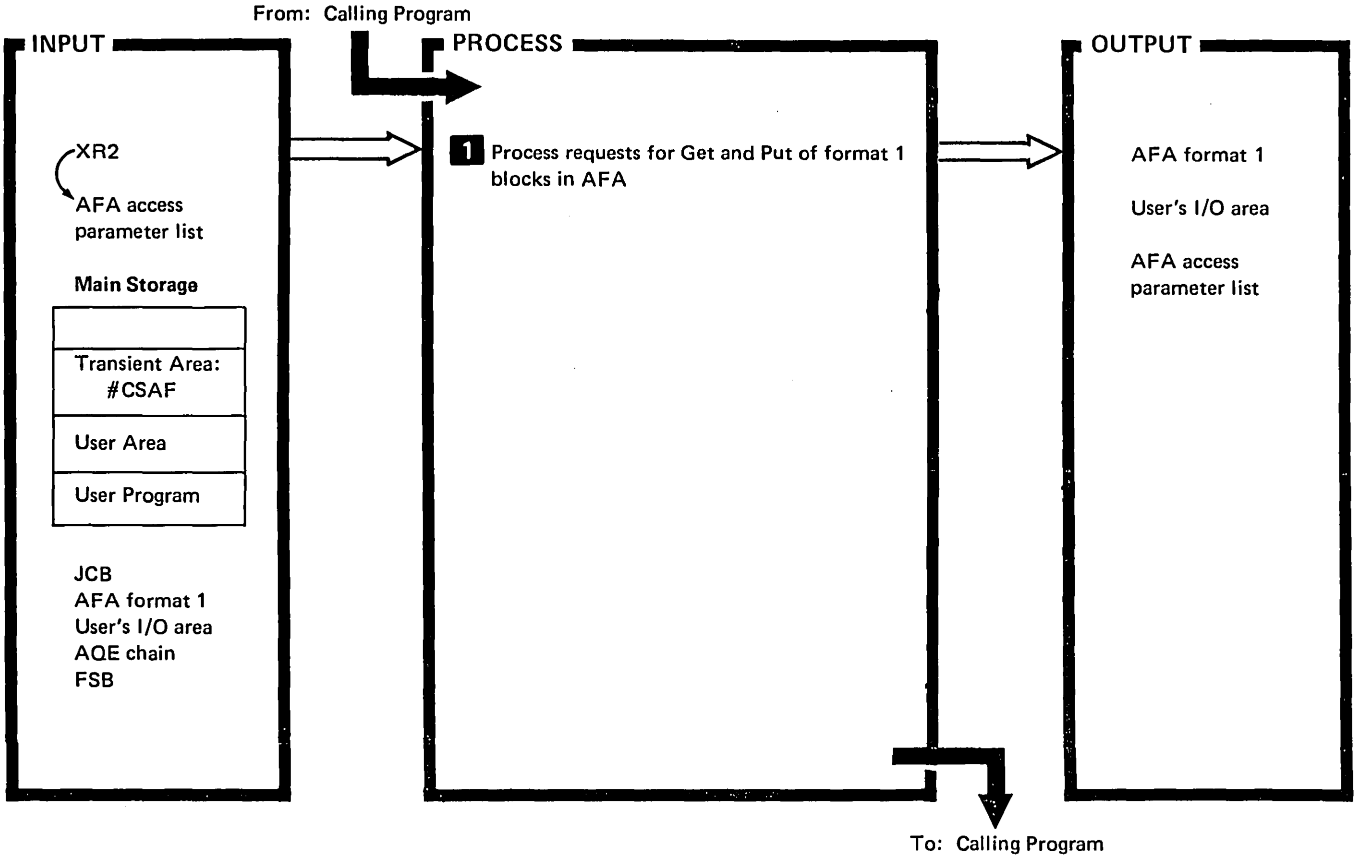
DESCRIPTION	MODULE/ ROUTINE
1 Check get/put bit (LCBGTPT) to determine operation requested. If put request, go to 2.	$MAPGS
If requested, find requested library member.	#MASFN
If program temporary fix (PTF) added to module, retrieve PTF information.	$MAPTF
Read requested library member and place in user provided I/O buffer.	Disk IOS
Return.	$MAPTF
▶ When entire member is retrieved: • Update library control block (LCB). • Return to calling program.	$MAPGS ◀
2 If open bit (LCBOPEN) on, open requested library and return to caller.	#MACOM or $MALCO
Write library member from user provided I/O buffer to specified library.	Disk IOS
When entire member written to library: • Close library if requested.	$MACOM or $MALCO
▶ • Update LCB. • Return to calling program.	$MAPGS ◀

Diagram 6.6. Perform Library Sector Get/Put Function



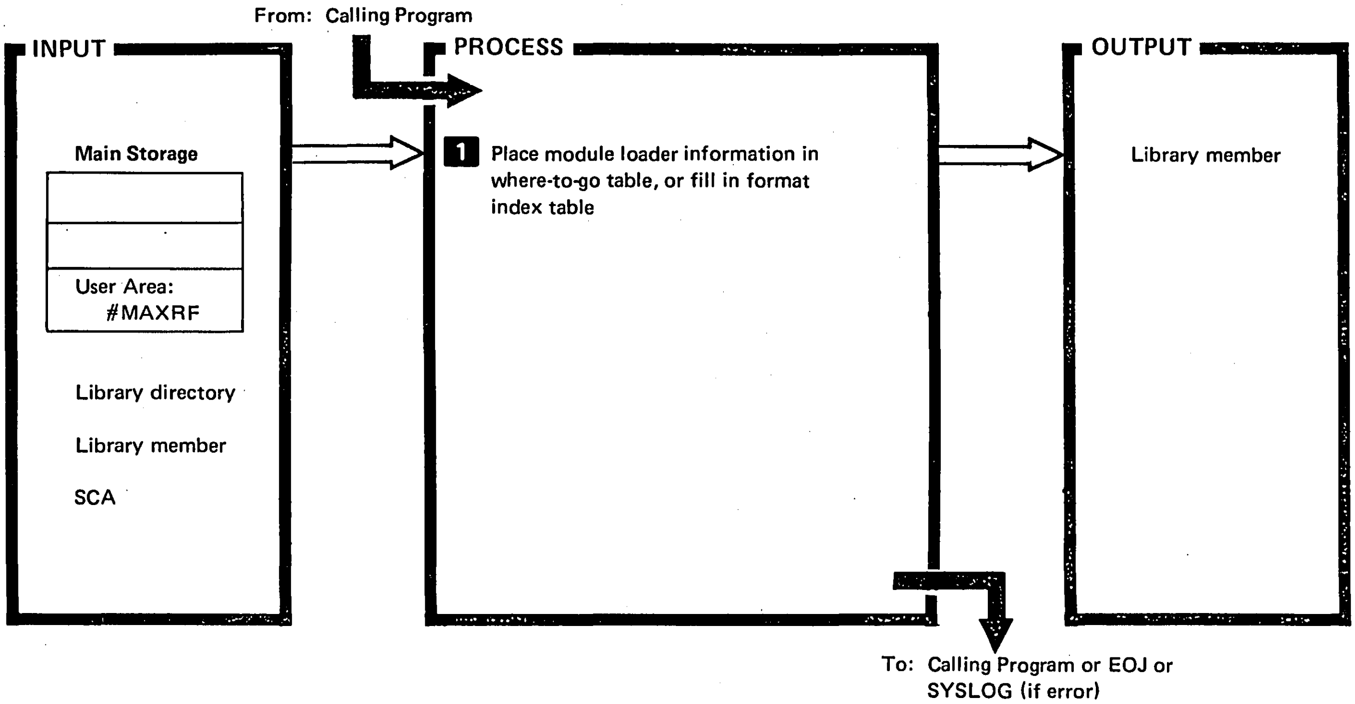
DESCRIPTION	MODULE/ ROUTINE
1 Enqueue source entry utility (SEU) member chain queue header. Add new member element to chain.	#MAMPM
Using parameter list information, search specified library for requested member.	#MASFN
Build member chain element from: ○ Task control block (TCB) address. ○ Terminal ID in terminal unit block (TUB). ○ If #MASFN found specified member, library directory member address (SSS). ○ If #MASFN did not find specified member, member address of zeros. Add terminal ID to end of parameter list. Search member chain for matches on terminal ID and member address.	#MAMPM
If matches (except for member address of zeros), cancel job and display appropriate error messages.	#CLXS
If no matches, dequeue SEU member chain queue header and return to calling program.	#MAMPM

Diagram 6.7. Perform Library Member Protection Function



DESCRIPTION	MODULE/ ROUTINE
1 Determine function requested by. caller. If get-by-label request for unit F1: ● Search AFA chain for format 1 with specified label (date and ID may also be given). ● Move format 1 to caller's I/O area. If get-by-label request for unit I1: ● Scan FSB chain and examine the associated format 1's. ● If format 1 contains correct unit and label, place format 1 address into caller's parameter list (date verify may also be given). ● Move format 1 to caller's I/O area. If get-by-name request: ● Search FSB chain for specified name (FSB contains address of format 1). ● If format 1 contains correct unit, place format 1 address into caller's parameter list. ● If requested, move format 1 to caller's I/O area. If get-by-address request, move format 1 at specified address to caller's I/O area. If put request, move format 1 in caller's I/O area into AFA format 1 specified in caller's parameter list. Update return code in parameter list. Return to calling program.	#CSAF

Diagram 6.8. Perform Active Format 1 Area Access Function

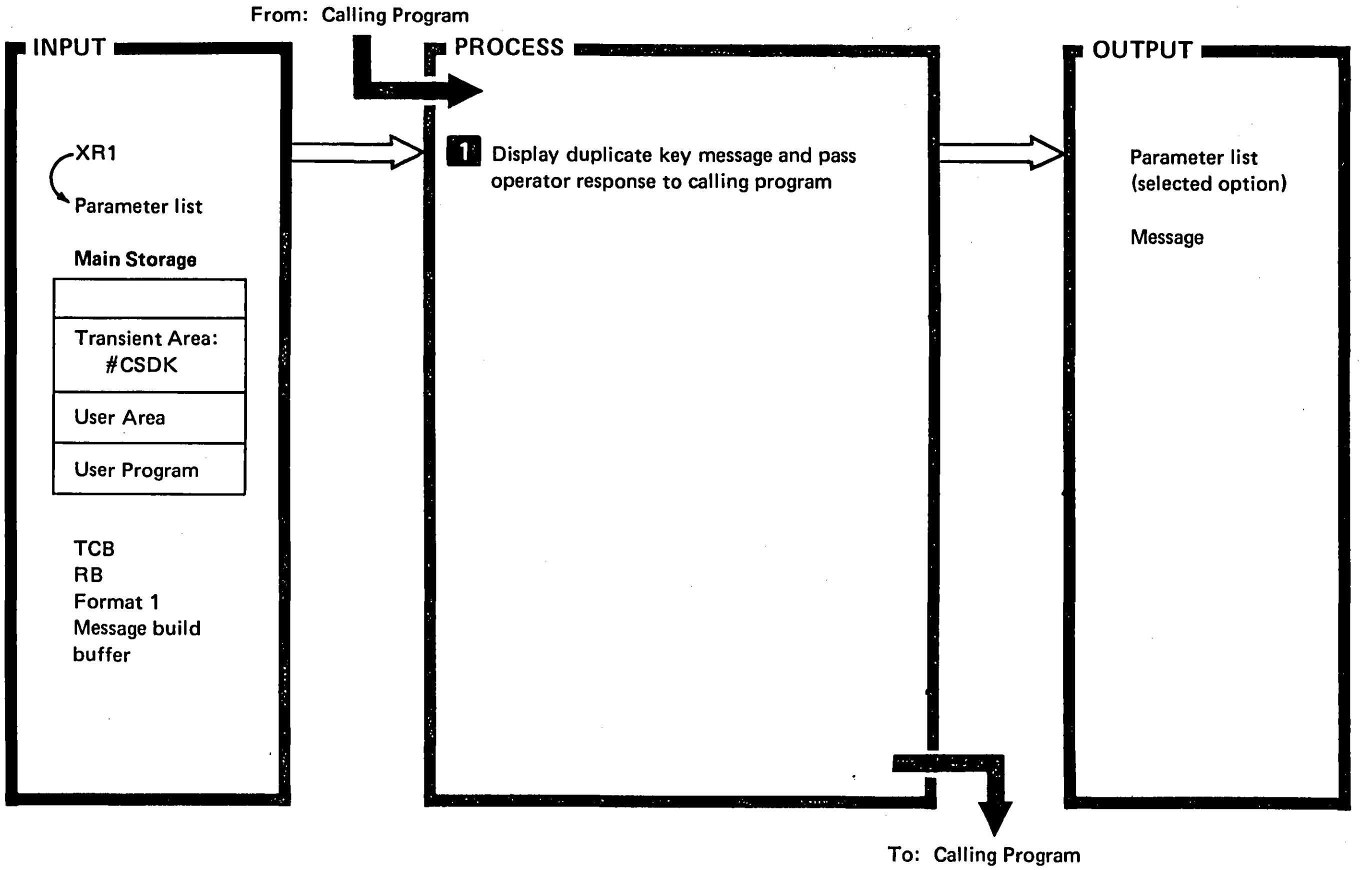

From: Calling Program
INPUT
Main Storage
User Area:
#MAXRF
Library directory
Library member
SCA
PROCESS
1 Place module loader information in where-to-go table, or fill in format index table
OUTPUT
Library member
To: Calling Program or EOJ or SYSLOG (if error)

DESCRIPTION	MODULE/ ROUTINE
1 Get address of #LIBRARY from system communication area (SCA).	#MAXRF
Read system library directory.	Disk IOS
Build resident table of all SSP load modules (O-type). Build main storage format index.	#MAXRF
For each entry in the resident table with the where-to-go (WTG) or format index table indicator on, read the last four text sectors of corresponding library member.	Disk IOS
If module has a WTG table, search resident table for match with WTG table entries. If match found, move loader data from resident table into WTG table: ● SSS disk address. ● Number of text sectors. ● RLD displacement. If no match, move loader data for no-op module (#MANOP) into WTG table (see note). If module has a format index table: ● Use displacement to get corresponding entry from the main storage format index. ● Move fields from main storage format index to the module's format index table: – SSS disk address. – Number of FDT. – Number of text sectors. – Input length of screen. *Note:* #MANOP is executed when unresolved module is called. #MANOP issues halt by way of SYSLOG.	#MAXRF

Diagram 6.9 (Part 1 of 2). Perform Cross Reference Resolver Function

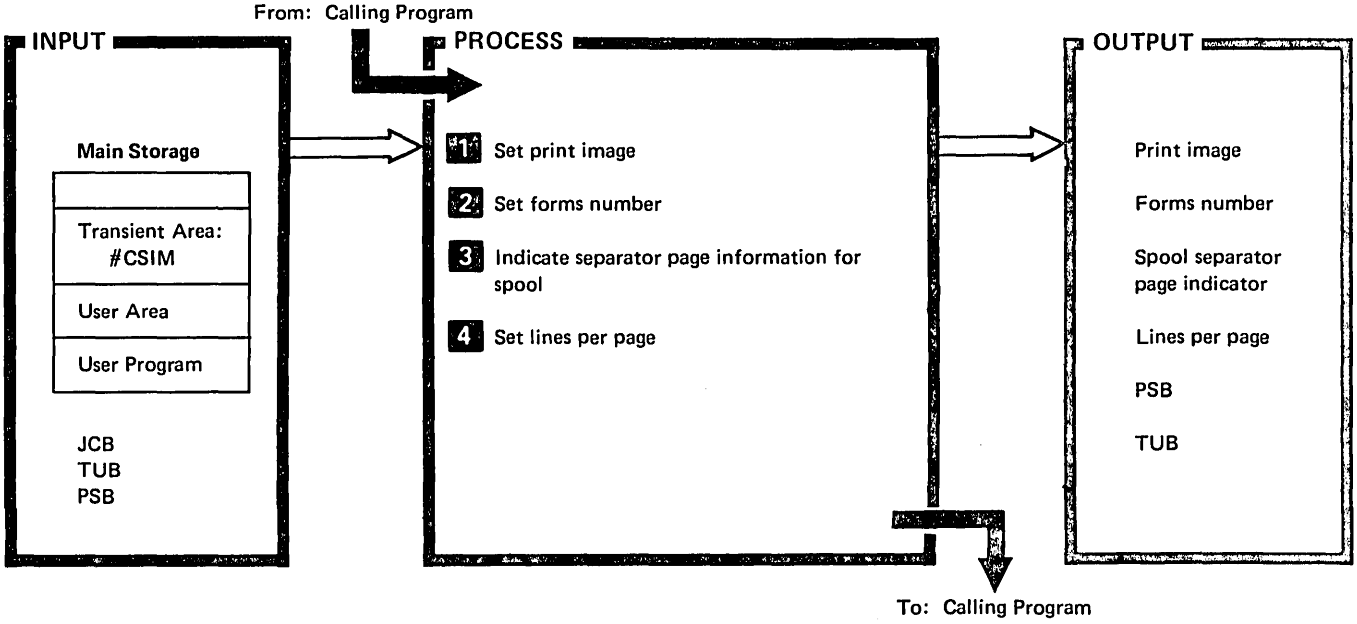
DESCRIPTION	MODULE/ ROUTINE
If WTG table or format index table is invalid (no delimiter of X'FFFF':	#MAXRF
• Display module name.	#CLXS
• Prepare to read next module. • If during IPL, issue system error.	#MAXRF
When all WTG table references and format index table entries are processed, write last four sectors of module back to original location on disk unless no changes were made.	Disk IOS
When all modules processed, return to calling program or EOJ as requested.	#MAXRF

Diagram 6.9 (Part 2 of 2). Perform Cross Reference Resolver Function



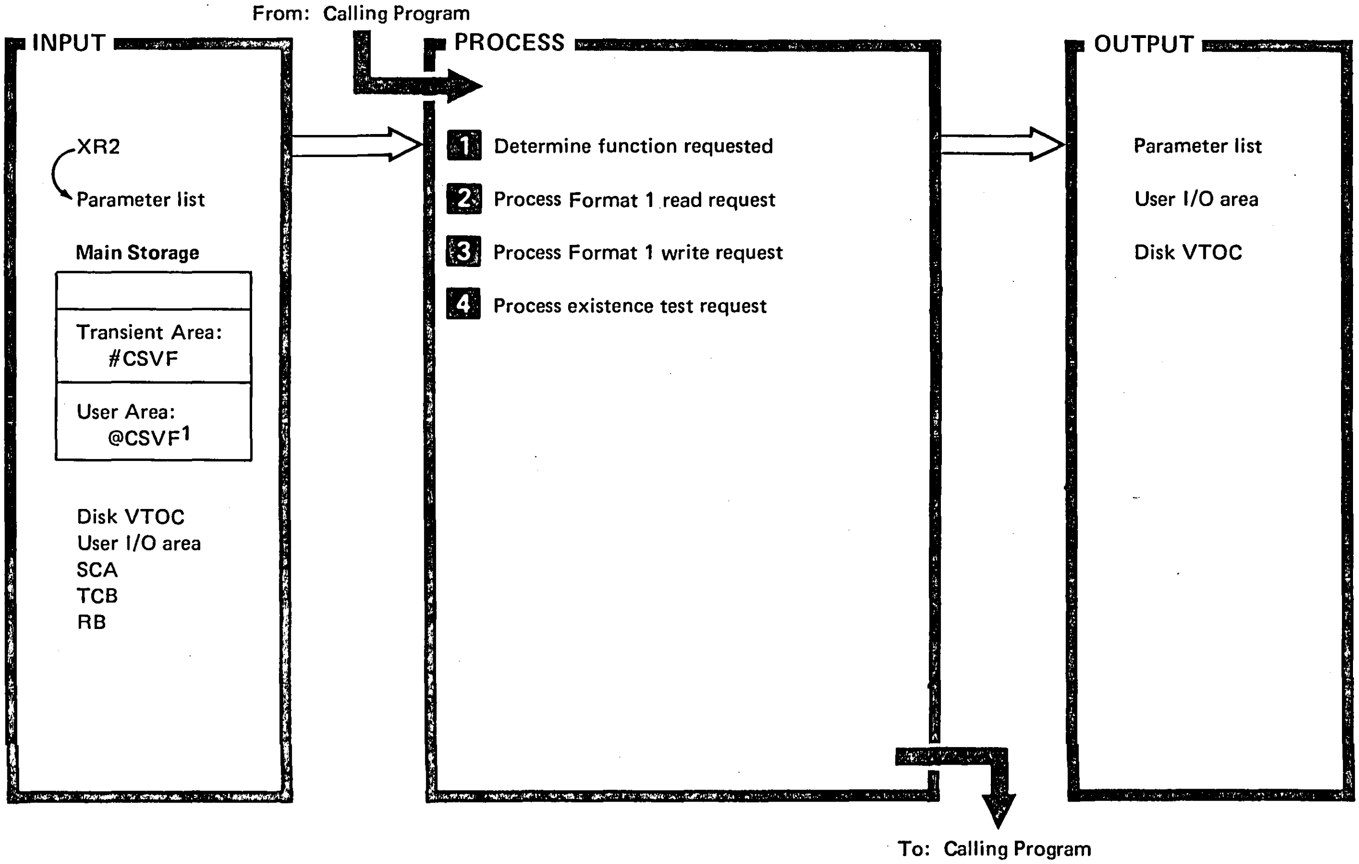
DESCRIPTION	MODULE/ ROUTINE
1 Save duplicate key display parameter list address and program mode register setting.	#CSDK
Display duplicate key found message and specify file name. Place operator selected option in SYSLOG parameter list.	#CLXS
Retrieve operator selected option from SYSLOG parameter list: ● 0 option = display key in EBCDIC. ● 1 option = display key in hexadecimal notation. Move key to message buffer area (if hexadecimal notation request, convert first).	#CSDK
Output duplicate key message to system console. Place operator selected option in SYSLOG parameter list.	#CLXS
Retrieve operator selected option from SYSLOG parameter list and place in duplicate key message parameter list (flag byte). Return to calling program.	#CSDK

Diagram 6.10. Perform Duplicate Key Display Function



DESCRIPTION	MODULE/ ROUTINE
1 Read work station configuration record. Read current printer image.	Disk IOS
Compare current print image to work station configuration record.	#CSIM
If images do not match: ● Output change printer image message to operator. ● Place operator selected option in SYSLOG parameter list.	#CLXS
If option 1 (image change request), change image in printer control unit. ● Make current printer image equal to work station config record.	#CSIM WSIOCH
2 Compare forms number in printer specification block (PSB) and terminal unit block (TUB).	#CSIM
If forms numbers do not match: ● Output change forms message to operator. ● Place operator selected option in SYSLOG parameter list.	#CLXS
If option 1 (forms number change), set forms number in TUB. **3** Check spool indication in TCB.	#CSIM
If spool: ● Output spool separator page message to operator. ● Place operator selected option in SYSLOG parameter list.	#CLXS
If option 1 (separator pages requested), update PSB flag byte. **4** Check lines per page flag in PSB.	#CSIM
If lines per page flag on, send lines per page to printer control unit.	WSIOCH
After all PSBs on chain are processed, return to calling program.	#CSIM

Diagram 6.11. Perform Printer Image Verify Function



From: Calling Program

INPUT

XR2

Parameter list

Main Storage

Transient Area:
#CSVF

User Area:
@CSVF[1]

Disk VTOC
User I/O area
SCA
TCB
RB

PROCESS

1 Determine function requested

2 Process Format 1 read request

3 Process Format 1 write request

4 Process existence test request

OUTPUT

Parameter list

User I/O area

Disk VTOC

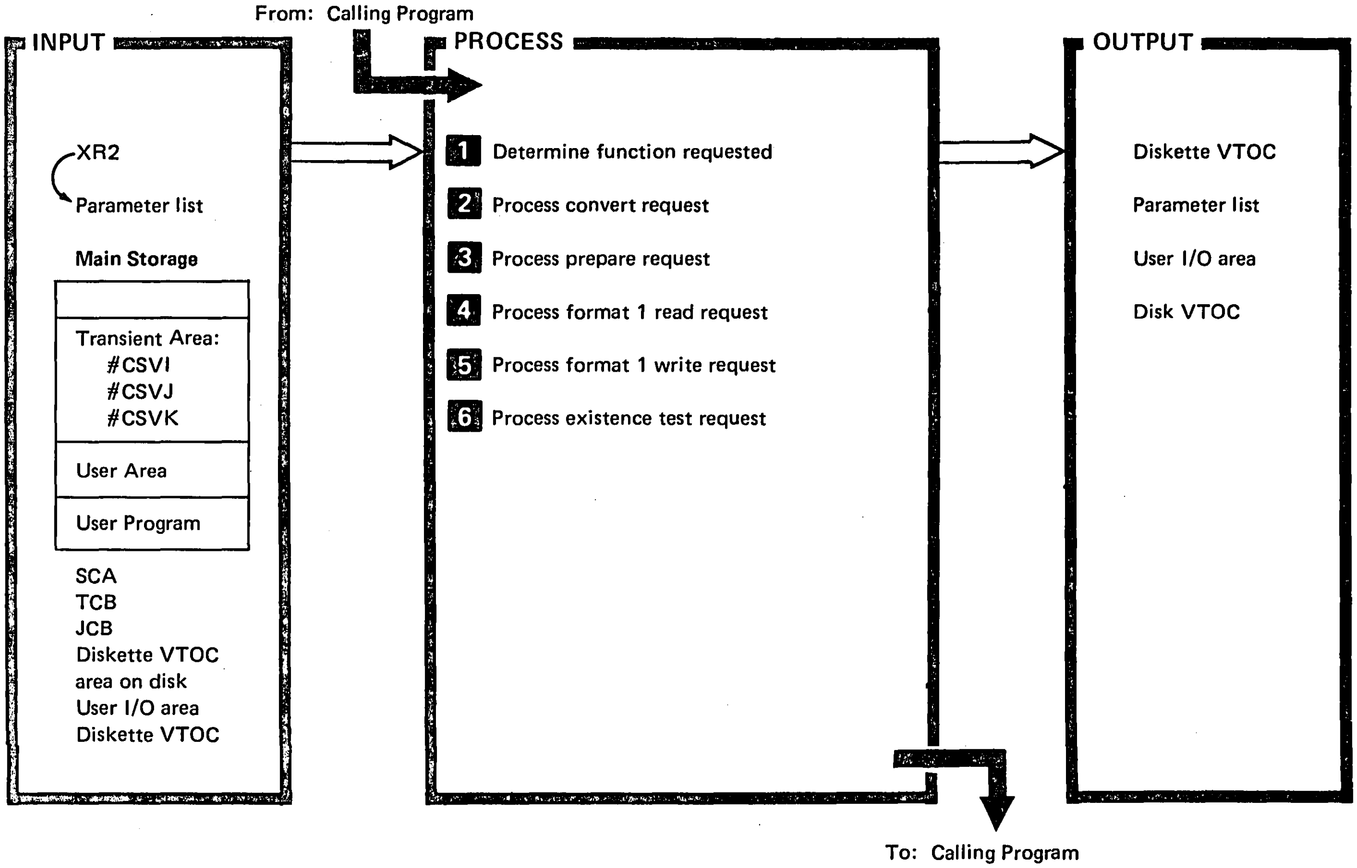
To: Calling Program

DESCRIPTION	MODULE/ ROUTINE
1 Examine function byte in parameter list: ● If format 1 read request, go to **2**. ● If format 1 write request, go to **3**. ● If existence test request, go to **4**. **2** Determine type of format 1 read request: ● Read next: — Examine parameter list to determine if this is first request.	#CSVF[1]
— If first request, start scan at first VTOC format 1 area sector. — If not first request, increment sector number in parameter list and start scan in next sector.	Disk IOS
— Read sector identified by disk IOS. ● Read next-same label: — Set up scan mask to compare on specified label. — Perform same steps as for read next. ● Read specific: — Set date indicator scan if requested.	#CSVF
— Scan VTOC format 1's.	Disk IOS
If requested VTOC format 1 found or read by sector displacement: ● Place sector/displacement of format 1 in parameter list. ● Read format 1 into data area. ● Move format 1 to caller's I/O area. [1] Link-edit version is @CSVF.	#CSVF

Diagram 6.12 (Part 1 of 2). Process Disk VTOC Read/Write Request

DESCRIPTION	MODULE/ ROUTINE
If request not found, set request not met return code in parameter list.	#CSVF
Return to calling program.	
3 Set up IOB to read sector specified in parameter list.	
▶ Read specified sector from disk.	Disk IOS
Move format 1 from caller's I/O area into sector just read.	#CSVF
Write sector with updated format 1 back to disk.	Disk IOS
Return to calling program.	#CSVF
4 Perform read specific processing but do not move format 1 into caller's I/O area.	
Return to calling program.	

Diagram 6.12 (Part 2 of 2). Process Disk VTOC Read/Write Request

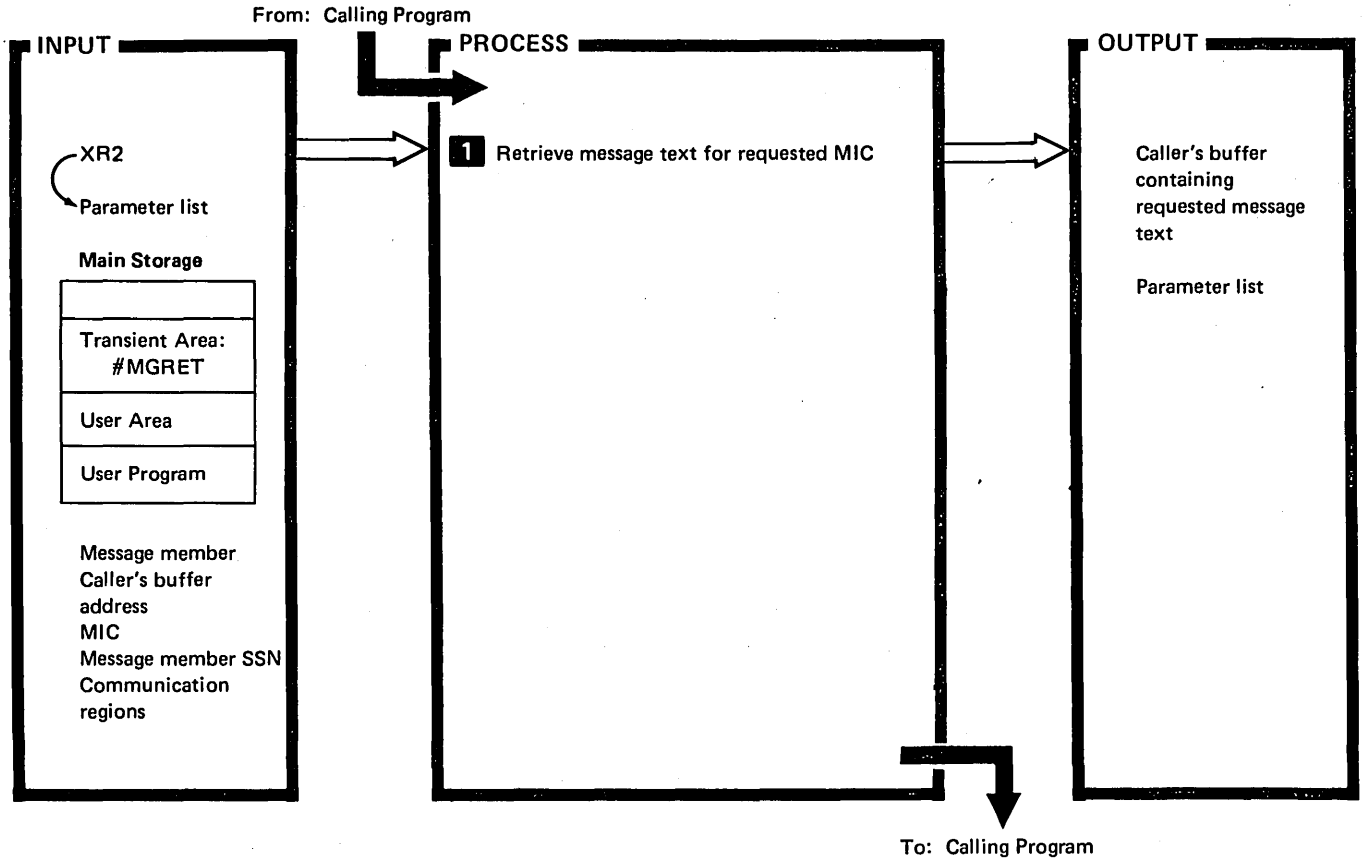


DESCRIPTION	MODULE/ ROUTINE
1 Examine function byte in parameter list:	#CSVI
• If convert request, go to **2**.	
• If prepare request, go to **3**.	
• If format 1 read request, go to **4**.	
• If format 1 write request, go to **5**.	
• If existence test request, go to **6**.	
2 If request for header 1's convert:	
• Initialize diskette IOB for read operation and disk IOB for write operation.	#CSVJ
• Read header 1's from diskette.	Diskette IOS
• Convert header 1's to format 1's.	#CSVJ
• Write format 1 to disk.	Disk IOS
• Return.	#CSVJ
If request for format 1 convert:	#CSVI
• Verify lock sector.	#CSVM
• Initialize disk IOB for read operation, and diskette IOB for write operation.	#CSVK
• Read format 1 from diskette format 1 area on disk.	Disk IOS
▷ • Convert format 1 to header 1.	#CSVK ◀
• Write header 1 to diskette.	Diskette IOS
• Return.	#CSVK

Diagram 6.13 (Part 1 of 2). Process Diskette VTOC Read/Write Requests

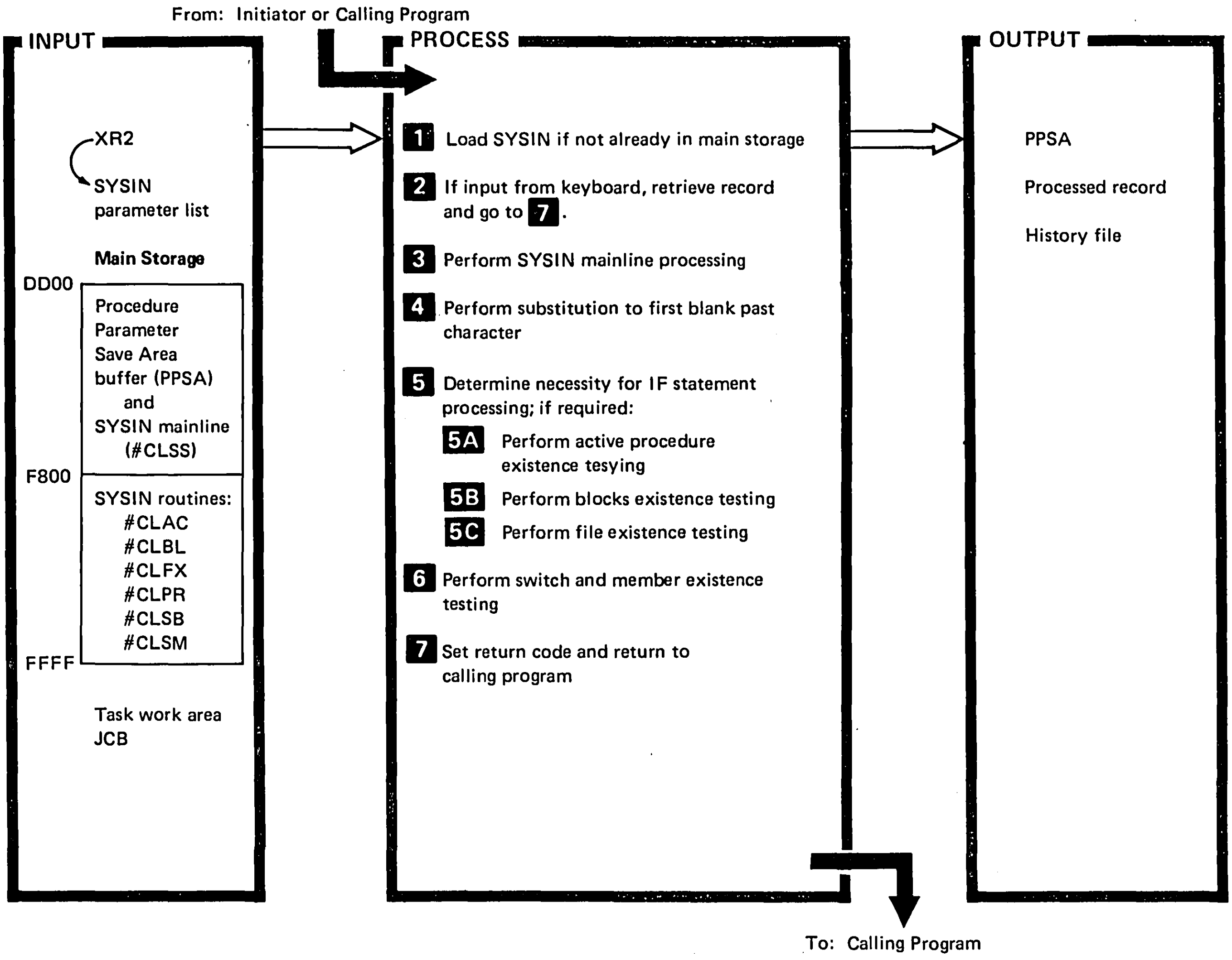
DESCRIPTION	MODULE/ ROUTINE
3 Initialize diskette IOB for read operation.	#CSVI
Read diskette volume label.	Diskette IOS
Perform recalibrate if necessary.	
Verify volume label format.	#CSVI
Put volume label and physical attribute byte in SCA.	
Create diskette lock sector.	
Build diskette VTOC area on disk.	#CSVJ
4 Determine type of format 1 read request:	#VSVI
● Read next:	
– Examine parameter list to determine if this is first request.	
– If first request, start scan at first VTOC format 1 area sector.	Disk IOS
– If not first request, increment sector number in parameter list and start scan in next sector.	
● Read next-same label:	#CSVI
– Set up scan mask to compare on specified label.	
– Perform same steps as for read next.	
● Read specific:	
– Set date indicator scan if requested.	
– Scan VTOC format 1's.	Disk IOS
If requested VTOC format 1 found or read by sector displacement:	#CSVI
● Place sector/displacement of format 1 in parameter list.	
● Read format 1 into data area.	Disk IOS
● Move format 1 to caller's I/O area.	#CSVI
If request not found, set request not met return code in parameter list.	
Return to calling program.	
5 Set up IOB to read sector specified in parameter list.	
Read specified sector from disk.	Disk IOS
Move format 1 from caller's I/O area into sector just read.	#CSVI
Write sector with updated format 1 back to disk.	Disk IOS
Return to calling program.	#CSVI
6 Perform read-specific processing but do not move format 1 into caller's I/O area.	
Return to calling program.	

Diagram 6.13 (Part 2 of 2). Process Diskette VTOC Read/Write Requests



DESCRIPTION	MODULE/ ROUTINE
1 Ensure valid message member: • Check message retrieve parameter list for valid indicators. • Check message member pointers in appropriate communication region for nonzero SSN.	#MGRET
Scan message member for sector identified by message identification code (MIC) greater than or equal to one requested.	Disk IOS
Find desired message by reading message member sector, looking for requested MIC. Blank caller's buffer. Place requested message in buffer. Place message length in parameter list. If error condition detected, return error MIC in parameter list. Return control to calling program.	#MGRET

Diagram 6.14. Perform Message Retrieve Function



From: Initiator or Calling Program

INPUT

XR2

SYSIN parameter list

Main Storage

DD00

Procedure Parameter Save Area buffer (PPSA) and SYSIN mainline (#CLSS)

F800

SYSIN routines:
#CLAC
#CLBL
#CLFX
#CLPR
#CLSB
#CLSM

FFFF

Task work area
JCB

PROCESS

1 Load SYSIN if not already in main storage

2 If input from keyboard, retrieve record and go to **7**.

3 Perform SYSIN mainline processing

4 Perform substitution to first blank past character

5 Determine necessity for IF statement processing; if required:

5A Perform active procedure existence tesying

5B Perform blocks existence testing

5C Perform file existence testing

6 Perform switch and member existence testing

7 Set return code and return to calling program

OUTPUT

PPSA

Processed record

History file

To: Calling Program

DESCRIPTION	MODULE/ ROUTINE
1 Push out one track of user main storage to make room for #CLSS and 2K area for SYSIN routines. Load SYSIN mainline (#CLSS) into main storage area just cleared (X'DD00'). Pass control to #CLSS.	#CLSN
2 Call command processor (#CMWI) to accept input from keyboard. If log to printer, call SYSLOG (#CLSG) to print keyboard input. Move input record to user's buffer.	#CLSS
Output records to history file if log bit on in SYSIN parameter list operation code.	@HFPUT
Pull user program back into main storage if previously pushed out to disk. Go to **7**.	#CLSN

Diagram 6.15 (Part 1 of 4). Perform SYSIN Function

DESCRIPTION	MODULE/ ROUTINE
3 Call source get (#MASYG) to retrieve record from procedure library and place in input buffer. Move characters one at a time from input buffer to output buffer looking for question mark (?). If ? found, perform character substitution by checking proper syntax. If syntax error, call SYSLOG (#CLXS).	#CLSS
Determine type substitution required: ● Type 1 ?#? ● Type 2 ?#R? ● Type 3 ?#R'msg id'? ● Type 4 ?#'default'? ● Type 5 ?#T'default'? ● Type 8 ?R? ● Type 6 ?WS? ● Type 7 ?L'dsp,lng'? ● Type 9 ?Mxxxx? ● Type 10 ?M'xxxx,dsp,lng'? If substitution required, go to 4. Look for // blank in each procedure statement. A If // blank found, determine keywork: ● IF. ● ELSE. ● CANCEL. ● RETURN. ● RESET. ● Nonkeyword. *IF statement* — If the IF expression conditions are met, blank out IF keyword and expression and continue processing. If expression not met, set switch TSSSW4 and read another record into input buffer. Search character string for / or — character: ● If /, perform character string comparison. ● If —, perform existence testing: — For active procedure existence test, go to 5A. — For blocks existence test, go to 5B. — For file existence test, go to 5C. — For switch and member existence test, go to 6. *ELSE keyword* — If previous IF statement invalid (check TSSSW4), blank out ELSE keyword and process record. Otherwise, read next record and ignore entire record with ELSE. *CANCEL keyword* — If IF statement just processed (check TSSSW5) set TSSSW6 which causes EOJ transient ($EJ1-control storage) to be called. *RETURN keyword* — If IF statement just processed, (check TSSSW5) set TSSSw7 which causes same function as end-of-file processing. *RESET keyword* — Set TSSSW14. If TSSSW5 on, IF statement just processed. Since another IF expression may follow, return to 3 A to continue processing. When all IF expressions processed, go to 6 to substitute remainder of record.	#CLSS

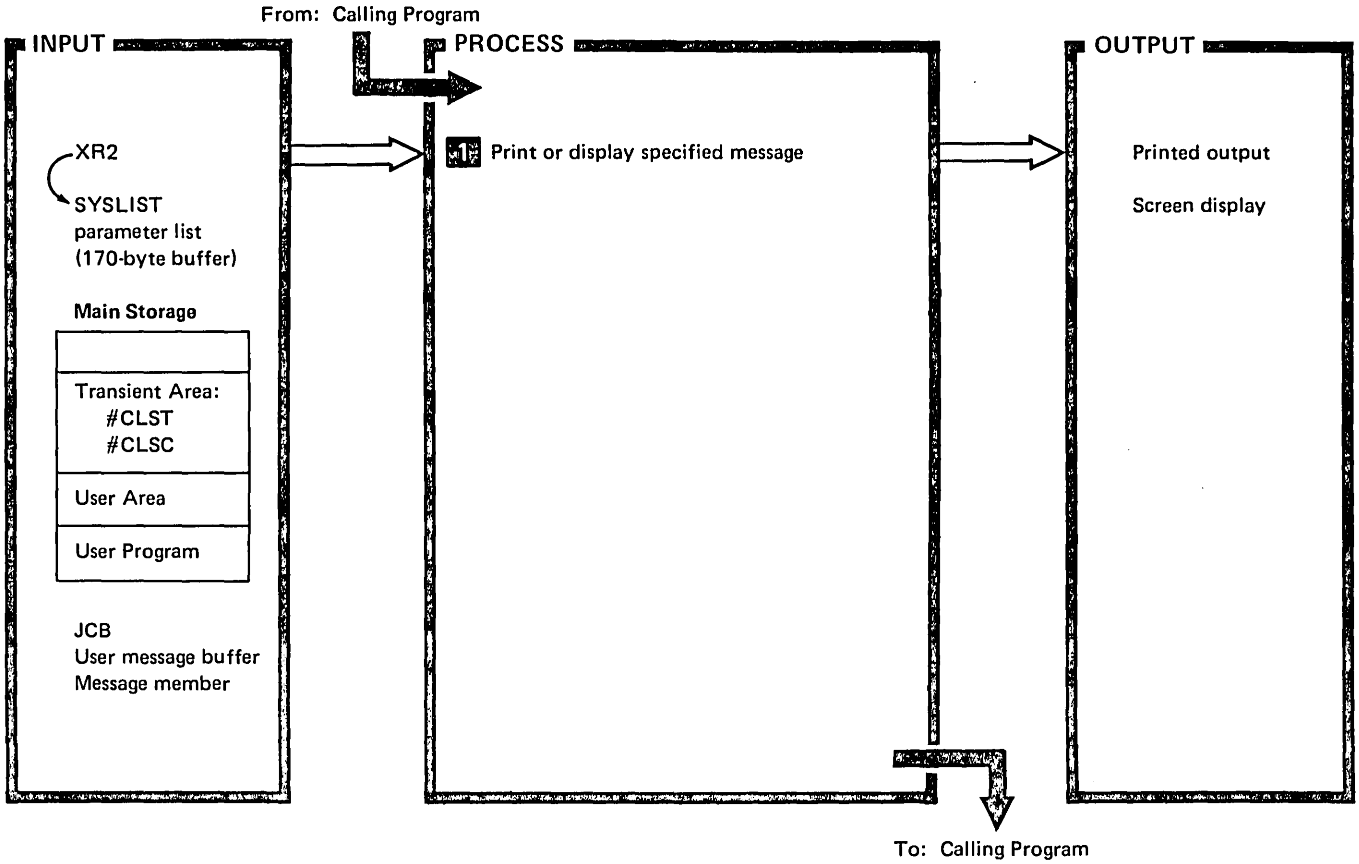
Diagram 6.15 (Part 2 of 4). Perform SYSIN Function

DESCRIPTION	MODULE/ ROUTINE
When record with substitution, IF, and ELSE processing complete and in user's output buffer. ● Write record to history file if log bit on in SYSIN parameter list (use @HFPUT which is link edited with #CLSS).	#CLSS
● Pull user program back into main storage.	#CLSN
● Return control to calling program.	#CLSS
4 Perform actual substitution: ● *Type 1* — parameter passed to procedure indicated by number is substituted. If no parameter passed, pass null to output buffer. ● *Type 2* — R indicates parameter required. Thus if no parameter passed, operator prompt is ENTER MISSING PARAMETER. Response is saved in procedure save area as parameter (up to 8 characters). ● *Type 3* — R indicates parameter required. User message is displayed to operator. Response saved in procedure save area as parameter. ● *Type 4* — Default value enclosed in quotation marks used as parameter. ● *Type 5* — Default value used but T indicates parameter is temporary. Procedure parameter save area not updated with default value. ● *Type 8* — Prompt for up to 8 characters but result not saved in PPSA.	#CLPR
● *Type 6* — Substitute work station ID. ● *Type 7* — Substitute requested displacement and length from local area. ● *Type 9* — Substitute characters from user 1 message member indicated by MIC number XXXX. ● *Type 10* — Substitute from MIC with specified displacement and length.	#CLSB
If non blank character found, set internal switch. If blank found, nonblank switch on, and substitution only to first blank past character, return to caller. Otherwise, move character to output buffer. Repeat until all characters substituted.	#CLSS
When record substituted, output to history file.	@HFPUT
5 When IF statement processing necessary, perform steps 5A , 5B , and 5C as required. When IF statement processing not necessary, perform substitution to end of statement. When IF statement evaluated, return to 4 .	#CLSS
5A Check for quote mark following dash. If quote found, there is list of procedure names to check. If no quote, only one procedure name. Move procedure name characters to scan mask (8 characters maximum). If active procedure found, set SS5TRVE switch. If error detected, call SYSLOG (#CLXS) to display error message. Return to #CLSS on ARR.	#CLAC
5B Check for 1 to 5 valid digits after BLOCKS–. If digits valid, convert EBCDIC digits to hexadecimal notation. Set up disk IOB to read F5 area (first 5 sectors of VTOC).	#CLBL
Read format 5 area from disk.	Disk IOS
Check for available blocks. ● Save first format 5s size and address. ● Search format 5s for first format 5 that fits blocks requested. ● If space found, set SS5TRVE switch.	#CLBL

Diagram 6.15 (Part 3 of 4). Perform SYSIN Function

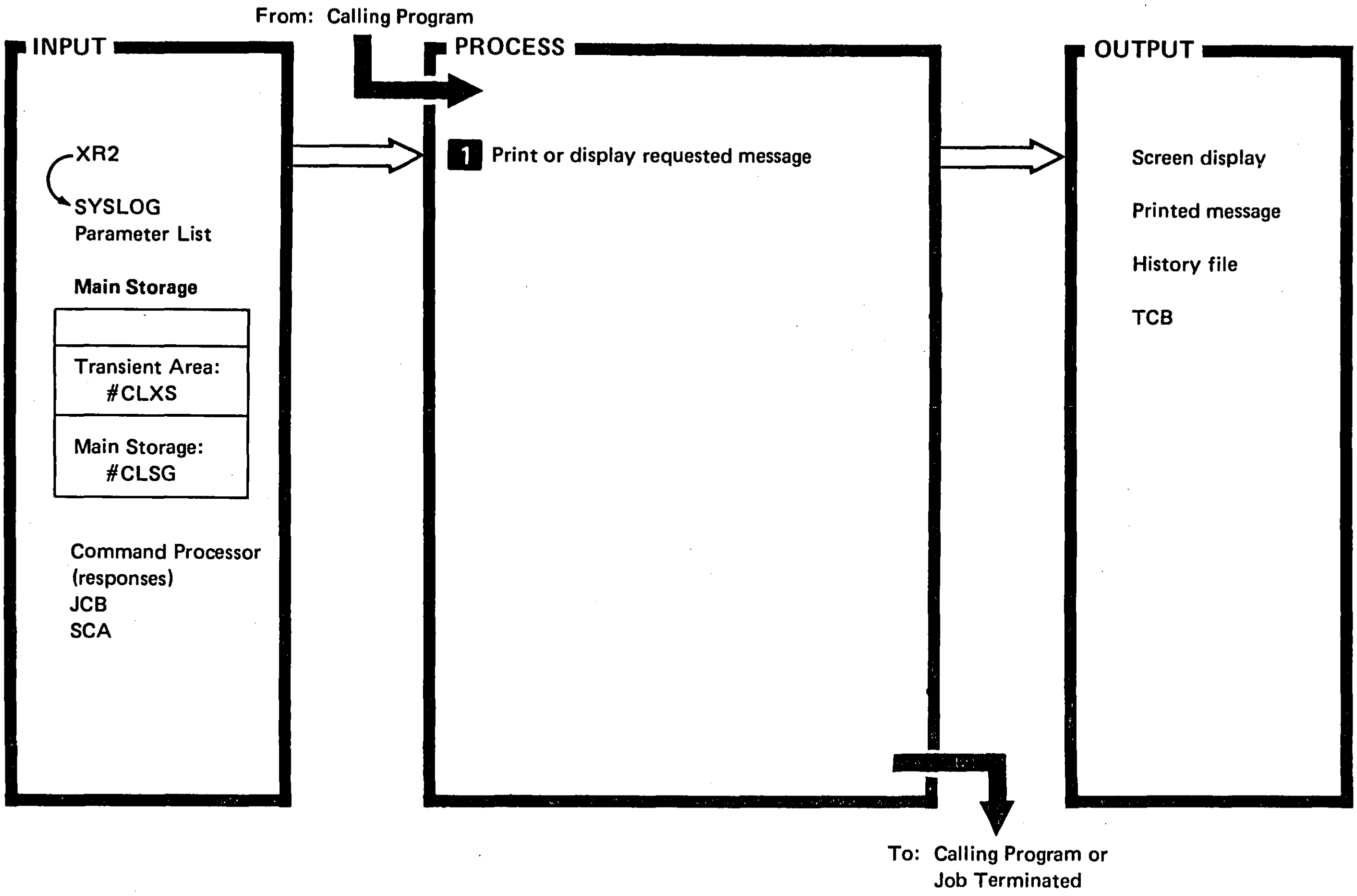
DESCRIPTION	MODULE/ ROUTINE
If error detected, call SYSLOG (#CLXS) to output error message. Return to #CLSS on ARR.	#CLBL
5C Check for quote mark following –. If quote found, indicates file existence test with file name and date. Perform syntax checking. Determine if disk or diskette to be checked. If diskette: ● Allocate diskette device. ● Call diskette VTOC read/write (#CSVI) to perform existence test. ● If specified file found, set SS5TRVE switch. ● Deallocate diskette device. If disk: ● Call disk VTOC read/write (#CSVF) to read VTOC. ● If specified file found, set SS5TRVE switch. If error detected, call SYSLOG (#CLXS) to display error message. Return to #CLSS on ARR.	#CLFX
6 Check for quote mark following –. If quote found, indicates library name follows library member name. Perform syntax checking.	#CLSM
Find specified library.	#MAFLB
If library found, find specified library member.	#MASFN
If library and member found, set SS5TRVE switch. If SWITCH keyword, perform switch existence test: ● Compare UPSI switch with SWITCH keyword. ● If compare ok, set SS5TRVE switch.. If error detected, call SYSLOG (#CLXS) to display error message. Return to #CLSS on ARR.	#CLSM
7 If successful operation set return code of X'40' in parameter list. If /* in first two positions of keyboard record, set return code of X'50' in parameter list. Return control to calling program.	#CLSS

Diagram 6.15 (Part 4 of 4). Perform SYSIN Function



DESCRIPTION	MODULE/ ROUTINE
1 Check SYSLIST indicator (JCBDSLST) and if null (X'0000'), return to calling program.	#CLST
If parameter list is type 1 (output from message member):	
○ Check parameter list for message member to use.	
○ Retrieve message from message member and place in SYSLIST message buffer (170 byte buffer supplied by the user).	#MGRET
If parameter list is type 2 (output from program), the user's message is already in the SYSLIST message buffer.	#CLST
Check SYSLIST indicator (JCBDSLST) to determine SYSLIST device.	
If SYSLIST device is printer (not X'0000' or X'EEEE'):	
▶ ● Allocate printer.	#CAPT ◀
○ Build print buffer from user supplied buffer.	#CLST
○ Skip to new page if requested in parameter list.	WSIOCH
○ Print message from print buffer.	
○ Space according to value in parameter list (0 to 3).	
● Skip to new page if within six lines of page size value in JCB (JCBDLNPG).	
○ Return.	#CLST
If SYSLIST device is display screen (X'EEEE'):	
○ Build command processor parameter list around message in SYSLIST buffer.	
○ Display message on screen.	#CMLS
Return control to calling program.	#CLST

Diagram 6.16. Perform SYSLIST Function

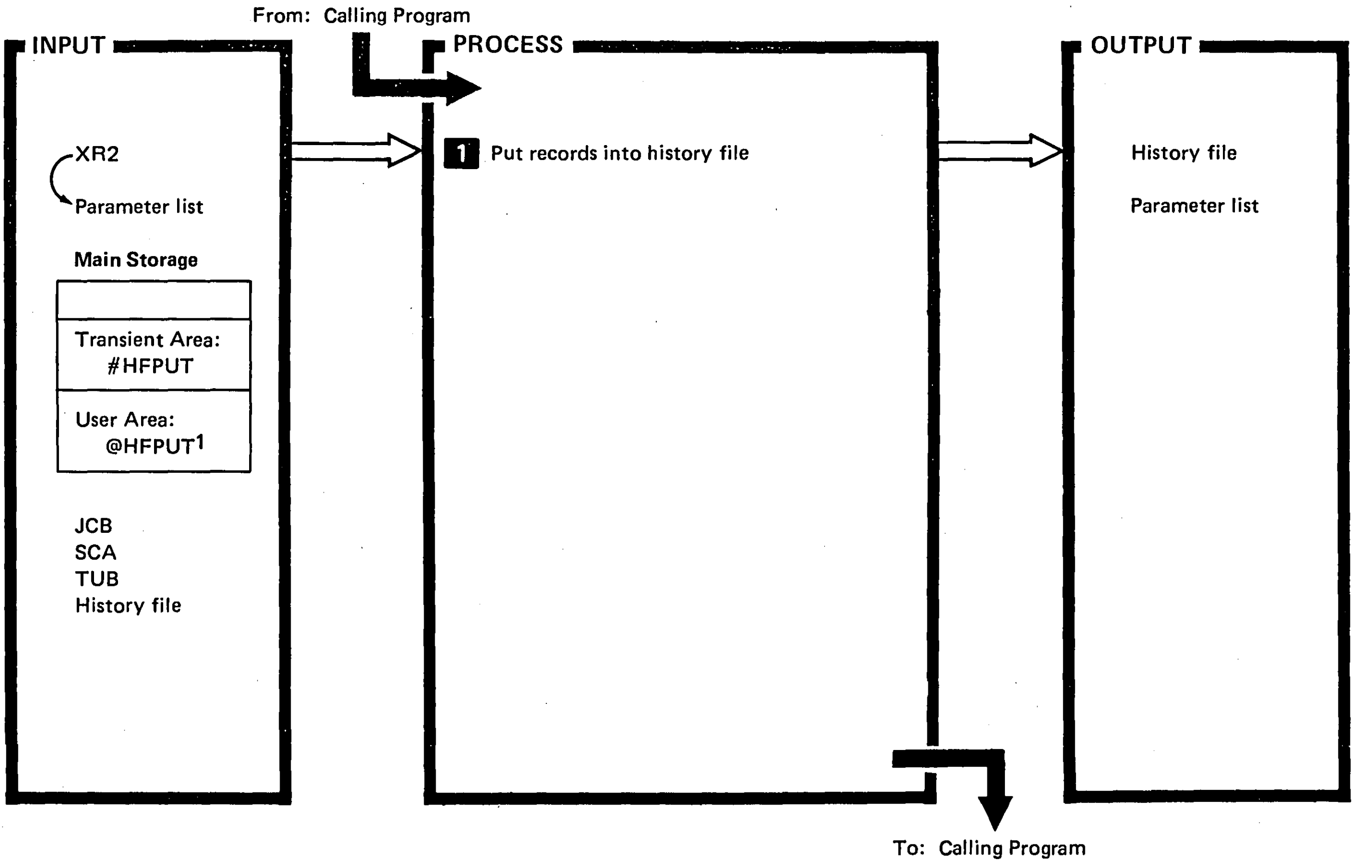


DESCRIPTION	MODULE/ ROUTINE
1 If SYSLOG push/pull transient called by user: ● Move SYSLOG parameter list from calling program to parameter list buffer in transient area. ● If type 2, 2R, or 3 message required, move message from calling program to message buffer in transient area. ● Push 10K of main storage (user area) to disk. ● Move parameter list buffer and message buffer (if used) to main storage area just cleared (X'C800' – X'C8FF'). ● Load SYSLOG mainline (#CLSG) into main storage area (at X'C900').	#CLXS
Build format line (if requested) and log to history file (history file put – @HFPUT link edited with #CLSG). If type 1, 1R, or 4 message, move message from message member (message retrieve – @MGRET link edited with #CLSG) to main storage save area. If type 2, 2R, or 3 message : ● Message is in main storage save area. ● Log message to history file (@HFPUT).	#CLSG
Display message (format line and text line) on system operator display screen or work station display screen as requested.	#CMWO and #CMCI
If system in single program mode and printer is SYSLOG device, also print message as well as display it.	WSIOCH
If data response required, log message and response to history file (@HFPUT).	#CLSG

Diagram 6.17 (Part 1 of 2). Perform SYSLOG Function

DESCRIPTION	MODULE/ ROUTINE
If option response to message: ● Display and print (if print conditions met) valid option taken by operator. ● If 3 option taken, call end of job transient (#CTEI). ● If D option taken, return to #CLXS. If SYSLOG push/pull transient called by #CLSG:	#CLSG
● Move SYSLOG parameter list from main storage to parameter list buffer in transient area. ● Pull main storage area back from disk. ● If option response taken by operator, pass response to calling program. ● If option response was D, dump main storage and call EOJ transient. ● If data response required, pass response to calling program. ● Return control to user program.	#CLXS

Diagram 6.17 (Part 2 of 2). Perform SYSLOG Function

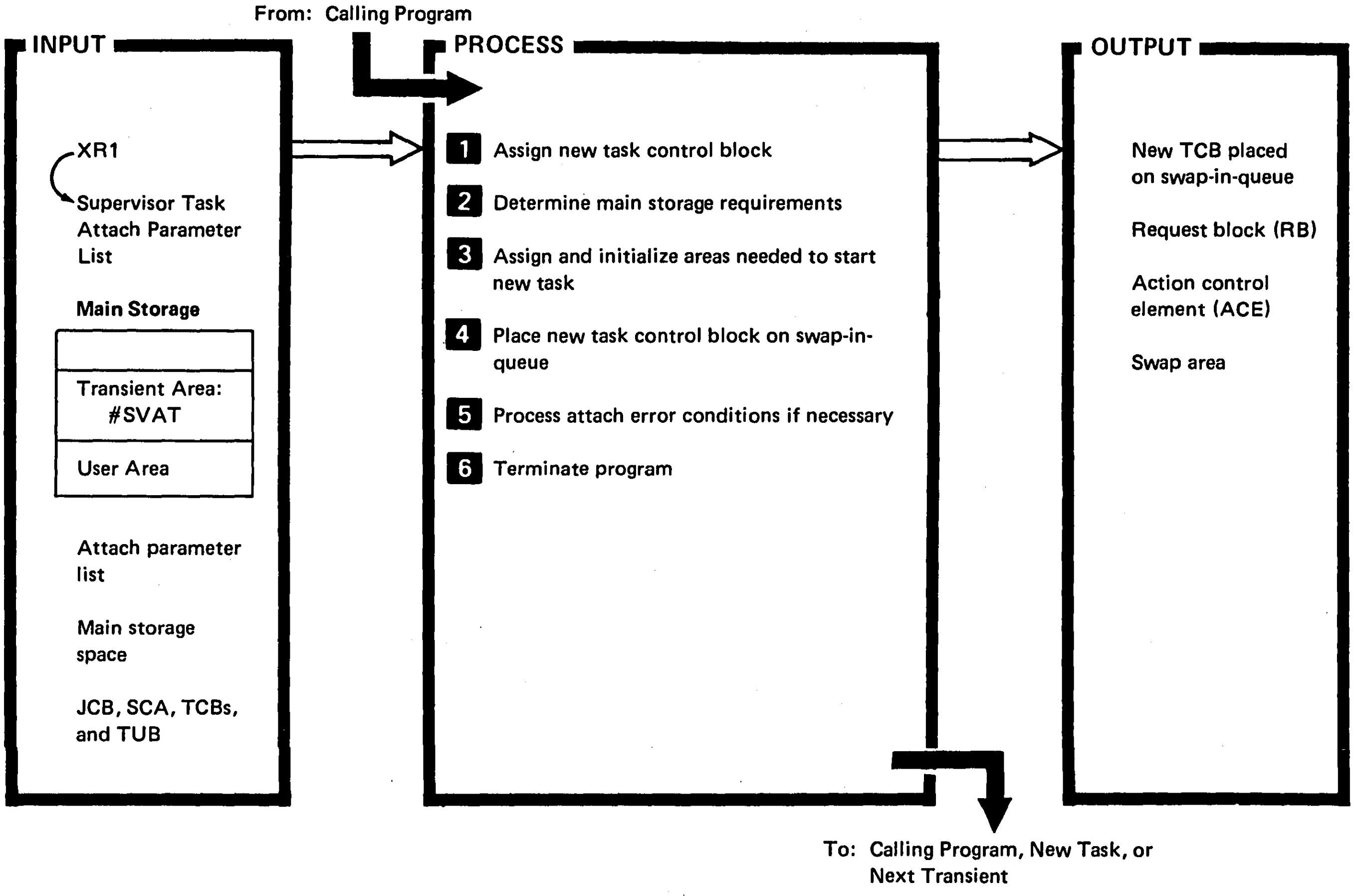


DESCRIPTION	MODULE/ ROUTINE
1 Lock the history file. Obtain history file controls from system communication area (SCA): ● History file start sector address. ● History file size. ● Current entry sector address.	#HFPUT
Read current sector(s) from history file and place in history file I/O buffer.	Disk IOS
A Remove trailing blanks from input record. Place record in history file data buffer. Place additional information about record in history file data buffer: ● User ID. ● Job identifier. ● Terminal ID. ● Broadcast or display indicators. ● Entry length. ● Time of day. Move completed entry from history file data buffer to history file I/O buffer placing new entry immediately after previous current entry. [1] History file link edit version.	#HFPUT

Diagram 6.18 (Part 1 of 2). Perform History File Put Function

DESCRIPTION	MODULE/ ROUTINE
If new entry overflows into next sector, update current sector address in SCA. If more input for history file, return to 1 A and repeat process (maximum of 4 lines may be presented to history file at one call).	#HFPUT
Write current sector(s) from I/O buffer to history file on disk.	Disk IOS
Unlock and dequeue the history file. Place return code in history file parameter list.	#HFPUT

Diagram 6.18 (Part 2 of 2). Perform History File Put Function



From: Calling Program

INPUT

XR1

Supervisor Task Attach Parameter List

Main Storage

Transient Area: #SVAT

User Area

Attach parameter list

Main storage space

JCB, SCA, TCBs, and TUB

PROCESS

1 Assign new task control block

2 Determine main storage requirements

3 Assign and initialize areas needed to start new task

4 Place new task control block on swap-in-queue

5 Process attach error conditions if necessary

6 Terminate program

OUTPUT

New TCB placed on swap-in-queue

Request block (RB)

Action control element (ACE)

Swap area

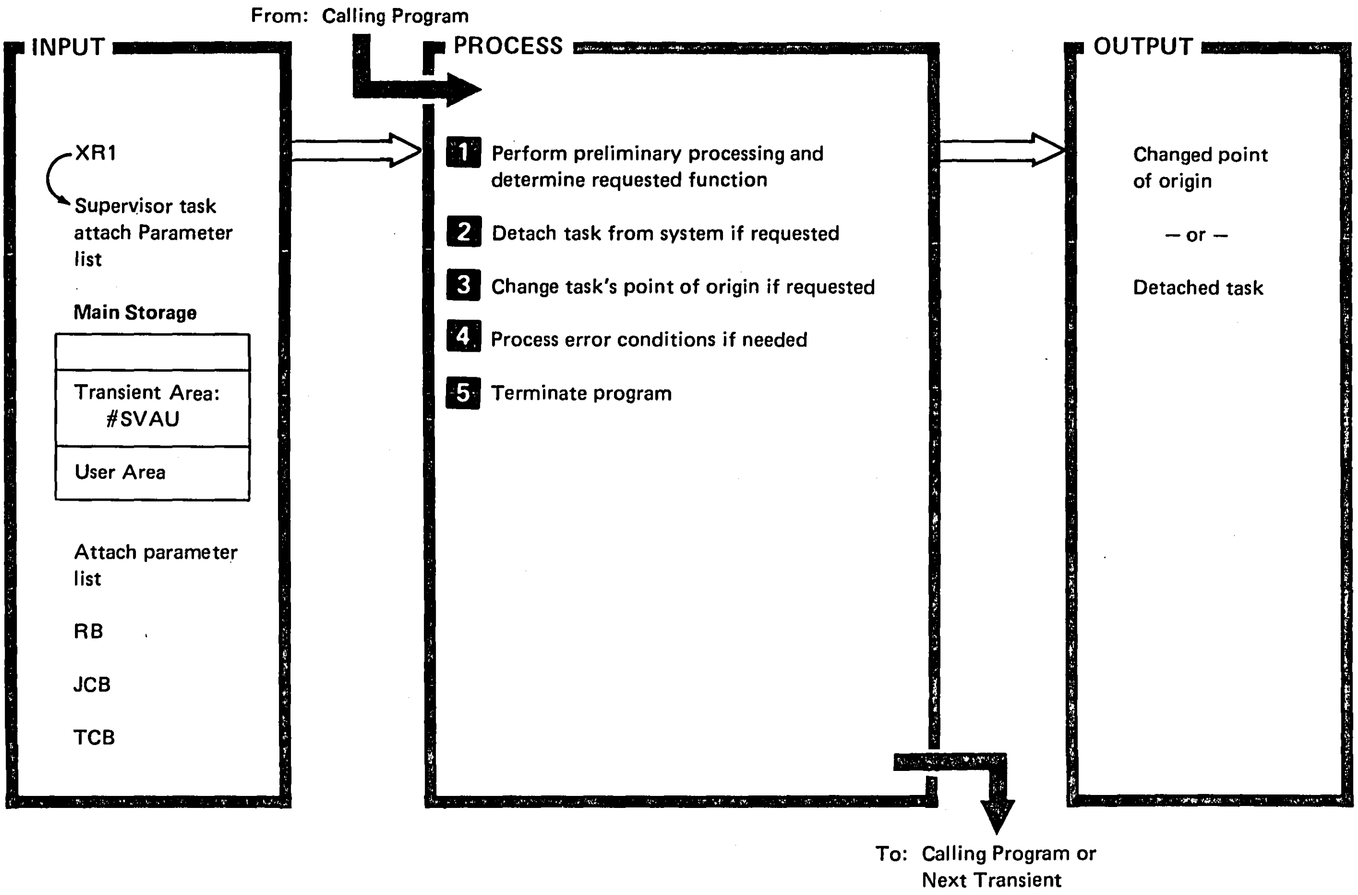
To: Calling Program, New Task, or Next Transient

DESCRIPTION	MODULE/ ROUTINE
1 Use control storage assign function to assign task control block (TCB) to run new task.	#SVAT
If unable to get TCB space, go to 5 with $ATERR04 return code.	
Initialize all TCB fields to zero.	
Calculate logical program begin number and place in TCB (TCBBEGL).	
Calculate logical start address. (This would be link edit address if module was link edited to 2K-byte boundary.)	
2 Calculate number of 2K-byte main storage blocks required to start task (TCBMSSIZ). (This is the largest program size to load program mainline or program size to execute.)	
Make all address translation register (ATR) values serial from logical zero to new tasks begin address.	
Ensure tasks main storage requirements do not exceed currently available main storage.	
Calculate maximum active swappable region in system (excluding attach or task).	
If insufficient swappable storage exists and task is swappable, go to 5 with $ATERR01 return code.	
If task is nonswappable and storage requirements may disable another task, go to 5 with $ATERR03 return code.	

Diagram 6.19 (Part 1 of 2). Perform Supervisor Task Attach Function (#SVAT)

DESCRIPTION	MODULE/ ROUTINE
3 If no terminal unit block (TUB) associated with attach request, but job control block (JCB) address (JCB@) passed in attach parameter list (used by spool writer and batch job queue attach requests): ● Allocate one-track work station work area (WSWA) on disk. ● If allocate not successful, go to 5 with $ATERR07 return code. ● Move allocated WSWA address into TCB. If TUB associated with attach request: ● Indicate TUB owned by new task. ● Use control storage assign function to assign action control element (ACE) to new task. ● If assign fails, go to 5 with $ATERR08 return code. ● Use control storage queue function to place completed ACE on tasks TCB complete queue. Use control storage assign function to assign request block (RB) in order to start new task. If assign fails, go to 5 with $ATERR05 return code. Allocate required disk swap area. If swap area not available, go to 5 with $ATERR06 return code. Assign task ID if not passed in attach parameter list.	#SVAT
If SYSIN data available, use control storage task work area (TWA) function to put SYSIN data in WSWA.	Disk IOS
If job name to be assigned, use control storage time-of-day function to place time in JCB. 4 Move system loader parameter list from attach parameter list to task — start bootstrap code. Move bootstrap code to assigned RB. Use control storage queue function to place new TCB on swapin queue. Use control storage stack manipulation function to unstack RB from chain. 5 Deallocate WSWA (if any). Use control storage free function to free ACE, RB, and TCB (if any). Put error code in XR1 (format is 00XX, where XX is error code). Return to calling program. 6 Store passed XR2 address. Get current TCB address. Update RB. If attach failed, return to calling program with error return code. If attach worked, and if next transient address ($ATSSSN) given, pass control to next transient. If attach worked, exit transient area.	#SVAT

Diagram 6.19 (Part 2 of 2). Perform Supervisor Task Attach Function (#SVAT)



From: Calling Program

INPUT

XR1

Supervisor task attach Parameter list

Main Storage

Transient Area: #SVAU

User Area

Attach parameter list

RB

JCB

TCB

PROCESS

1 Perform preliminary processing and determine requested function

2 Detach task from system if requested

3 Change task's point of origin if requested

4 Process error conditions if needed

5 Terminate program

OUTPUT

Changed point of origin

– or –

Detached task

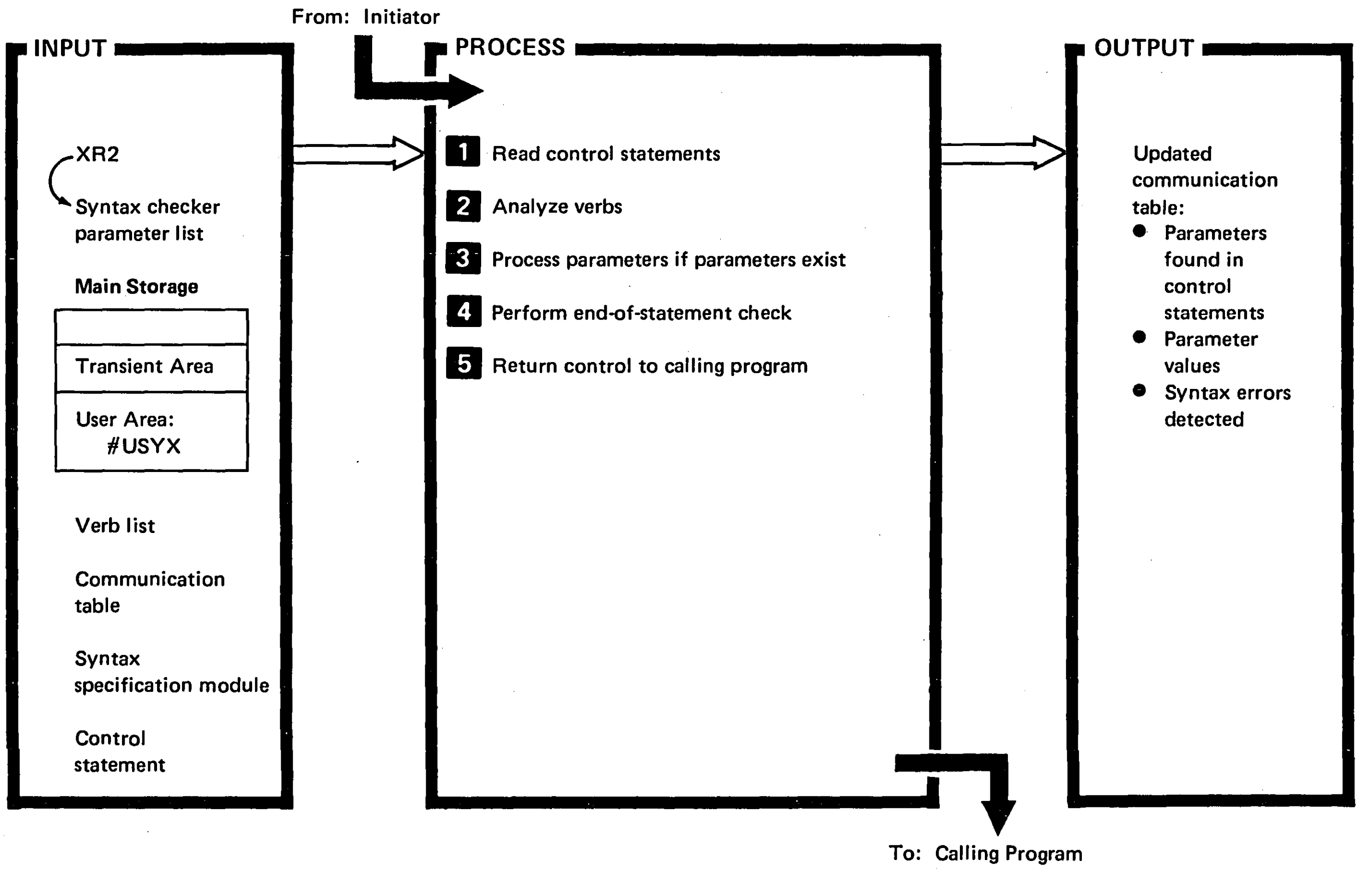
To: Calling Program or Next Transient

DESCRIPTION	MODULE/ ROUTINE
1 Calculate maximum swappable active region size not including this task and store value in TCB@INL1. Deallocate all push elements for current TCB: ● Use control storage free function to free push element. ● Deallocate track on disk. Calculate old swap size in tracks and place value in TCB@INL2. If request to detach task from system, go to **2**. If request to change task's point of origin, go to **3**.	#SVAU
2 Use control storage free function to free all assign free queued areas for task being detached. Use control storage free page function to free user main storage area. Deallocate task's swap area on disk. If no terminal unit block (TUB) associated with task and job control block (JCB) address not zero: ● Use control storage free function to free JCB. ● If not request to detach batch job, deallocate work station work area (WSWA). Post command processor to restart batch job queue.	

Diagram 6.20 (Part 1 of 2). Perform Supervisor Task Detach or Change Org Point Function (#SVAU)

DESCRIPTION	MODULE/ ROUTINE
Use control storage dequeue function to dequeue this task control block (TCB) from all system queues.	#SVAU
Post command processor for stop system request.	
Use control storage free function to free all request blocks (RBs) associated with task and free this tasks TCBs.	
Go to 5 to terminate program.	
3 Calculate program size in number of blocks required to load program and place value in TCB@INL3.	
Calculate required disk swap area size and place track size in TCB@INL4.	
If not enough main storage space available to load task, go to 4 with $ATERR01 return code.	
If task not swappable: ● If task initiation will disable active task, go to 4 with $ATERR03 return code. ● If task will not leave at least 14K bytes main storage available, go to 4 with $ATERR02 return code.	
If larger disk swap area needed or task initiation and swap area larger than needed: ● Deallocate old swap area on disk (Size in TCB@INL2). ● Allocate new swap area on disk (size in TCB@INL4). ● If allocate not successful, allocate old swap area back and go to 4 with $ATERR06 error code.	
Use control storage free page function to free all main storage except first 2K bytes associated with task.	
Move bootstrap code to first 2K-byte area.	
4 Store callers address recall register (ARR).	
Put error code in XR1 (format is 00XX, whereXX is error code).	
Return to calling program.	
5 Store new instruction address register (IAR) value.	
Store new XR1 value (RB address in XR1).	
If request to change tasks point of origin, pass control to program specified in attach parameter list.	
If request to detach task, pass control to the dispatcher (control storage).	

Diagram 6.20 (Part 2 of 2). Perform Supervisor Task Detach or Change Org Point Function (#SVAU)



From: Initiator

INPUT

XR2

Syntax checker parameter list

Main Storage

Transient Area
User Area: #USYX

Verb list

Communication table

Syntax specification module

Control statement

PROCESS

1. Read control statements
2. Analyze verbs
3. Process parameters if parameters exist
4. Perform end-of-statement check
5. Return control to calling program

OUTPUT

Updated communication table:

- Parameters found in control statements
- Parameter values
- Syntax errors detected

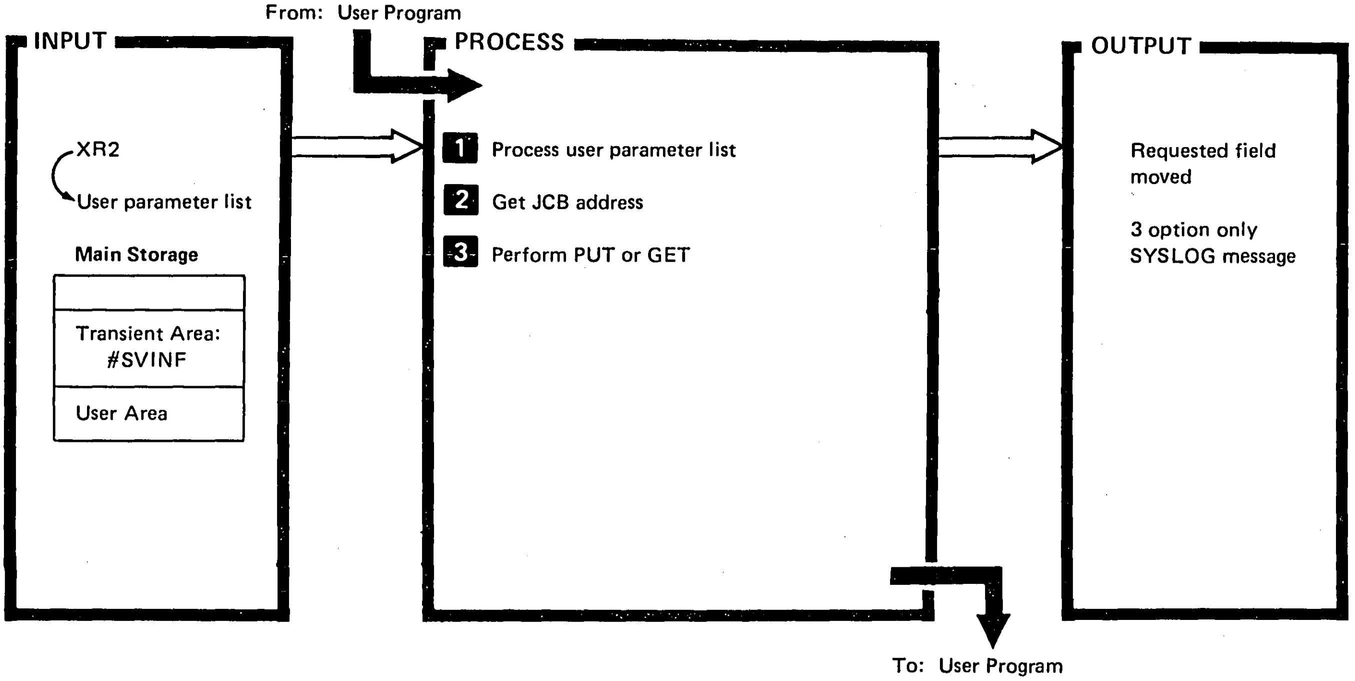
To: Calling Program

DESCRIPTION	MODULE/ ROUTINE
1 Retrieve heading inserts.	#MGRET
Load calling program's syntax specification module:	#USYX
● Find module.	#MASFN
● Use main storage relocating loader to load module into main storage.	#USYX
Put control statement into internal buffer:	
● If user providing input, copy it.	
● Read control statement if not user provided.	#CLSN
2 Ensure input starts with //	#USYX
Skip over blanks.	
Collect characters until blank encountered (verb is characters collected).	
Ensure verb corresponds to name field of verb entry in specification module.	
If verb in list of verbs currently acceptable, copy verb number from verb entry in specification module and place in communication table at USCTVRBN.	
Initialize internal pointers.	

Diagram 6.21 (Part 1 of 2). Perform Syntax Checking Function (#USYX)

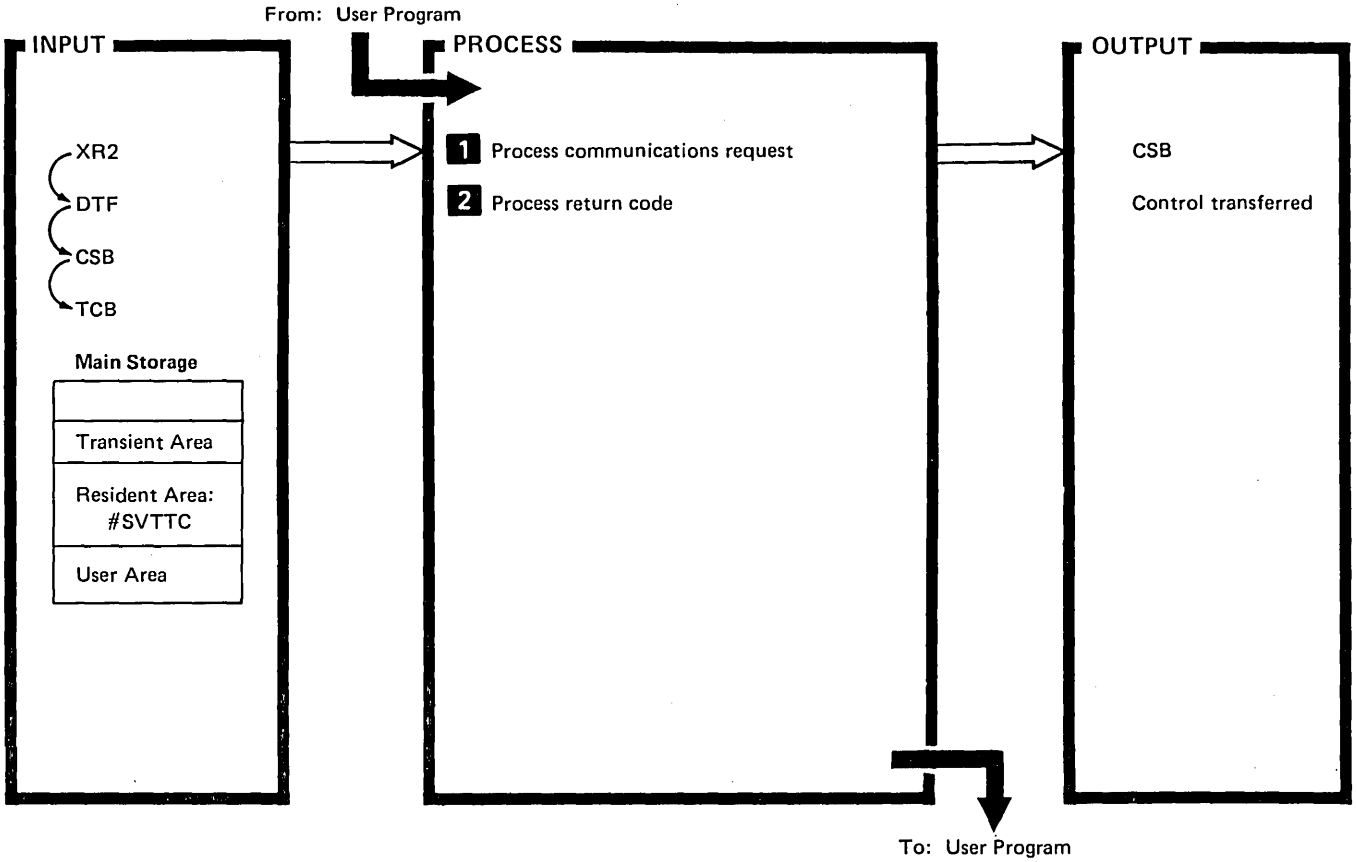
DESCRIPTION	MODULE/ ROUTINE
3 Check if parameters specified. Set scan terminator as comma, hyphen, or blank. Collect characters preceding terminator to form parameter. If scan terminated by hyphen: ● Indicate parameter is keyword. ● Ensure keyword entry has corresponding field name in specification module. ● Save keyword entry attributes. If scan terminated by comma or blank: ● Indicate parameter is positional. ● Ensure positional parameter entry exists in specification module. ● Save positional parameter entry attributes. Ensure parameter not already encountered. A Verify parameter values: ● If parameter attribute is numeric: – Ensure value is numeric. – Retain character format. – Convert value to 3-byte signed binary field. ● If parameter attribute is a date: – Ensure format is acceptable. – Ensure month, day, and year characters are numeric. – Convert date to packed YYMMDD format. ● If parameter attribute is a label, ensure first character of value is alphabetic. ● If parameter attribute is a quoted string: – Ensure all quotes are paired. – Ensure first and last characters of value are quotes. ● Else parameter value is a string. If required, place verified parameter value in caller's output area. If required, a specific parameter value causes placement of a one, two, or 3-byte value in caller's output area. Continue processing until all parameters analyzed. 4 Point to parameter entries one at a time. If optional parameter with default and value not in input, go to 3 A to verify default value, then return here. If required parameter not in input, set appropriate error condition in communication table. Verify parameter combinations if VALCM records exist: ● If parameter appeared in input, ensure parameter must not be missing. ● If specified parameter must have specific value, ensure value is correct. ● Set appropriate error condition in communication table as necessary. 5 Update communication table. Set return code as required. Copy updated communication table into caller's area. Pass control back to calling program.	#USYX

Diagram 6.21 (Part 2 of 2). Perform Syntax Checking Function (#USYX)



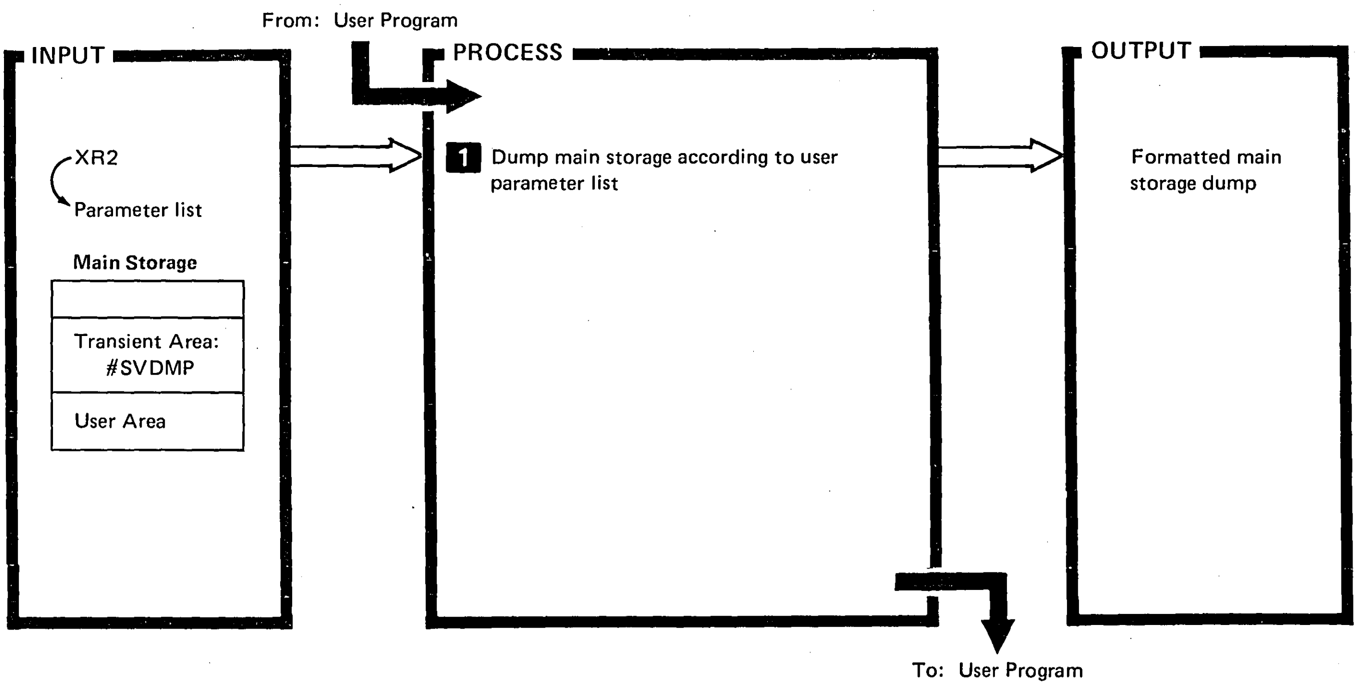
DESCRIPTION	MODULE/ ROUTINE
1 Move parameter list to transient area. ·Verify fields in parameter list: ● If local request, check for valid entries in offset and length fields.	#SVINF
● If invalid entry found, issue 3 option only message.	#CLSG
2 If parameter list contains a terminal ID, get the JCB address from that work station's TUB. Otherwise, get JCB address from the TCB. **3** If put request: ● If update UPSI switch request, move value to JCBDUPSI. ● If update language compiler byte request, move value to JCBDLANG. ● If update program message member one address request, move value to JCBDPRG1. ● If update program message member two address request, move value to JCBDPRG2. ● If update local area on disk request, move value to WSWA on disk. ● If update user message member one address request, move value to JCBDUSR1. If get request: ● If return date format request, move value from JCBDSCH1. ● If return program date request, move value from JCBDPDAT. ● If return session date request, move value from JCBDDATE. ● If return UPSI switch value request, move value from JCBDUPSI. ● If return inquiry byte value request, move value from JCBDSCH2.	#SVINF

Diagram 6.22. Perform Information Retrieval Function



DESCRIPTION	MODULE/ ROUTINE
1 If the data communications task has been terminated, return to caller. Otherwise, post the data communications task for execution. Increment the calling task's nonswap count to ensure that it is not swapped until the data communications task has retrieved internal data.	#SVTTC
2 Wait for a return post from data communications: ● If return code from data communications task is hex 01, go to **1**. ● If return code from data communications task is hex 02: – Call the transient addressed in the CSB. – Go to **1**. ● If return code from data communications task is hex 00, return to caller.	

Diagram 6.23. Perform Data Management Task Transfer Control Function


From: User Program

INPUT

XR2

Parameter list

Main Storage

Transient Area: #SVDMP

User Area

PROCESS

1 Dump main storage according to user parameter list

OUTPUT

Formatted main storage dump

To: User Program

DESCRIPTION	MODULE/ ROUTINE
1 Move parameter list to temporary area.	#SVDMP
Initialize work area.	
Calculate dump limits.	
If limits invalid:	
● Issue error message. ● Return to caller.	#CLSP
Get headings messages for dump header.	#SVDMP
Blank buffer.	
Set up buffer for SYSLIST.	
Move all limits information to header.	
A Set up line of dump.	
Print line of dump.	#CLSP
If more lines to dump, go to 1 A.	#SVDMP
Free buffer area.	
Return to user program.	

Diagram 6.24. Perform Snap Dump Function

Program Organization

Figures 6-2 through 6-26 show the control flow of the system service functions.

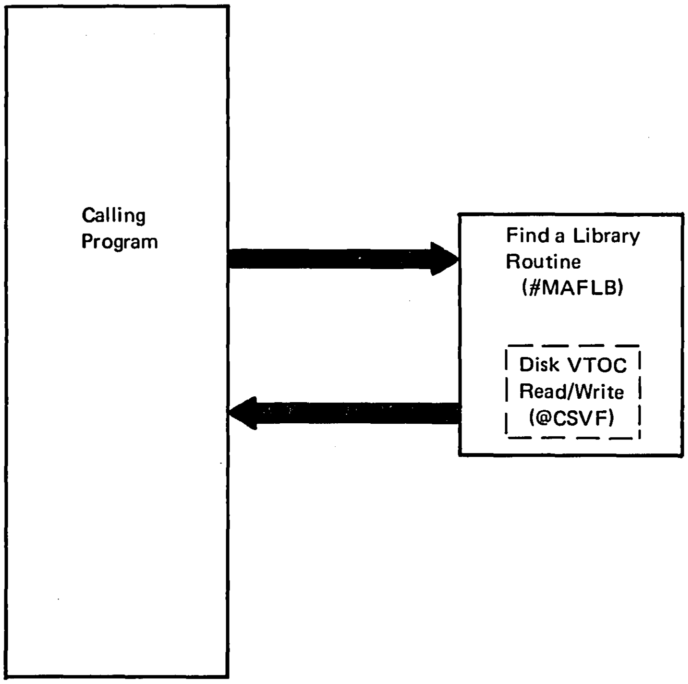


Figure 6-2. Find a Library Control Flow

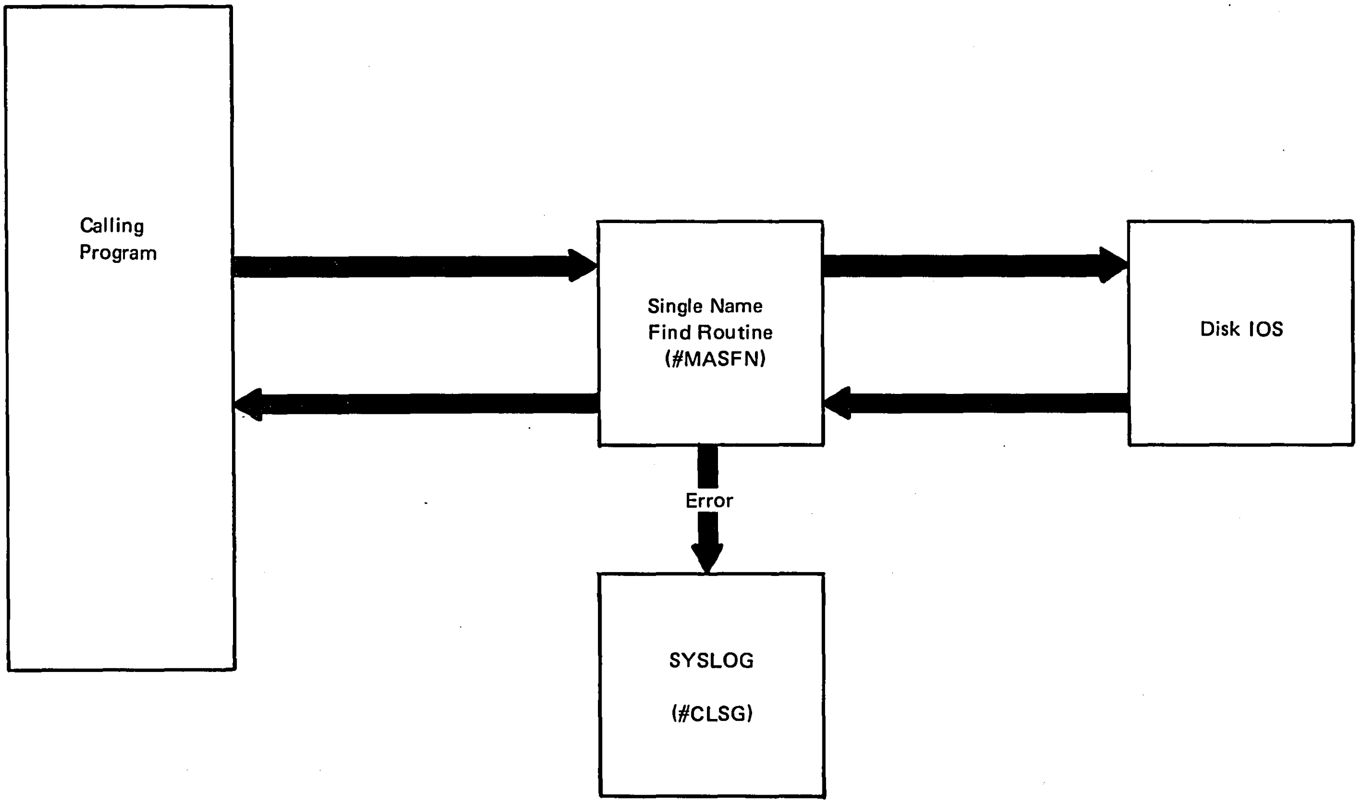


Figure 6-3. Single Name Find Routine Control Flow

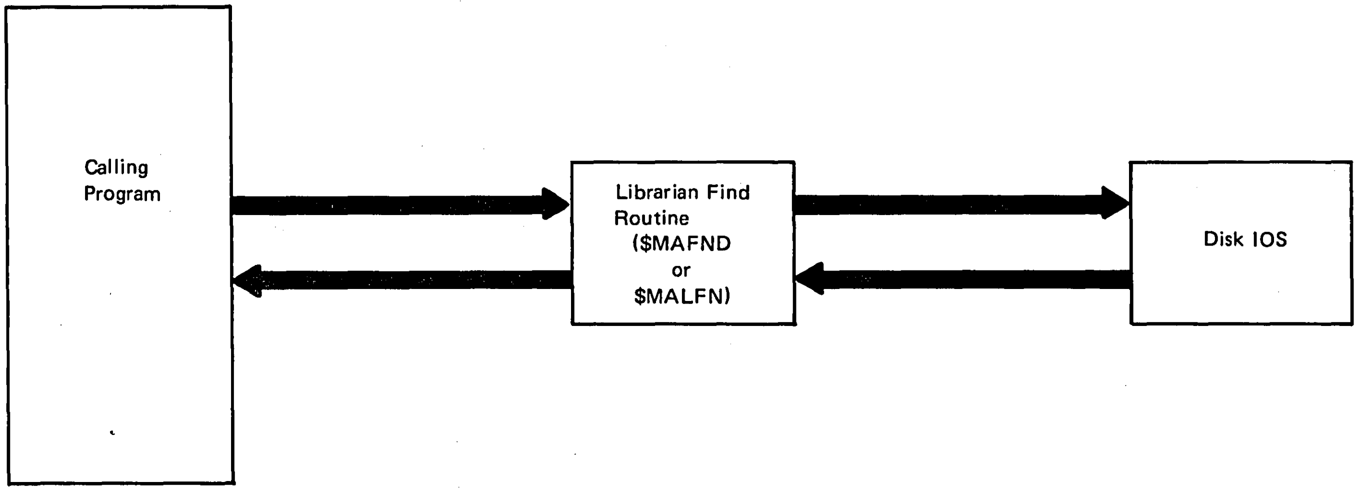


Figure 6-4. Librarian Find Control Flow

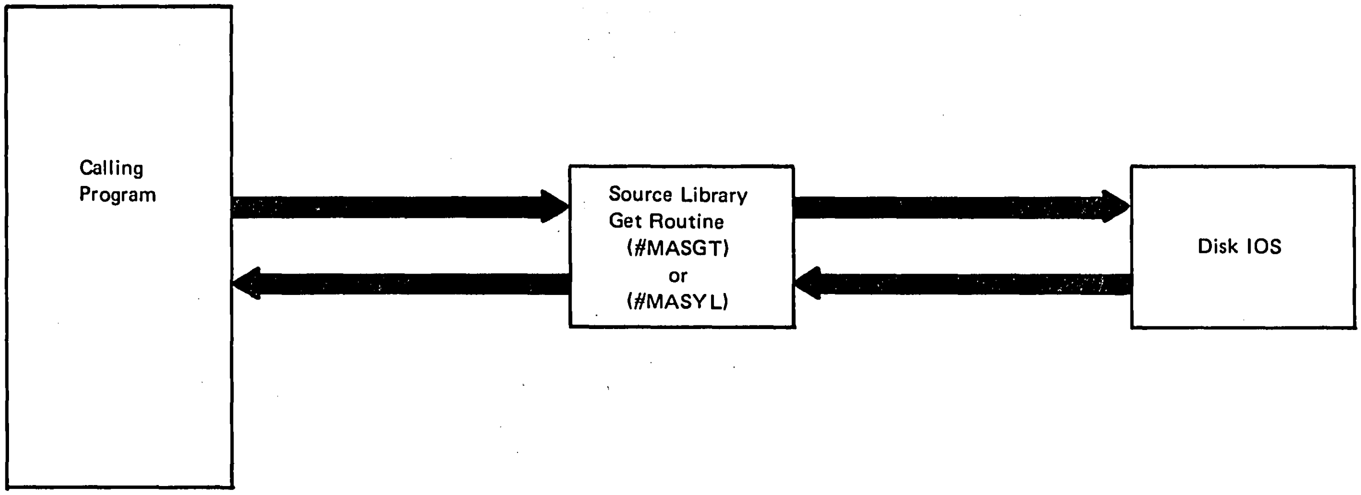


Figure 6-5. Source Library Get Routine Control Flow

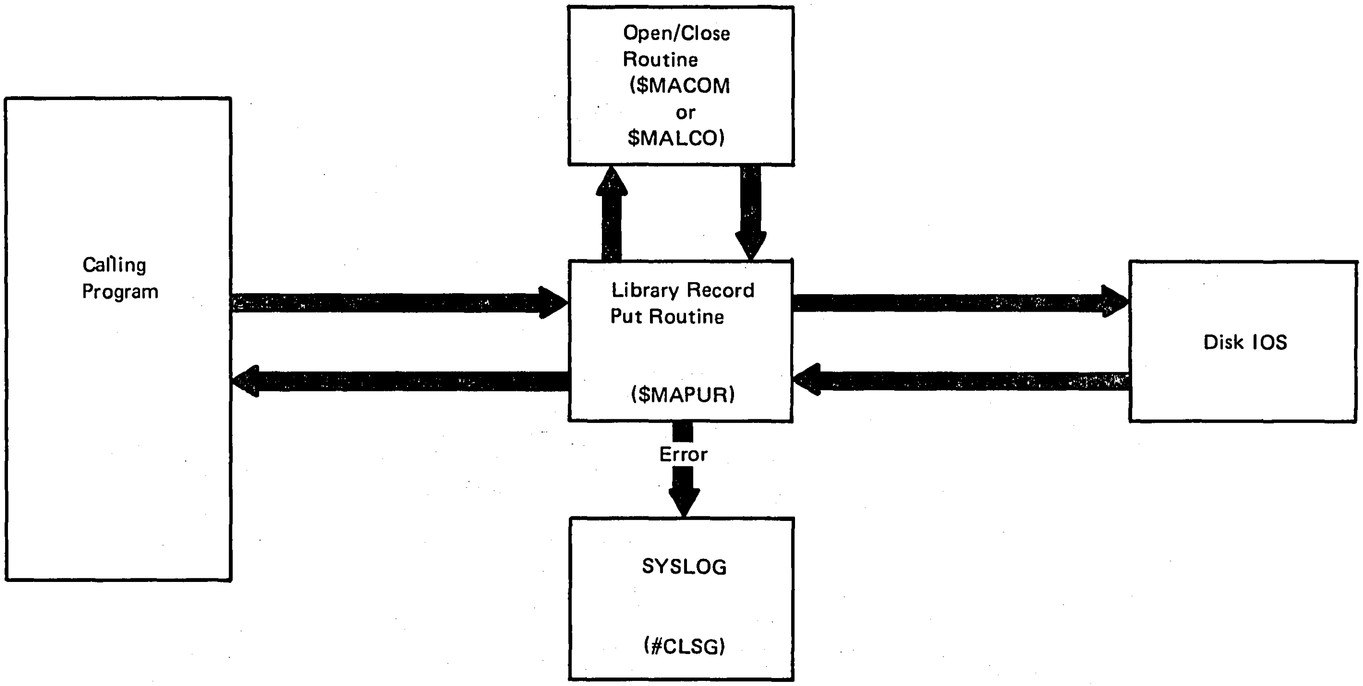

Open/Close Routine ($MACOM or $MALCO)
Calling Program
Library Record Put Routine ($MAPUR)
Disk IOS
Error
SYSLOG (#CLSG)

Figure 6-6. Library Record Put Routine Control Flow

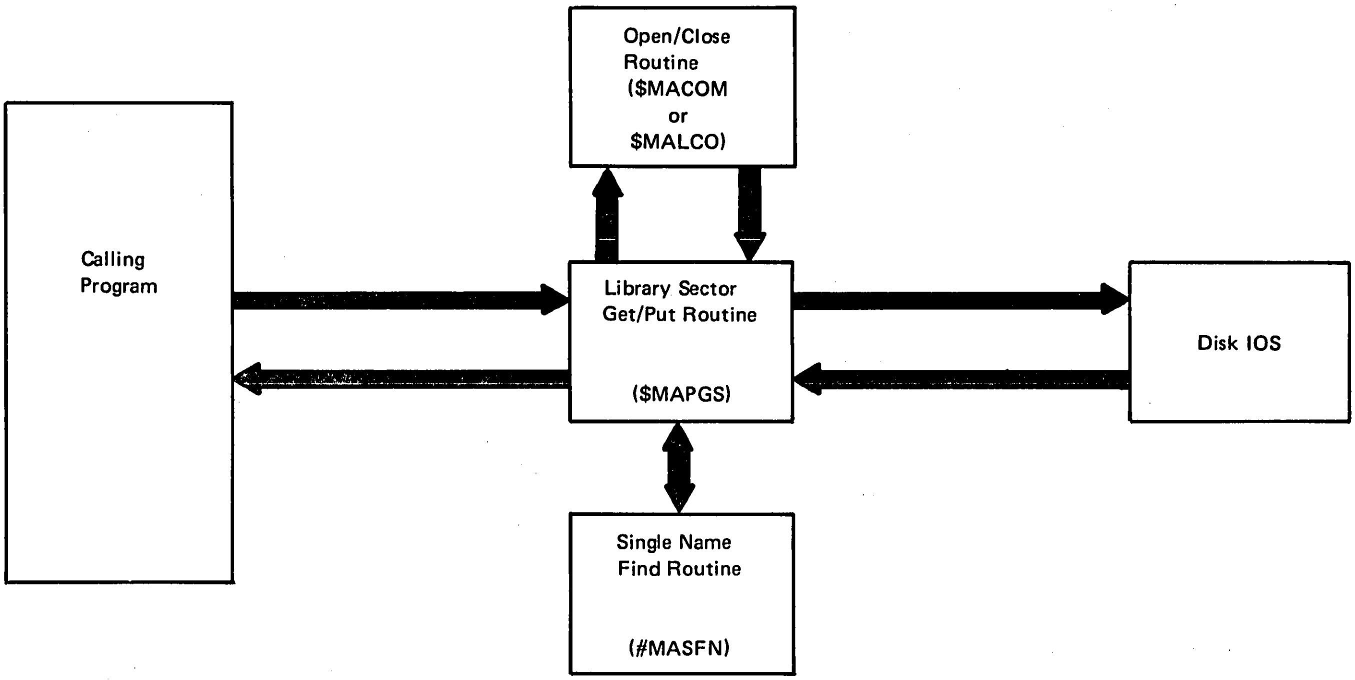

Open/Close Routine ($MACOM or $MALCO)
Calling Program
Library Sector Get/Put Routine ($MAPGS)
Disk IOS
Single Name Find Routine (#MASFN)

Figure 6-7. Library Sector Get/Put Routine Control Flow

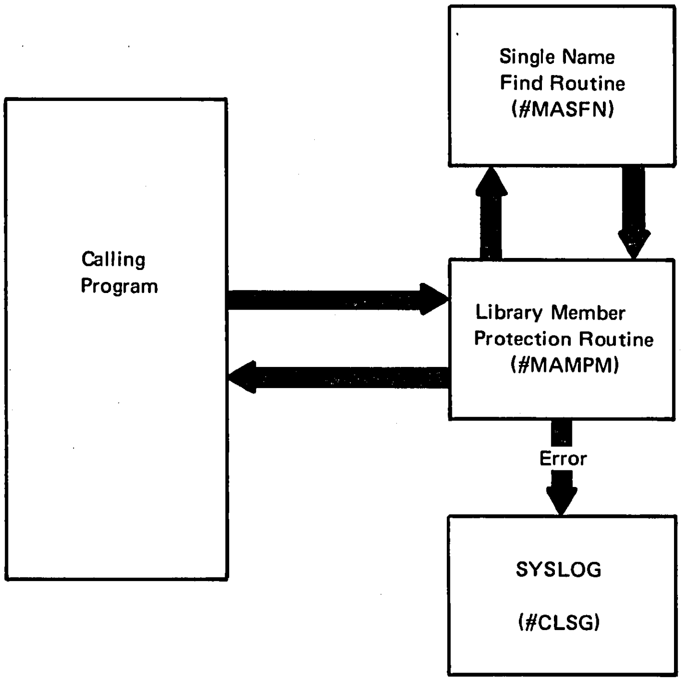

Single Name Find Routine (#MASFN)
Calling Program
Library Member Protection Routine (#MAMPM)
Error
SYSLOG (#CLSG)

Figure 6-8. Library Member Protection Control Flow

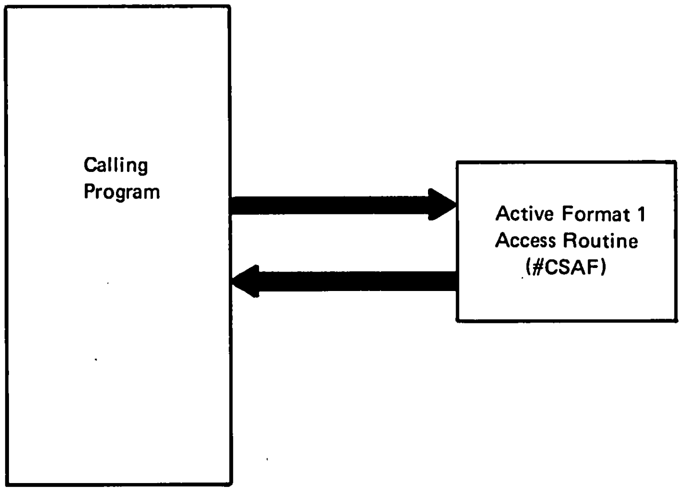


Figure 6-9. AFA Access Routine Control Flow

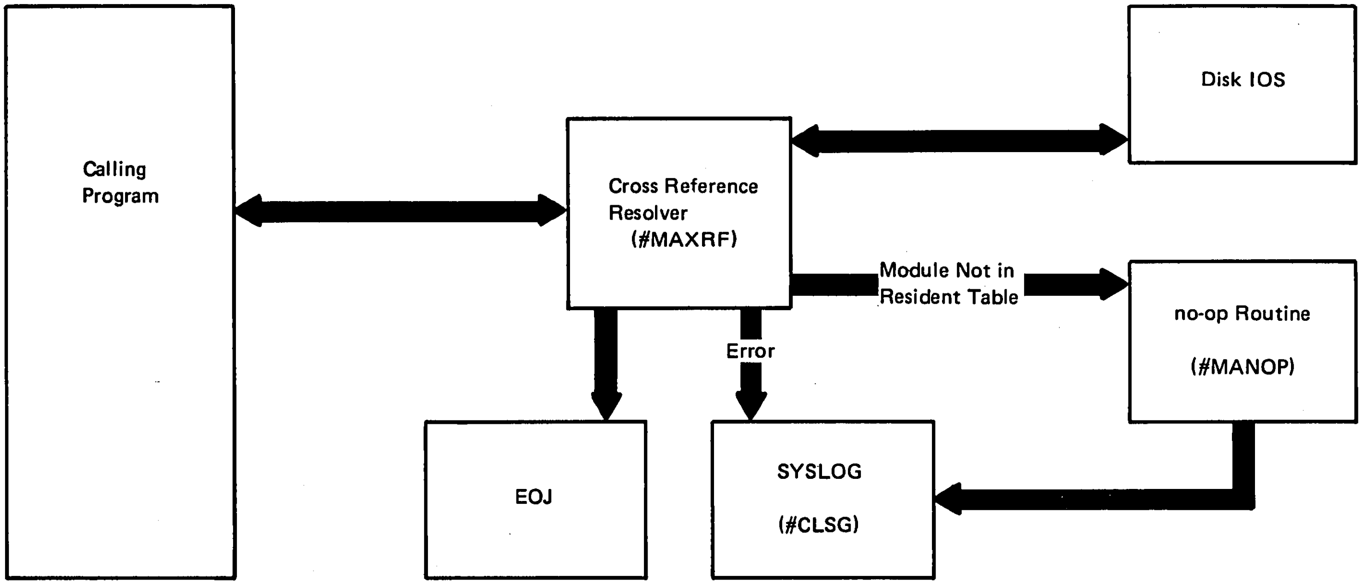


Figure 6-10. Cross Reference Resolver Control Flow

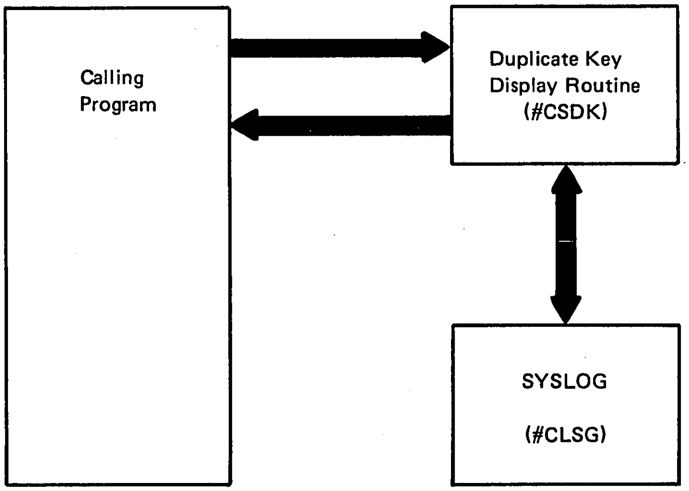


Figure 6-11. Duplicate Key Display Control Flow

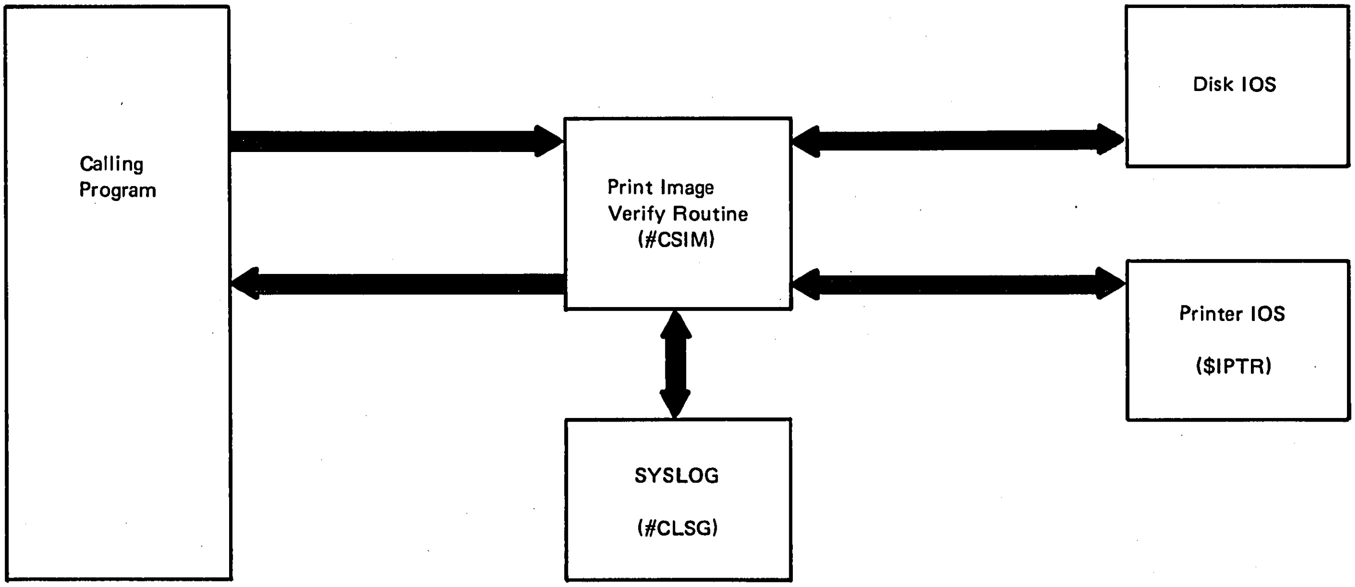


Figure 6-12. Print Image Verify Routine Control Flow

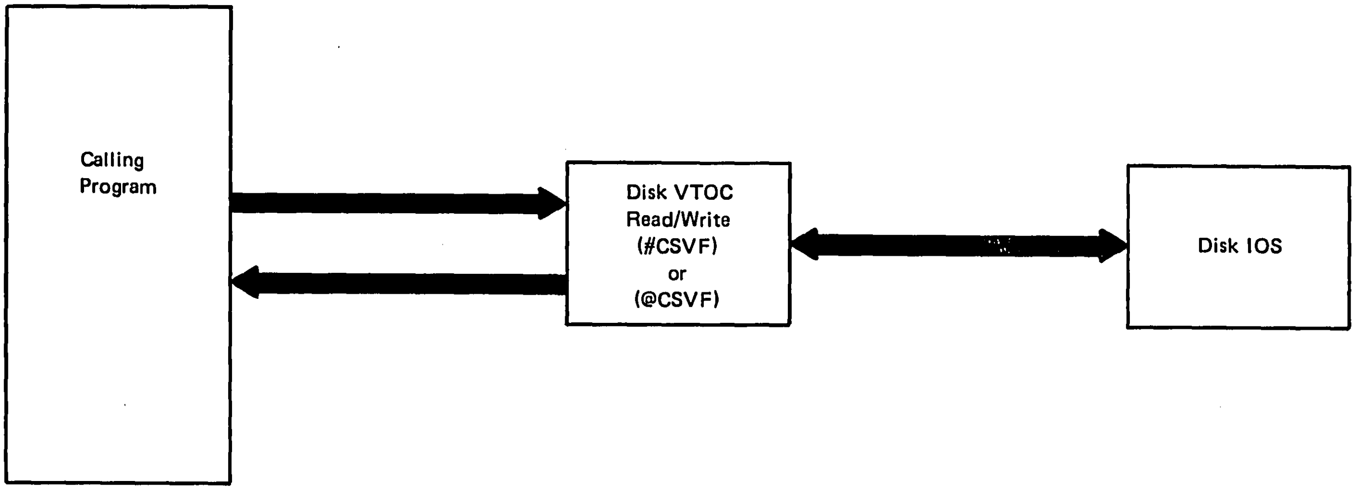


Figure 6-13. Disk VTOC Read/Write Control Flow

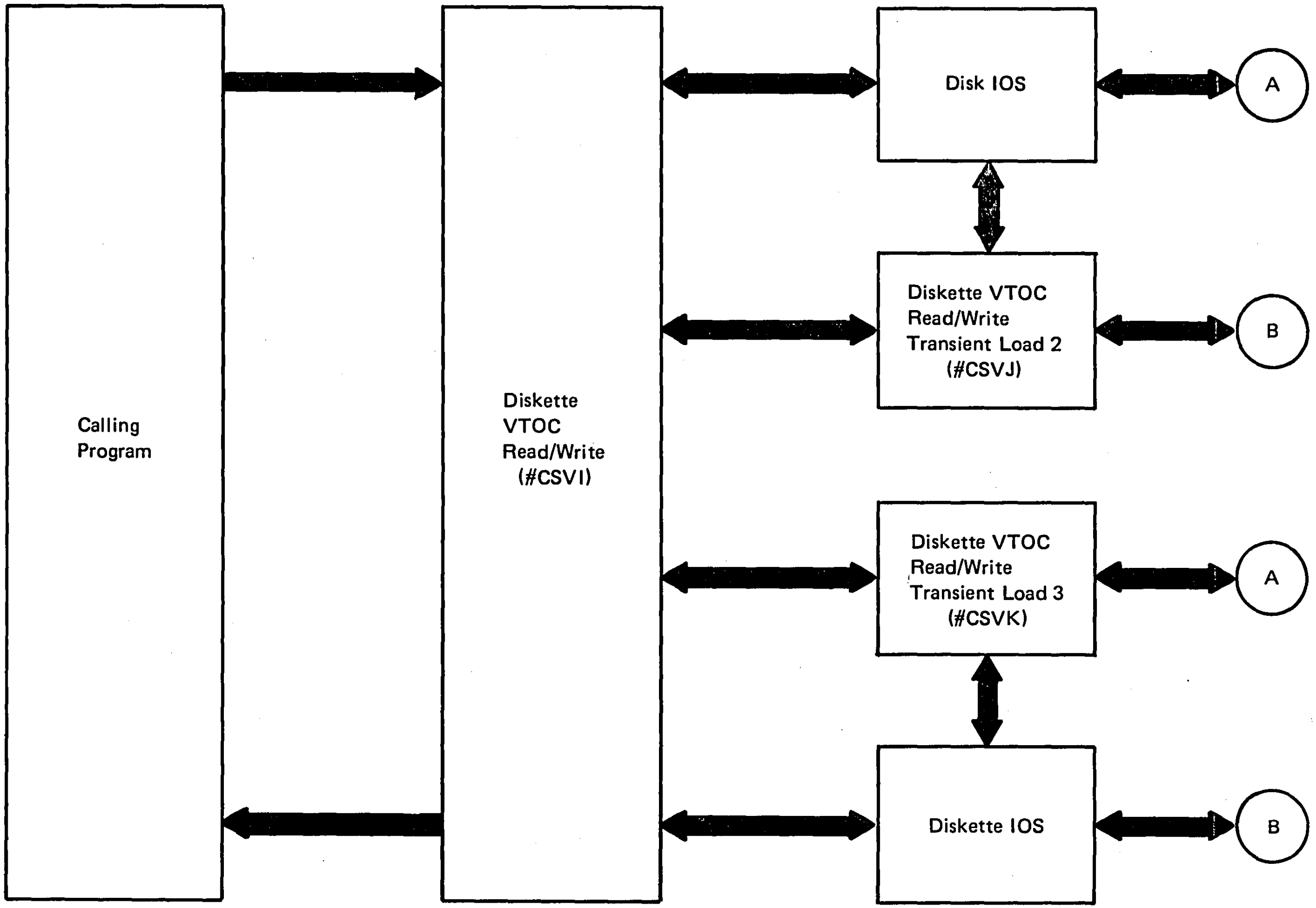

Calling Program
Diskette VTOC Read/Write (#CSVI)
Disk IOS
A
Diskette VTOC Read/Write Transient Load 2 (#CSVJ)
B
Diskette VTOC Read/Write Transient Load 3 (#CSVK)
A
Diskette IOS
B

Figure 6-14. Diskette VTOC Read/Write Control Flow

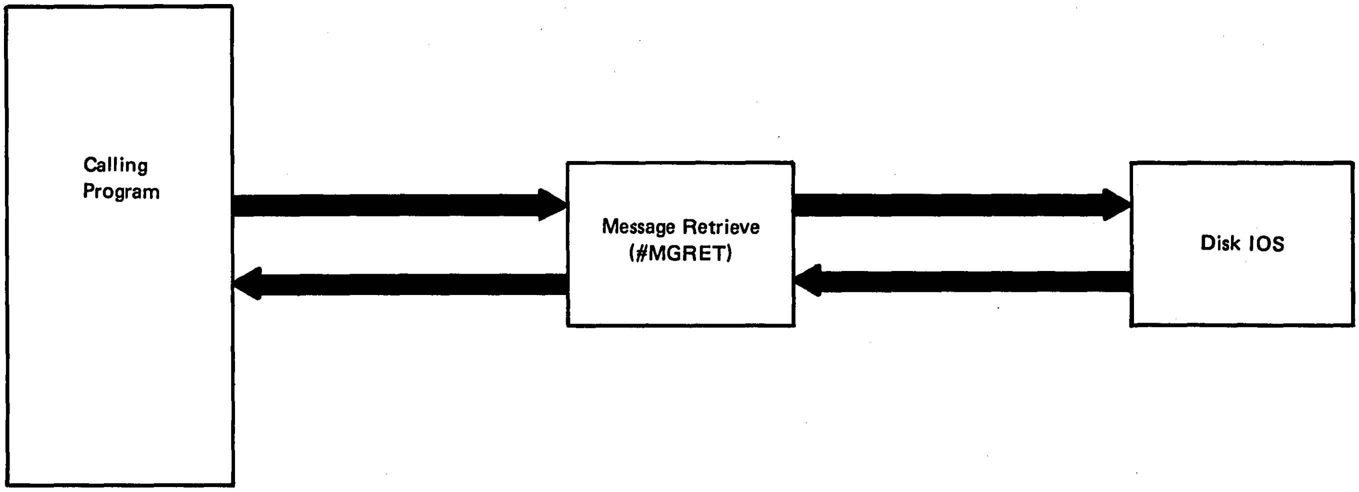


Figure 6-15. Message Retrieve Control Flow

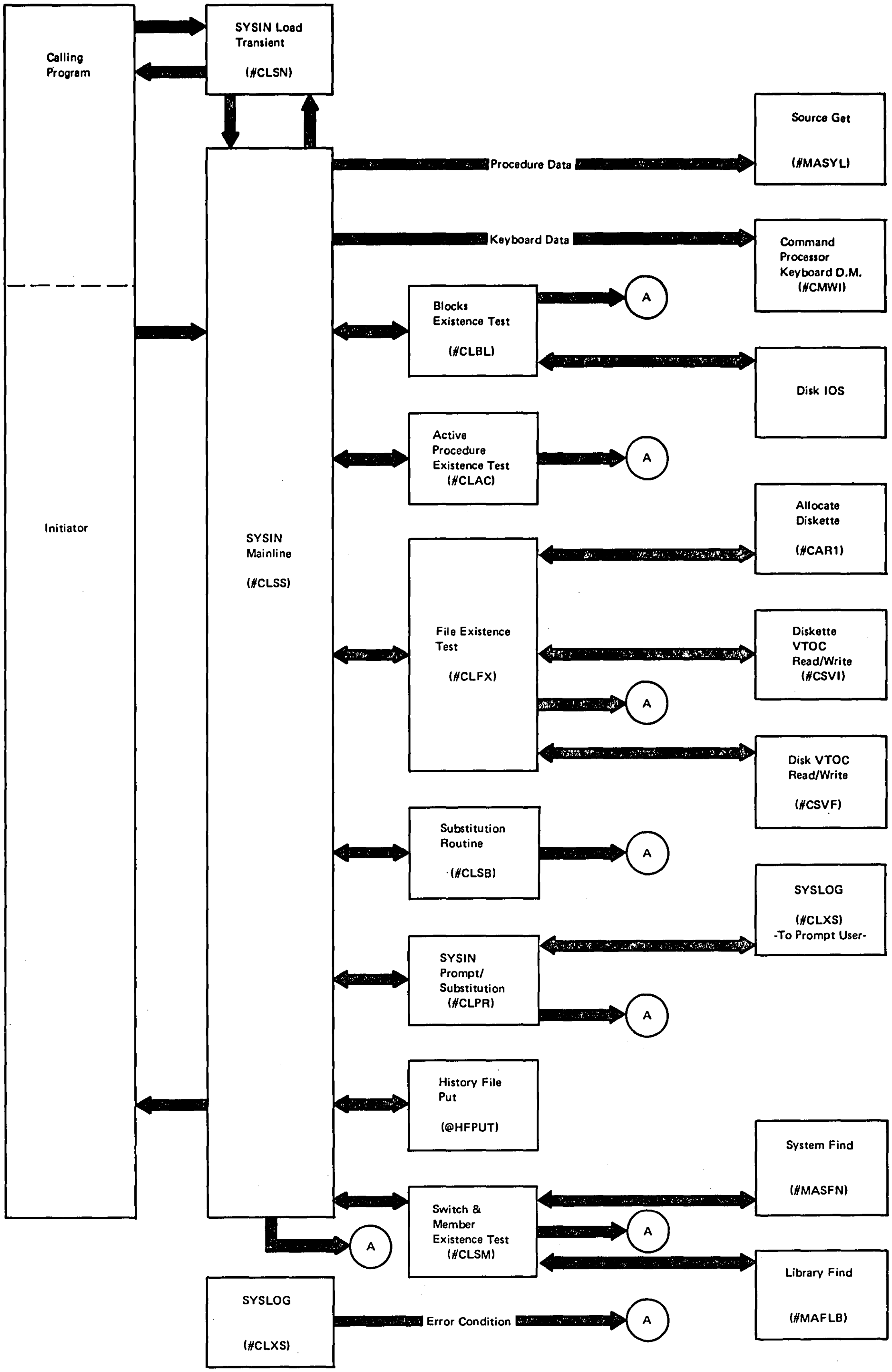
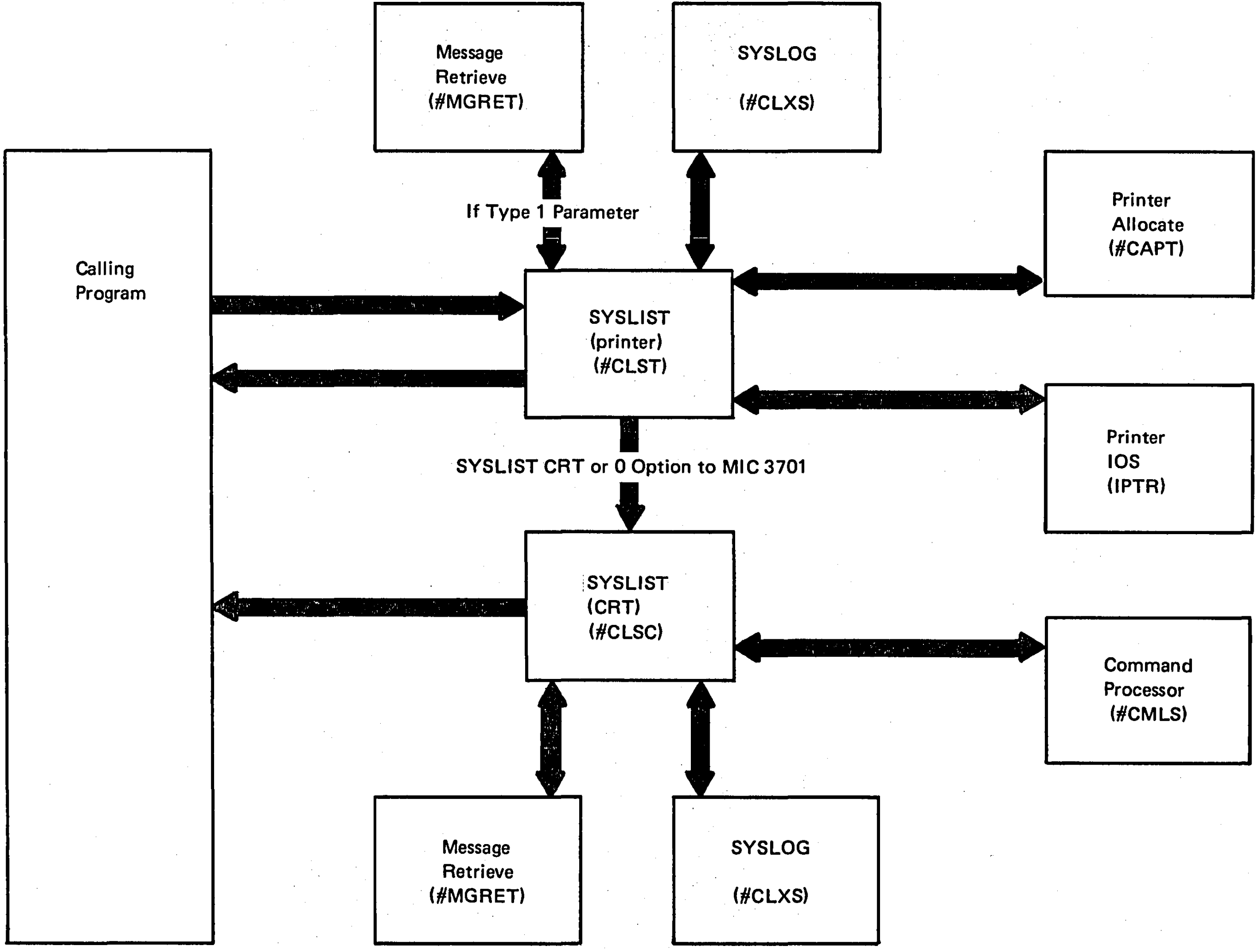
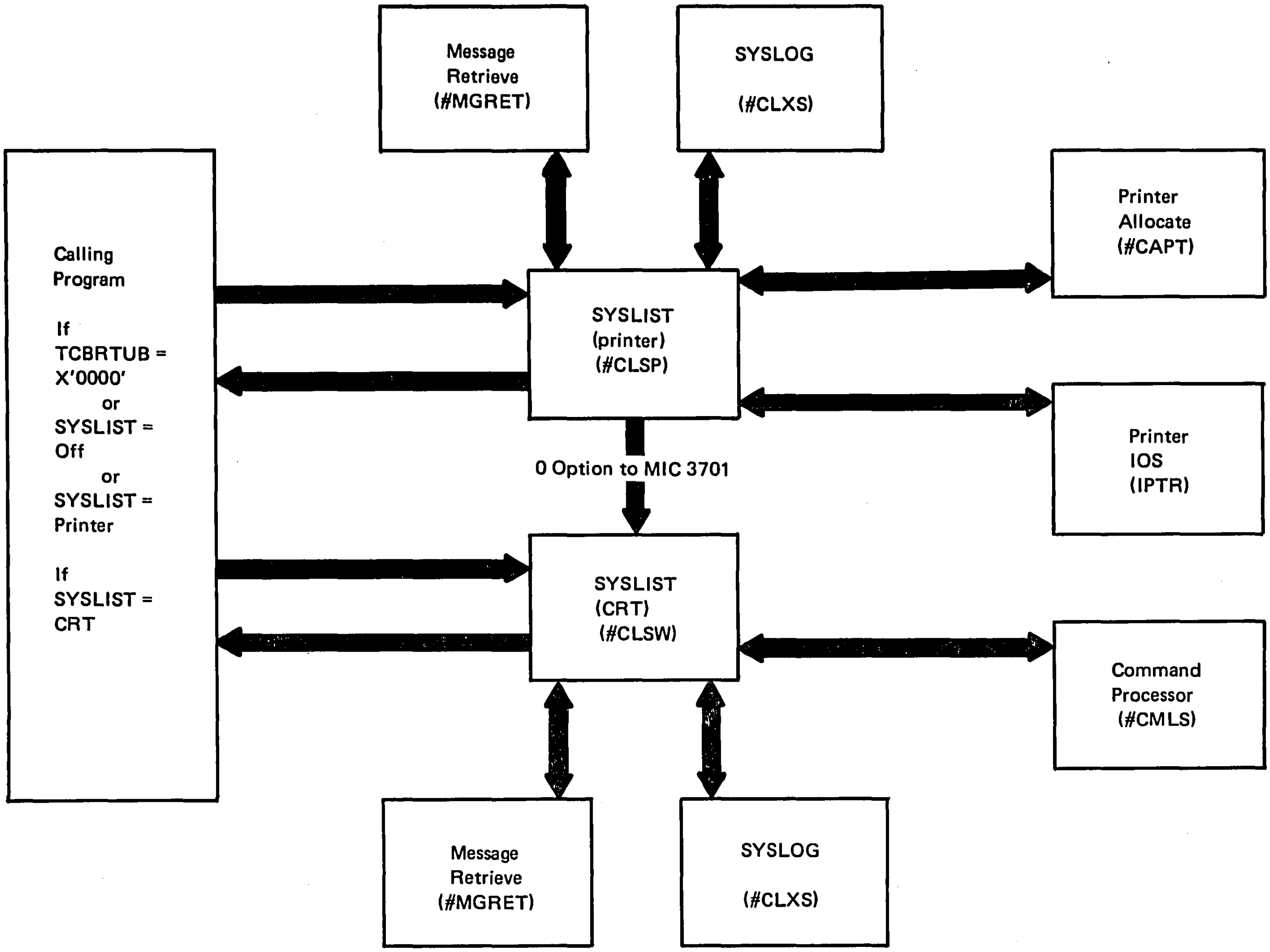

Calling Program
SYSIN Load Transient (#CLSN)
Source Get (#MASYL)
Procedure Data
Keyboard Data
Command Processor Keyboard D.M. (#CMWI)
Blocks Existence Test (#CLBL)
A
Disk IOS
Active Procedure Existence Test (#CLAC)
A
Initiator
SYSIN Mainline (#CLSS)
Allocate Diskette (#CAR1)
File Existence Test (#CLFX)
Diskette VTOC Read/Write (#CSVI)
A
Disk VTOC Read/Write (#CSVF)
Substitution Routine (#CLSB)
A
SYSLOG (#CLXS) -To Prompt User-
SYSIN Prompt/ Substitution (#CLPR)
A
History File Put (@HFPUT)
System Find (#MASFN)
Switch & Member Existence Test (#CLSM)
A
A
Library Find (#MAFLB)
SYSLOG (#CLXS)
Error Condition
A

Figure 6-16. SYSIN Control Flow


Message Retrieve (#MGRET)
SYSLOG (#CLXS)
Printer Allocate (#CAPT)
Calling Program
If Type 1 Parameter
SYSLIST (printer) (#CLST)
Printer IOS (IPTR)
SYSLIST CRT or 0 Option to MIC 3701
SYSLIST (CRT) (#CLSC)
Command Processor (#CMLS)
Message Retrieve (#MGRET)
SYSLOG (#CLXS)

Note: Whenever SYSLIST (transient) is called, #CLST is loaded into the transient area first.

Figure 6-17. Transient SYSLIST Control Flow



Note: The SLSTL macro loads either #CLSP or #CLSW into the user specified area.

Figure 6-18. Loadable SYSLIST Control Flow

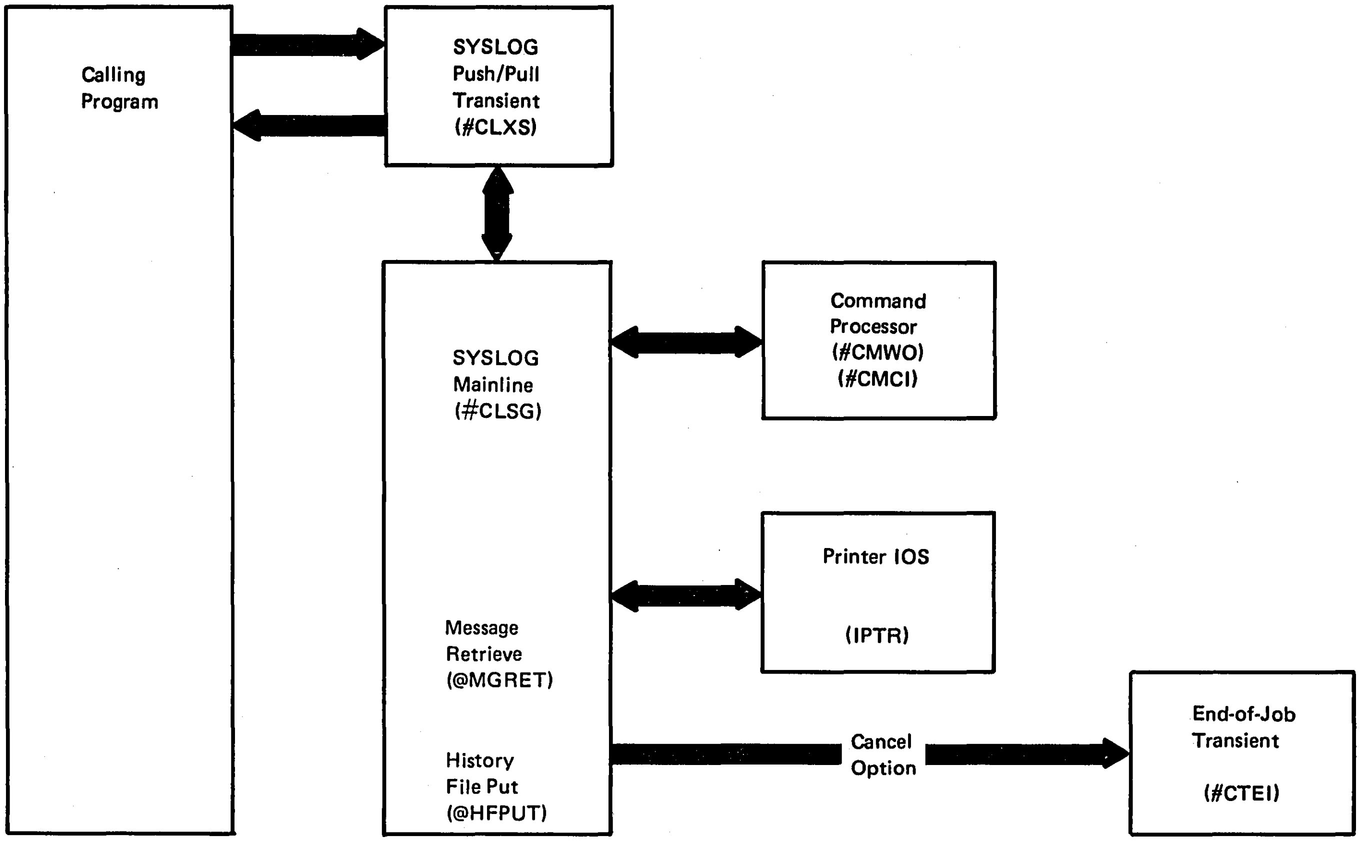

Calling Program
SYSLOG Push/Pull Transient (#CLXS)
SYSLOG Mainline (#CLSG)
Message Retrieve (@MGRET)
History File Put (@HFPUT)
Command Processor (#CMWO) (#CMCI)
Printer IOS (IPTR)
Cancel Option
End-of-Job Transient (#CTEI)

Figure 6-19. SYSLOG Control Flow

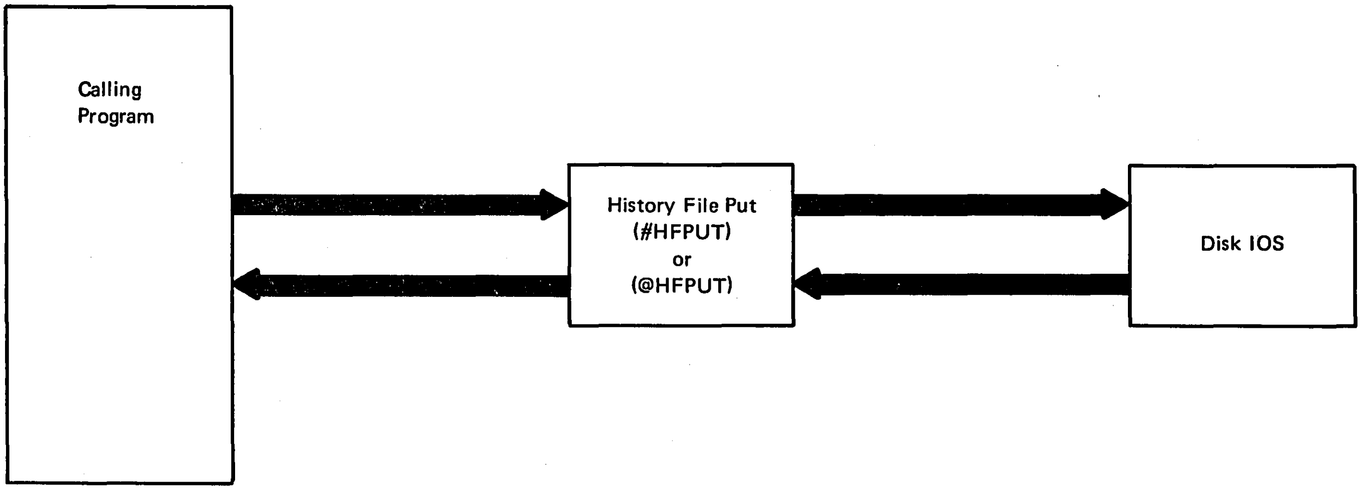


Figure 6-20. History File Put Control Flow

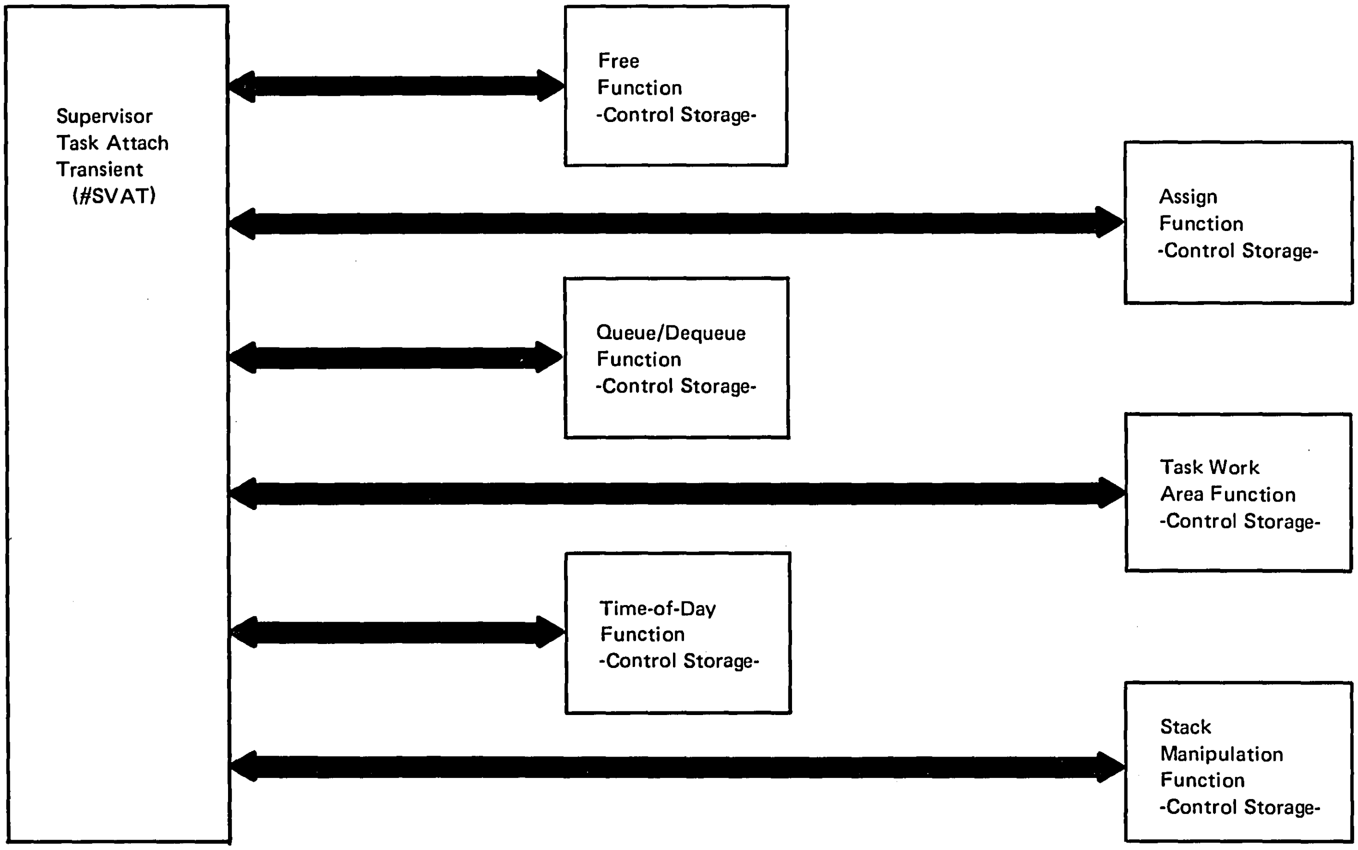


Figure 6-21. Supervisor Task Attach Function Control Flow (#SVAT)

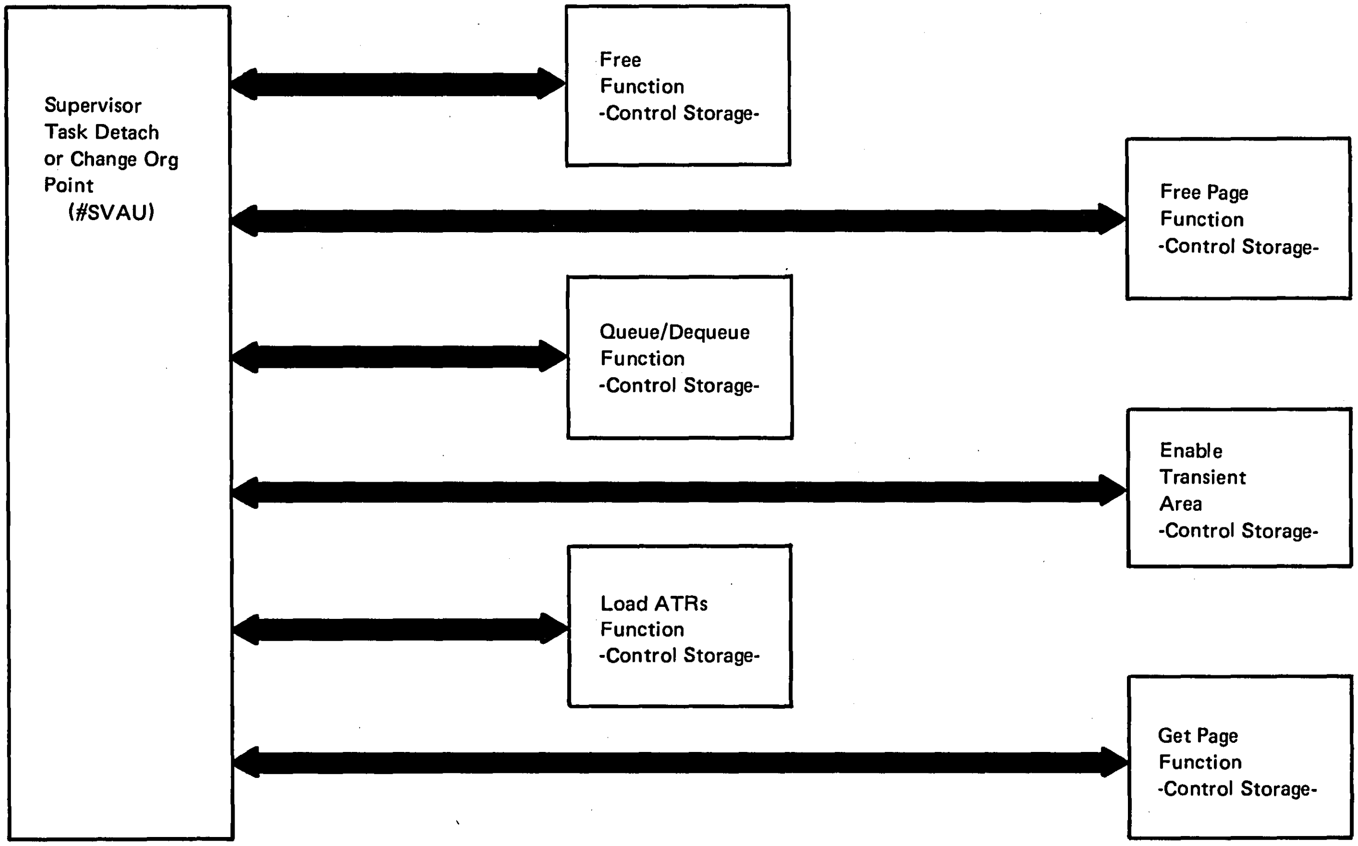


Figure 6-22. Supervisor Task Detach or Change Origin Point Function Control Flow (#SVAU)

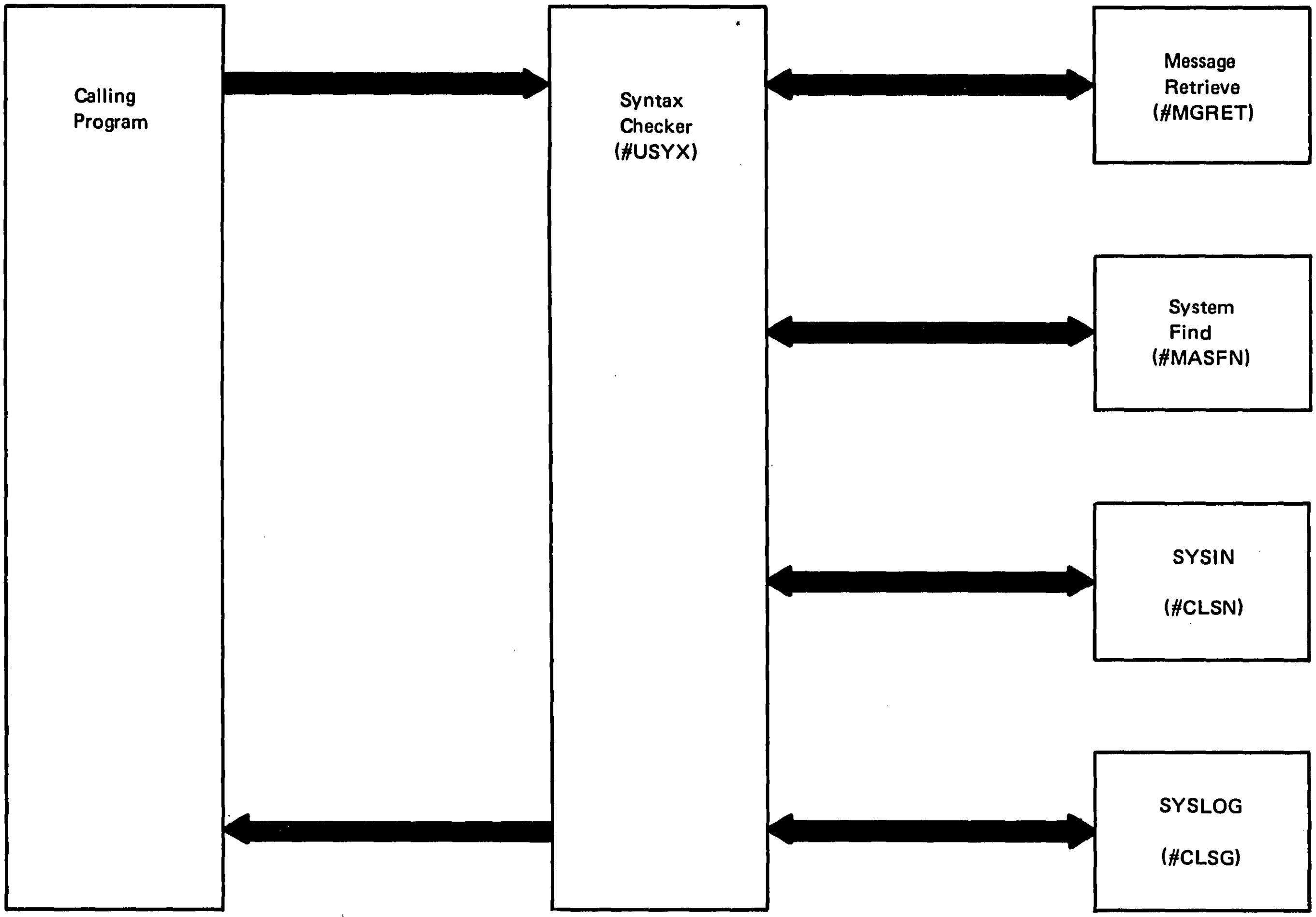


Figure 6-23. Syntax Checker Control Flow (#USYX)

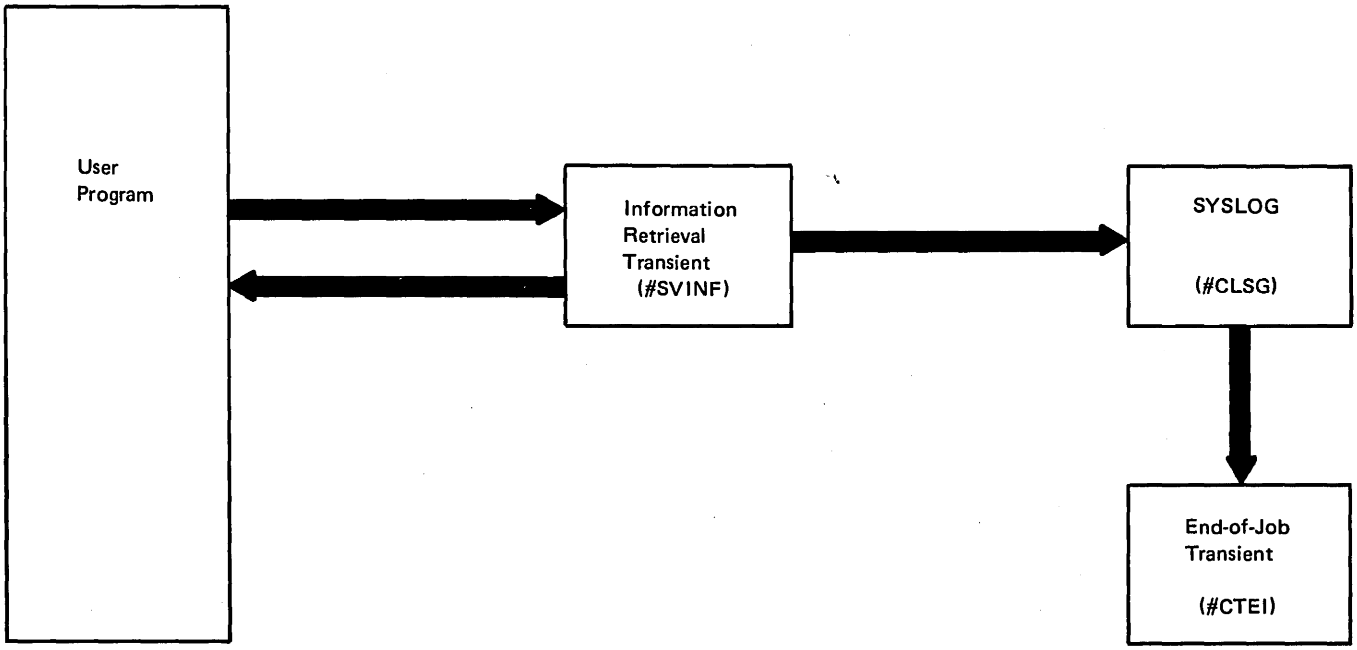
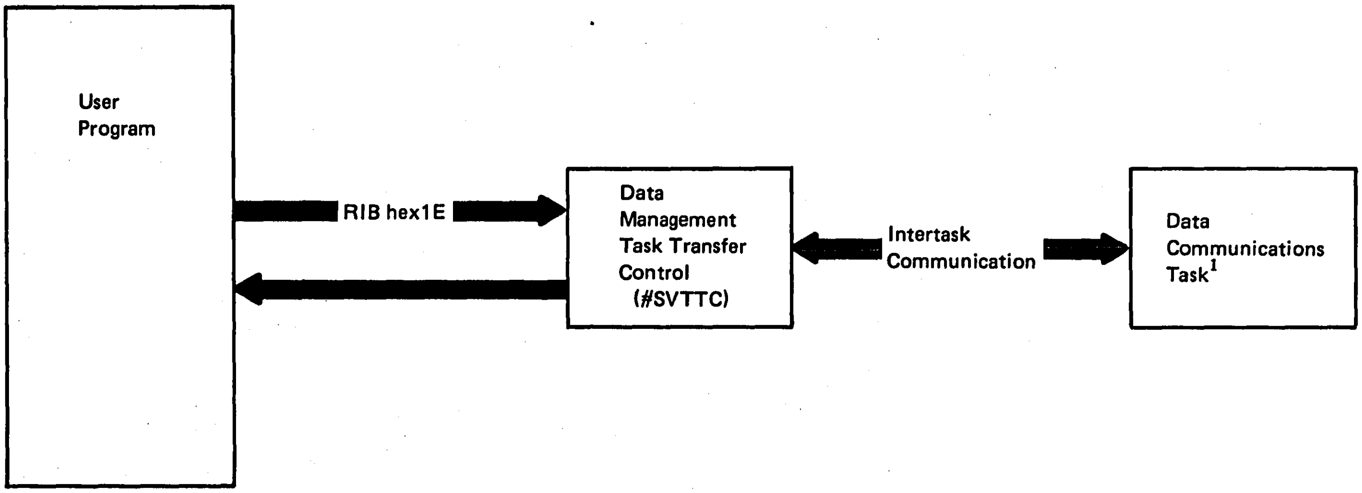

User Program
Information Retrieval Transient (#SVINF)
SYSLOG (#CLSG)
End-of-Job Transient (#CTEI)

Figure 6-24. Information Retrieval Control Flow


User Program
RIB hex1E
Data Management Task Transfer Control (#SVTTC)
Intertask Communication
Data Communications Task[1]

[1] See *IBM System/34 System Support Program Logic Manual: Data Communications,* LY21-0051.

Figure 6-25. Data Management Task Transfer Control Flow

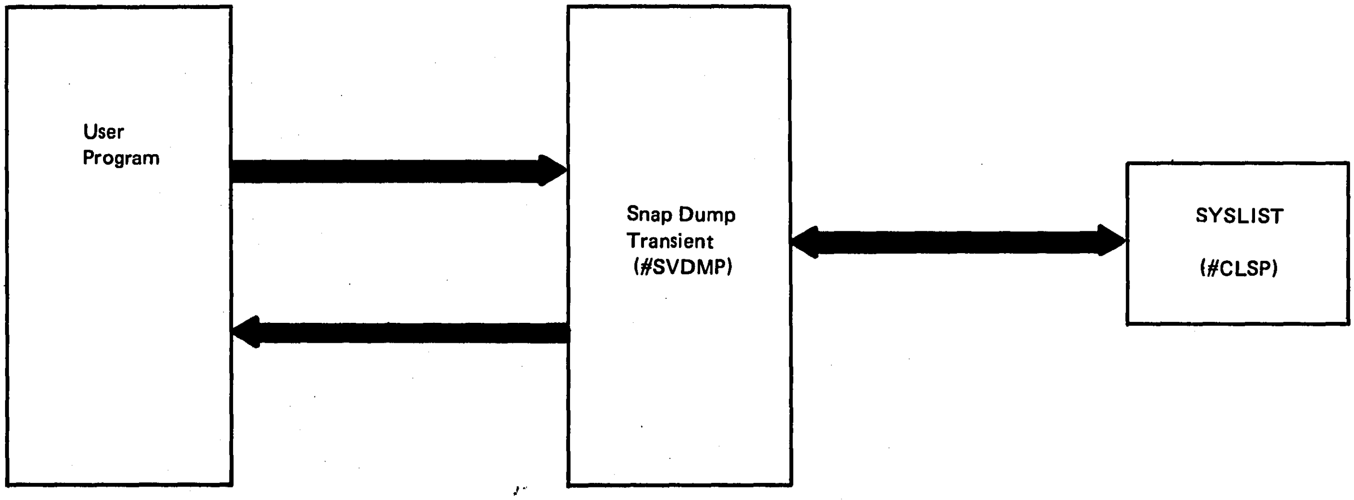


Figure 6-26. Snap Dump Control Flow

FIND A LIBRARY PARAMETER LIST

The find a library parameter list is a 10-byte parameter required when the find a library routine (#MAFLB) is called. The caller of #MAFLB places the address of the leftmost byte of this parameter list in XR2.

Figure 6-27 shows the format and contents of the parameter list.

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes	Description
0	FLBNAME	8	Library name
8	FLBF1A	2	Address of F1 in AFA

Figure 6-27. Find a Library Parameter List

SYSTEM FIND PARAMETER LIST

The system find parameter list is required when single name find routine (#MASFN) is called. The caller of #MASFN places the address of the leftmost byte of the parameter list in XR2.

Figure 6-28 shows the format of the system find parameter list for input. Figure 6-29 shows the two formats (loader and user) for output.

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes	Description
0	$FNDDTYP	1	Library type
	$FNDMLD8		X'08' Load module
	$FNDMSB8		X'04' Subroutine
	$FNDMSRC		X'02' Source module
	$FNDMPRC		X'01' Procedure
1	$FNDDNM8	8	Member name
9	$FNDDOPR	1	Operation switches
	$FNDMSYS		X'80' Search system library only
	$FNDMLDR		X'40' Build loader parm list only
	$FNDMUSE		X'20' Search user library only
	$FNDMULB		X'10' Search user library in $FNDDF1A
	$FNDMRF1		X'08' Return library F1 address
A	$FNDDF1A	2	Library format 1 address
A	$FNDDLDA	2	Load address

Figure 6-28. Single Name Find Input Parameter List

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes	Description
		(Loader Format)	
0	$FNDDADR	3	Library disk address
3	$FNDDNOS	1	Number of text sectors
4	$FNDDLNK	2	Link edit address
6	$FNDDSCT	2	Start address
8	$FNDDRLD	1	RLD displacement
9	$FNDDTNS	1	Total number of sectors
A	$FNDDLDA	2	Load address
		(User Format)	
0	$FNDDADR	3	Library disk address
3	$FNDDNOS	1	Number of text sectors or record length
4	$FNDDLNK	2	Link edit address
	$FNDDNST	2	Number of statements (S or P)
6	$FNDDSCT	2	Start address (O modules)
	$FNDDF1F	2	Format 1 address of the library in which the member was found (if requested)
8	$FNDDRLD	1	RLD displacement (O modules)
9	$FNDDCRS	1	Program size-sectors (O modules)
A	$FNDDATT	3	Member attribute bytes
D	$FNDDMRT	1	Type O — MRTMAX count Type P — X'FF' MRT
E	$FNDDREL	1	Module release level
F	$FNDDTOT	2	Module size in sectors
11	$FNDDCOM	1	Completion code
	$FNDMSYR		X'80' Found in sys lib
	$FNDMUSR		X'40' Found in user lib

Figure 6-29. System Find Output Parameter List

LIBRARIAN FIND PARAMETER LIST

The librarian find parameter list is a 24-byte parameter required when the library find routine ($MAFND or $MALFN) is called. The caller of $MAFND or $MALFN places the address of the leftmost byte of this parameter in XR2.

The transient version ($MAFND) requires a 25-byte work area following the parameter list.

Figure 6-30 shows the format and contents of the parameter list.

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes	Description
0	LFDDTYPE	1	Library type
1	LFDDNAME	8	Member name or partial name[1]
9	LFDDFUNC	1	Function byte
	LFDMWRTS		X'80' = Write buffer after find
	LFDMTYPO		X'08' = Load module find
	LFDMTYPR		X'04' = Subroutine member find
	LFDMTYPS		X'02' = Source member find
	LFDMTYPP		X'01' = Procedure member find
A	LFDDFNC2	1	Function Byte 2
	LFDMSYSL		X'80' = Search system library
	LFDMUSEL		X'40' = Search designated user library
B	LFDDREPL	1	Reply byte
	LFDMFNDS		X'80' = Found a member
	LFDMFTBS		X'40' = Member found in this buffer
	LFDMEOFS		X'20' = No more members
	LFDMWTBS		X'10' = Previous buffer written
	LFDMSYSR		X'08' = Found in system library
	LFDMUSER		X'04' = Found in designated library
	LFDMFND1		X'01' = Found member in current library

[1] If partial name, eighth byte is length of name

Figure 6-30 (Part 1 of 2). Librarian Find Parameter List

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes	Description
C	LFDDLBF1	2	F1 address of library to search (if 0, check LFDMSYSL and LFDMUSEL)
E	LFDDIRPT	2	Address of found directory entry
10	LFDDBUF@	2	Address of caller's buffer
12	LFDDBUFS	1	Buffer size in sectors
13	LFDDIOBS	3	Save area for SSS
16	LFDDNSEC	2	Number of sectors in member

Figure 6-30 (Part 2 of 2). Librarian Find Parameter List

SOURCE LIBRARY GET PARAMETER LIST

The source library get parameter list is a 19-byte parameter required when the source library get routine (#MASGT or #MASYL) is called. The parameter list has two formats. Figure 6-31 shows the format of the input parameter list, used when a find is requested. Figure 6-32 shows the format of the output parameter list, used when the caller requests a get.

The caller of #MASGT or #MASYL must place the address of the leftmost byte of this parameter in XR2. The transient version (#MASGT) requires a 15-byte work area following the parameter list.

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes	Description
0	GETDFNCT	1	Function byte (input options)
	GETMFIND		X'80' Find request
	GETMFRST		X'40' Get first
	GETMNEXT		X'20' Get next
1	GETDTYPE	1	Member type
	GETMSRCE		S – Source
	GETMPROC		P – Procedure
2	GETDNAME	8	Member name
A	GETDLBF1	2	F1 address of library to search; if 0, search system library
C	GETDADDR	2	Record buffer address
E	GETDSIZE	1	Buffer size in bytes
F	GETDFCT2	1	Function byte 2
	GETMSOBK		X'20' Return start of blanks indicator in record buffer (X'FF')
	GETMREPR		X'10' Reprime request
10	GETDIOBF	2	Read source buffer
12	GETDBFSZ	1	Buffer size in sectors

Figure 6-31. Source Library Get Input Parameter List (find format)

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes	Description
0	GETDREPL	1	Reply byte (output options)
	GETMTER		X'08' Terminal error
	GETMTRNC		X'04' Truncated record
	GETMNFND		X'02' No find
	GETMEOF		X'01' End of file
1	GETDSSS	3	Sector address processing now
4	GETDEND	3	Sector address of last record
7	GETDBNOW	2	Address of byte in I/O buffer to process
9	GETDBEND	2	Address of last byte in I/O buffer to process
B	GETDLGTH	1	Record length
C	GETDADDR	2	Record buffer size
E	GETDSIZE	1	Record buffer address
F	GETDCOMP	1	Completion switch
10	GETDIOBF	2	I/O buffer address
12	GETDBFSZ	1	I/O buffer size

Figure 6-32. Source Library, Get Output Parameter List (get format)

AFA ACCESS PARAMETER LIST

The AFA access parameter list is required when the active format 1 area access routine (#CSAF) is called. The caller of #CSAF places the address of the leftmost byte of the parameter list in XR2.

Figure 6-33 shows the format of the AFA access parameter list.

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes	Description
0	AF1DFNCT	1	Function byte
	AF1MREAL		X'80' Real I/O area address
	AF1MVFID		X'40' Verify ID
	AF1MGTLB		X,20' Get by label
	AF1MGTAD		X'10' Get by address
	AF1MGTNM		X'08' Get by name
	AF1MTPUT		X'04' Put request
	AF1MDATE		X'02' Verify date request
	AF1MF1MV		X'01' Move F1 to user I/O area
1	AF1DRTRN	2	Return code
	AF1MNORM		X'40' Good completion
	AF1MINVD		X'41' Invalid request
	AF1MNTMT		X'44' Request not met
3	AF1DF1PT	3	Pointer to format 1
6	AF1DNMLB	8	Format 1 name or label
E	AF1DUNIT	1	Unit
	AF1MF1		X'00' Disk unit
	AF1MI1		X'10' Diskette unit
	AF1MNU		X'FF' No unit specified
F	AF1DDATE	3	Date
12	AF1DIOAR	3	User I/O area address

Figure 6-33. AFA Access Parameter List

DUPLICATE KEY DISPLAY PARAMETER LIST

The duplicate key display parameter list is required when the duplicate key display routine (#CSDK) is called. The caller of #CSDK must place the address of the leftmost byte of the parameter list in XR1.

Figure 6-34 shows the format of the duplicate key halt parameter list.

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes	Description
0	DKHDFLAG	1	Flag byte
	DKHMKEYR		X'80' Key address real
	DKHMMSGR		X'40' Message addr real
	DKHMPROC		X'08' Option 0
	DKHMRTRY		X'04' Option 1
	DKHMCNCL		X'02' Option 2
	DKHMTER1		X'01' Option 3
1	DKHDKEYA	2	Key address (left byte)
3	DKHDMSGA	2	Message address (left byte)
5	DKHDCOMP	4	Component ID
9	DKHDMICN	2	MIC number
B	DKHDF1AD	2	Format 1 address

Figure 6-34. Duplicate Key Display Parameter List

SUPERVISOR TASK ATTACH PARAMETER LIST

The task attach parameter list is a 16 byte parameter required when the supervisor task attach transient (#SVAT) or the supervisor task detach transient (#SVAU) is called. The caller of #SVAT or #SVAU places the address of the leftmost byte of the parameter in XR1.

Figure 6-35 shows the format and contents of the parameter list.

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes	Description
0	$ATLOAD	1	Loader parameter list offset
1	$ATFLAG	1	First flag byte offset
	$ATCREAT		X'80' = Create new TCB
	$ATREAL		X'40' = Real link address
	$ATTUBAS		X'20' = on = TUB address off = JCB address
	$ATNONAM		X'10' = on = Do not assign job name off = Assign job name
	$ATDATA		X'08' = Put data in session work area
	$ATPRIV		X'04' = Task privileged
	$ATNSWAP		X'02' = Not swappable
	$ATERPRM		X'01' = Free attach parameter list
	$ATINIT		Start Initiator
	$ATSPOOL		Start Spool
	$ATBATCH		Start Batch
2	$ATMSSIZ	1	Number of 2K main storage blocks
3	$ATPRIOR	1	Priority of new task
4	$ATTUB@	2	TUB address

Figure 6-35 (Part 1 of 2). Supervisor Task Attach Parameter List

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes	Description
4	$ATJCB@	2	JCB address
6	$ATSSSN	4	SSSN value of next transient
A	$ATFLAG1	1	Second flag byte
	$ATRFRSH		X'80' = Refresh
	$ATCOMON		X'40' = Program has Common
	$ATQKINT		X'20' = Attach Initiator
	$ATINCJC		X'10' = Increment job count
	$ATSYSTK		X'08' = Attach system task
B	$ATTSKID	1	Task ID of attached task
C	$ATDATA@	2	Address of data to put
E	$ATLENG	1	Length of attach parameter list

Figure 6-35 (Part 2 of 2). Supervisor Task Attach Parameter List

SUPERVISOR TASK ATTACH ERROR RETURN CODE

Supervisor task attach supplies an error return code in XR1 if the task attach function is not successful. The format of the error code in XR1 is 00XX, where XX is the error return code. Figure 6-36 shows the format and contents of the error return code.

Label	XX	Description
$ATERR01	X'01'	Not enough storage space
$ATERR02	X'02'	Task nonswappable and not enough storage space
$ATERR03	X'03'	Task nonswappable and storage requirements will disable task
$ATERR04	X'04'	Assign failure on TCB
$ATERR05	X'05'	Assign failure on RB
$ATERR06	X'06'	Allocate failure for swap area
$ATERR07	X'07'	Allocate failure for work station work area
$ATERR08	X'08'	Assign failure for ACE

Figure 6-36. Supervisor Task Attach Error Return Codes

SYNTAX SPECIFICATION MODULE

The syntax specification module is used by the syntax checker (#USYX) to determine if the control statement passed is valid. Each utility has its own syntax specification module that resides in the system library. Figure 6-37 lists the syntax specification modules.

Utility	Syntax Specification Module Name
Library maintenance	$MASPC
Basic data exchange	$BITAB
Disk copy/display	$COTAB
File delete	$DETAB
Diskette copy	$DUTAB
History file display	$HISTAB
Diskette labeling and initialization	$INTAB
VTOC display	$LABTB
Create message member	$MGTAB
Set	$SETSM
File build	$FBTAB
Disk compress	$PAKTB
Menu build	$BMTB
Display format generator	$SFTB

Figure 6-37. Syntax Specification Modules

The USCTMODN field of the communication table passed by the calling utility contains the specification module name, and the USCTSSMP field contains the specification module address. The specification module is loaded following #USYX in main storage or at the address specified, and consists of one or more control statement syntax specifications as shown in Figure 6-38.

There is one control statement syntax specification for each possible control statement for the utility.

	Verb Record
Control Statement Syntax Specification	0 to 64 Parameter Records (positional, keyword, or both)
	0 to 64 Substitution Tables as Required to Define Parameter Values
	Valid Combination Records as Required to Define the Acceptable Combinations of Parameter Values
	Verb Record
	0 to 64 Parameter Records
	0 to 64 Substitution Tables
	Valid Combination Records

Note: The format and contents of the verb record, parameter records, substitution tables, and valid combination records, are included in this Data Areas section.

Figure 6-38. Syntax Specification Module

VERB RECORD – VERB

The verb record is a 14-byte record in the control statement syntax specification that contains the name and numeric identifier of a control statement verb. The first verb is at the main storage address specified in USCTSSMP and the current verb record is at the address specified in USCTVRB@. When a control statement is read by the syntax checker (#USYX), this record is checked to determine if the verb passed matches the verb of this verb record. If not, the VRBDNEXT fields contains the address of the next verb record to check.

Figure 6-39 shows the format and contents of the verb record.

PARAMETER RECORDS

There are two types of parameter records; positional and keyword. Parameter records follow the verb record in the control statement syntax specification. After the verb record for the control statement has been determined, the parameter records indicate the valid parameters and parameter values for the control statement.

POSITIONAL PARAMETER RECORD – POSIT

The positional parameter record, POSIT, is a 13-byte or 21-byte record that contains information for a single positional parameter. The USCTPOS@ field of the communication table contains the address of the current record. Figure 6-40 shows the format and contents of the positional parameter record.

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes	Description
0	VRBDHEAD	1	Constant V
1	VRBDVBID	1	Verb ID, X'01' to X'FF'
2	VRBDNAME	8	Verb name
A	VRBDNEXT	2	Pointer to next verb record
C	VRBDVALC	2	Pointer to a set of valid combination records (VALCM)

Figure 6-39. VERB Verb Record

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes	Description
0	POSDHEAD	1	Constant P
1	POSDPSID	1	Unique ID for this record X'01' to X'FB'
2	POSDNAME	1	Position number of the parameter in this control statement
3	POSDIND1	1	Parameter flag 0=optional parameter 1=required parameter
4	POSDRESV	1	Reserved

5	POSDATTR	1	Attribute
			L=label
			Z=partial name
			N=numeric characters
			C=numeric characters to be converted to binary
			S=string of characters except blank, comma, and hyphen
			Q=character string optionally enclosed in single quotes
			D=date format

6	POSDMINM	2	Minimum parameter value
8	POSDMAXM	2	Maximum parameter value
A	POSDOUT@	1	Displacement of leftmost byte of parameter value in output area of communication table (USCTPOUT)
B	PASDSIZE	1	Length of parameter value in output area of communication table (USCTPOUT)
C	POSDDEFL	1	Default length X'08'=default length X'00'=no default
D	POSDDEFV	8	Default value for parameter (optional field)

Figure 6-40. POSIT Positional Parameter Record

KEYWORD PARAMETER RECORD – KEYWD

The keyword parameter record, KEYWD, is a 22-byte or 30-byte record that contains information for a single keyword parameter. The USCTKEY@ field of the communication table contains the address of the current record. Once a keyword parameter has been found in the control statement, the parameter records are scanned to find the matching keyword parameter record. When a match is found, the keyword parameter record indicates where the value should be placed in the output area (USCTPOUT) of the communication table.

Figure 6-41 shows the format and contents of the keyword parameter record.

SUBSTITUTION TABLE – SUBEN

A substitution table is an 18-byte table that consists of a 3-byte header record and one or more 15-byte entry records. A substitution table specifies a parameter that should have a value substituted for it in the output area (USCTPOUT) of the communication table and the value that should be substituted. From 0 to 64 substitution tables can follow the last parameter record in a control statement syntax specification.

Figure 6-42 shows the format and contents of a header record. Figure 6-43 shows the format and contents of an entry record.

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes	Description
0	KEYDHEAD	1	Constant K
1	KEYDKYID	1	Unique number for this record X'01' to X'FB'
2	KEYDNAME	8	Parameter keyword
A	KEYDIND1	1	Parameter flag 0=optional parameter 1=required parameter
B	KEYDRESV	1	Reserved

C	KEYDATTR	1	Attribute
			L=label
			Z=partial name
			N=numeric characters
			C=numeric characters to be converted to binary
			S=string of characters except blank, comma and hyphen
			Q=character string enclosed in single quotes
			D=date format

D	KEYDMINM	3	Minimum parameter value
10	KEYDMAXM	3	Maximum parameter value

Figure 6-41 (Part 1 of 2). KEYWD Keyword Parameter Record

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes	Description
13	KEYDOUT@	1	Displacement of leftmost byte of parameter value in output area (USCTPOUT) of communication table
14	KEYDSIZE	1	Length of parameter value in communication table
15	KEYDDEFL	1	Default length X'08'=default length X'00'=no default
16	KEYDDEFV	8	Default value for parameters (optional field)

Figure 6-41 (Part 2 of 2). KEYWD Keyword Parameter Record

Displacement of Leftmost Byte	Label	Length in Bytes	Description
0	SBTDHEAD	1	Constant S
1	SBTDSBID	1	Unique number of parameter record associated with its substitution table
2	SBTDREST	1	Substitution value flag 1=only the specific values in the substitution table entries are meaningful 0=specific values in the substitution table entries are not all of the meaningful values

Figure 6-42. SUBEN Substitution Table Header Record

Displacement of Leftmost Byte	Label	Length in Bytes	Description
0	SBEDHEAD	1	Constant E
1	SBEDRCID	1	Unique number for this record X'01' to X'FF'
2	SBEDSUBS	3	Substitution value

5	SBEDATTR	1	Attribute
			L=label
			Z=partial name
			N=numeric characters
			C=numeric characters to be converted to binary
			S=string of characters except blank, comma, and hyphen
			Q=character string enclosed in single quotes
			D=date format

6	SBEDOUT@	1	Displacement of leftmost byte of substitution value in output area of syntax checker communication table
7	SBEDSIZE	1	Length of substitution value in communication table
8	SBEDVALU	8	Real value of parameter as it would appear in the control statement

Figure 6-43. SUBEN Substitution Table Entry Record

VALID COMBINATION RECORD

The valid combination record, VALCM, is a 7-, 10-, 13-, 16-, 19-, 22-, 25-, or 28-byte record used to determine if the parameters and/or parameter values are used in valid combination in the control statement. The VRBDVALC field of the verb record (VERB) contains the address of the set of valid combination records.

Figure 6-44 shows the format and contents of a valid combination record.

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes	Description
0	VALDHEAD	1	Constant C
1	VALDRCID	1	ID of a parameter record associated with this record
	VALDCOMP	1	Indicator for complemented entries; that is, parameter flag meaning or value is complemented
	VALDNMBR	1	Number of entries
4	VALDENT1	1	Parameter value flag X'FE'=parameter must not be present X'FD'=parameter is required and some value must be specified X'FC'=parameter is ignored X'FB' to X'01'=substitution table entry that contains the parameter value
5	VALDNXT1	2	Pointer to another valid combination record, or X'FFFF' indicating the end of the set of valid combinations

Note: Any pair of the following fields may appear (VALDEn with VALDXn) in the valid combination record as needed to describe the valid parameter combinations.

7	VALDENT2	1	Same as VALDENT1
8	VALDNXT2	2	Same as VALDNXT1
A	VALDENT3	1	Same as VALDENT1
B	VALDNXT3	2	Same as VALDNXT1
D	VALDENT4	1	Same as VALDENT1
E	VALDNXT4	2	Same as VALDNXT1
10	VALDENT5	1	Same as VALDENT1
11	VALDNXT5	2	Same as VALDNXT1
13	VALDENT6	1	Same as VALDENT1

Figure 6-44 (Part 1 of 2). VALCM Valid Combination Record

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes	Description
14	VALDNXT6	2	Same as VALDNXT1
16	VALDENT7	1	Same as VALDENT1
17	VALDNXT7	2	Same as VALDNXT1
19	VALDENT8	1	Same as VALDENT1
1A	VALDNXT8	2	Same as VALDNXT1

Figure 6-44 (Part 2 of 2). VALCM Valid Combination Record

SYNTAX CHECKER COMMUNICATION TABLE — USCTABLE

The syntax checker communication table, USCTABLE, is a table indicated by the utility that calls the syntax checker (#USYX). The first 2 bytes of the syntax checker parameter list passed by the utility contain the address of the communication table.

The communication table is returned to the utility to indicate the following control statement information:

- Errors that were detected
- Parameters specified

Figure 6-45 shows the format and contents of the communication table.

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes in Decimal	Description
0	USCTRES1	1	Reserved
1	USCTOUTL	1	Length of output area (from X'00' to X'FF')
2	USCTINBP	2	Address of input buffer
4	USCTRES2	1	Reserved
5	USCTINBL	1	Length of input buffer (from X'00' to X'78')
6	USCTSSMP	2	Address of syntax specification module
8	USCTMODN	8	Syntax specification module name

Note: The syntax checker overlays the USCTMODN field with the following 8 bytes.

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes in Decimal	Description
8	USCTERRT	1	Reserved
9	USCTERRC	2	Completion code X'FFFF'=no errors X'FFFE'=statement continued X'nnnn' =error, where nnnn is the message identification code (MIC) that is issued
B	USCTERRP	2	Address of the first byte of the field in the area of error
D	USCTVRBN	1	Hexadecimal constant associated with the control statement verb
E	USCTPARF	1	Hexadecimal constant ID of the first record after the verb record
F	USCTPARL	1	KEYNUM of last (current) POSIT or KEYWD record processed

Figure 6-45 (Part 1 of 2). USCTABLE Syntax Checker Communication Table

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes in Decimal	Description
10	USCTPMAP	8	Indication of parameters specified in the control statement
18	USCTCNTL	8	Control area to allow syntax checker to be reentrant with area subdivided as follows:
	USCTVRB@	2	Main storage address of current verb record
	USCTPOS@	2	Main storage address of current POSIT record
	USCTKEY@	2	Main storage address of current KEYWD record
		2	Reserved
20	USCTERNM	8	Contains from one to eight characters for insert into error message when USCTERRC contains a MIC
28	USCTPOUT	0 to 255	Output area for the syntax checker that contains the parameter values specified on the control statement or values substituted for the parameter values

Figure 6-45 (Part 2 of 2). USCTABLE Syntax Checker Communication Table

SYNTAX CHECKER PARAMETER LIST

Each time the syntax checker (#USYX) is called, register 2 must contain the address of a 6-byte parameter list that indicates the (1) address of a communication table, (2) address of the verb list to use, (3) name or address of the specification module, and (4) source of input. There is one parameter list for each verb list that can be passed. Figure 6-46 shows the general format and contents of a parameter list.

Displacement of Leftmost Byte	Label	Length in Bytes	Description
0	USCTMP	2	Address of communication table
2	VBLSTP	2	Address of verb list
4	ENTERD	1	Specification module information X'00'=Load the specification module named in the communication table X'01'=Do not load a specification module The address of a supplied specification module is in the communication table
5	ENTRY	1	Control statement input information X'00'=Read a control statement from the sysin device X'01'=The communication table indicates the address of the control statement X'02'=The statement being scanned is continued from a previous statement X'04'=Synonyms (more than one form) for verb names permitted

Figure 6-46. Syntax Checker Parameter List

Chapter 7. System Maintenance Programs

Introduction

The System/34 diagnostic aids that execute in main storage are:

- APAR ($FEAPR)
- Dump ($FEDMP)
- Program temporary fix installation program ($FEFIX)
- Patch ($FEPCH)
- Trace select ($FETRC)
- Error recording analysis procedure ($ERAP)

Introductory information about the diagnostic aid programs (their functions and how to run them) is contained in the *Data Areas Handbook.*

$FEAPR, $FEDMP, $FEFIX, $FEPCH, and $FETRC each require 14K bytes of main storage for program execution.

Method of Operation

This section contains function diagrams for the diagnostic aids. They are:

- APAR utility (Diagram 7.1)
- Dump utility (Diagram 7.2)
- PTF installation function (Diagram 7.3)
- Patch utility (Diagram 7.4)
- Trace select function (Diagram 7.5)
- ERAP utility (Diagram 7.6)

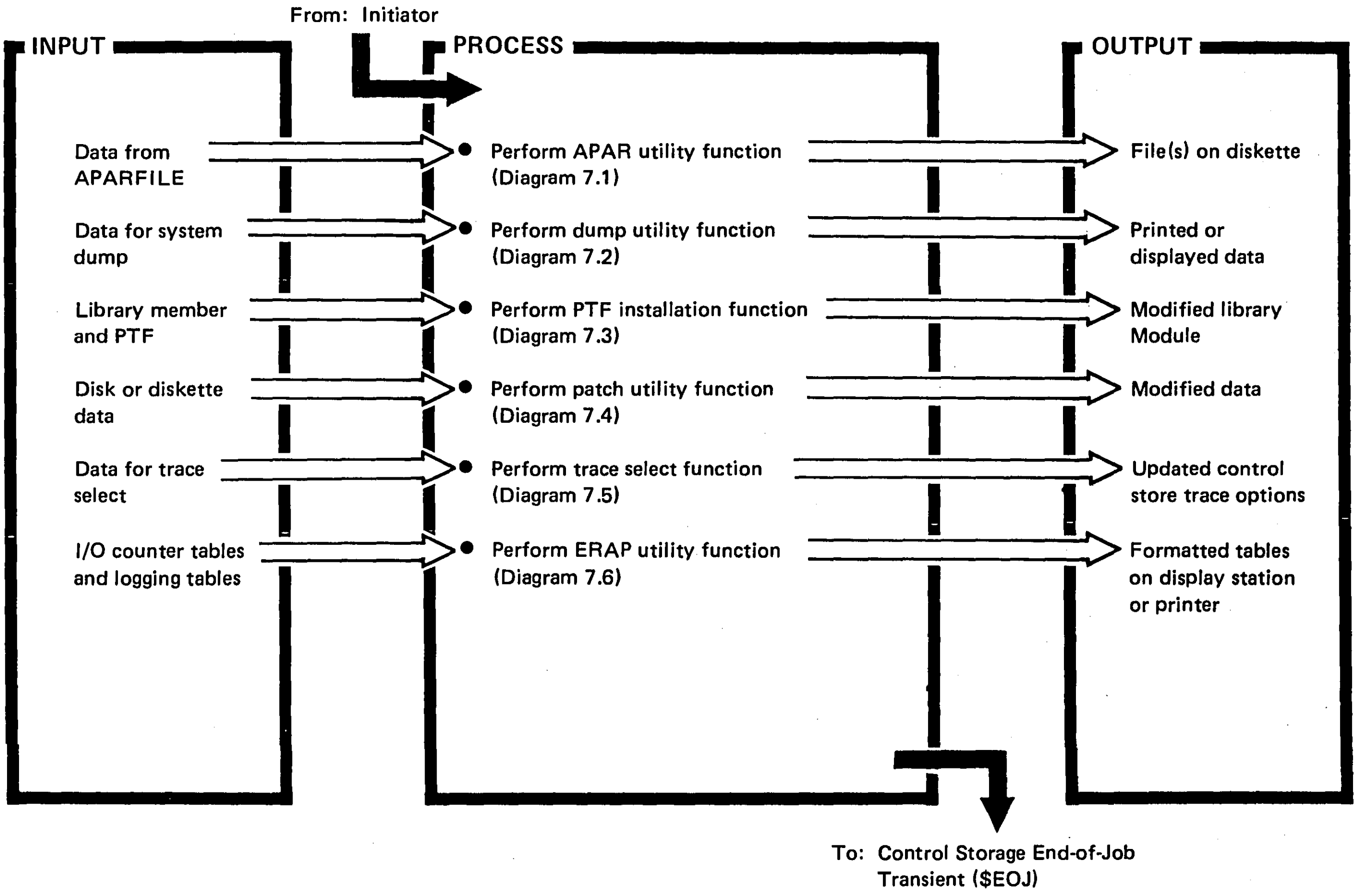
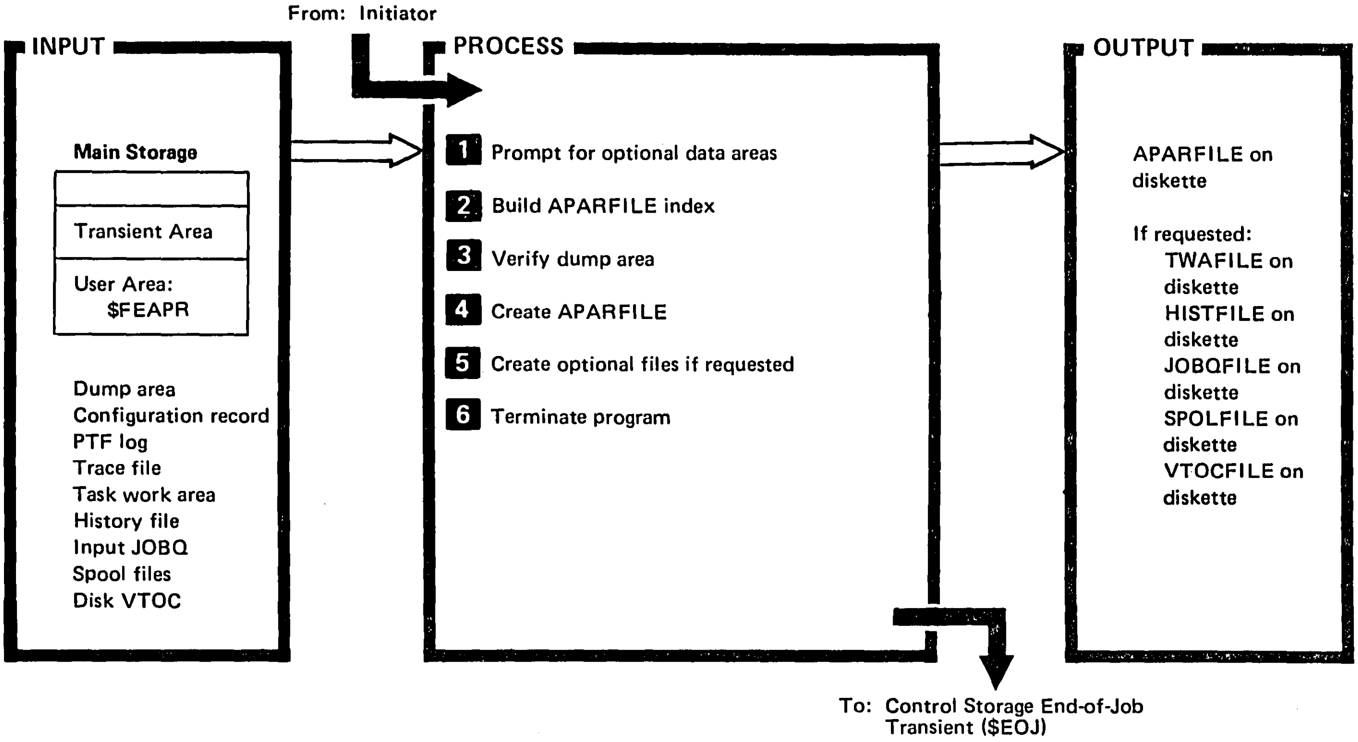
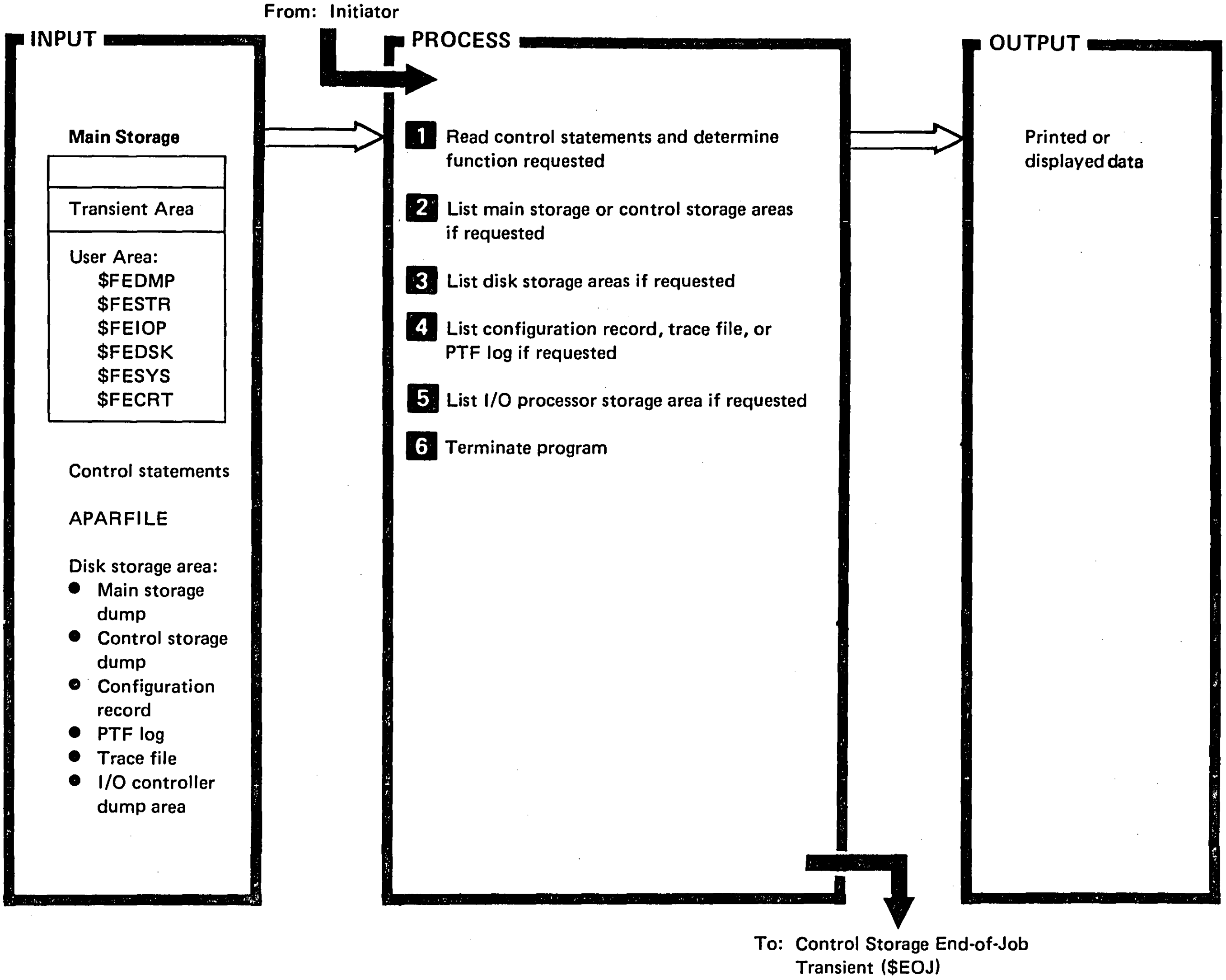

From: Initiator
INPUT
PROCESS
OUTPUT
Data from APARFILE
Perform APAR utility function (Diagram 7.1)
File(s) on diskette
Data for system dump
Perform dump utility function (Diagram 7.2)
Printed or displayed data
Library member and PTF
Perform PTF installation function (Diagram 7.3)
Modified library Module
Disk or diskette data
Perform patch utility function (Diagram 7.4)
Modified data
Data for trace select
Perform trace select function (Diagram 7.5)
Updated control store trace options
I/O counter tables and logging tables
Perform ERAP utility function (Diagram 7.6)
Formatted tables on display station or printer
To: Control Storage End-of-Job Transient ($EOJ)

Diagram 7.0. Overview of System Maintenance Programs



DESCRIPTION	MODULE/ ROUTINE
1 Issue prompt for additional system areas to be copied to diskette: ● Task work area (TWA). ● System trace file. ● System history file. ● Input JOBQ. ● Spool files. ● Disk VTOC.	$FEAPR
2 Build index record containing size and record offset within APARFILE of system data areas to be included. Write index record as first record of APARFILE.	
3 Read first sector of main storage dump area from disk. Check dump validity flag (SCADPIND) in SCA to determine dump validity.	
4 Write system area data to APARFILE: ● Disk dump area. ● Configuration record. ● #LIBRARY PTF log. ● Trace file (if requested).	
5 Create optional files on diskette if requested: ● History file (HISTFILE). ● Input JOBQ (JOBQFILE). ● Spool files (SPOLFILE). ● Disk VTOC (VTOCFILE).	
6 Pass control to end of Job control storage transient ($EOJ) to terminate program.	

Diagram 7.1. Perform APAR Utility Function ($FEAPR)



From: Initiator

INPUT

Main Storage

Transient Area

User Area:
$FEDMP
$FESTR
$FEIOP
$FEDSK
$FESYS
$FECRT

Control statements

APARFILE

Disk storage area:
- Main storage dump
- Control storage dump
- Configuration record
- PTF log
- Trace file
- I/O controller dump area

PROCESS

1 Read control statements and determine function requested

2 List main storage or control storage areas if requested

3 List disk storage areas if requested

4 List configuration record, trace file, or PTF log if requested

5 List I/O processor storage area if requested

6 Terminate program

OUTPUT

Printed or displayed data

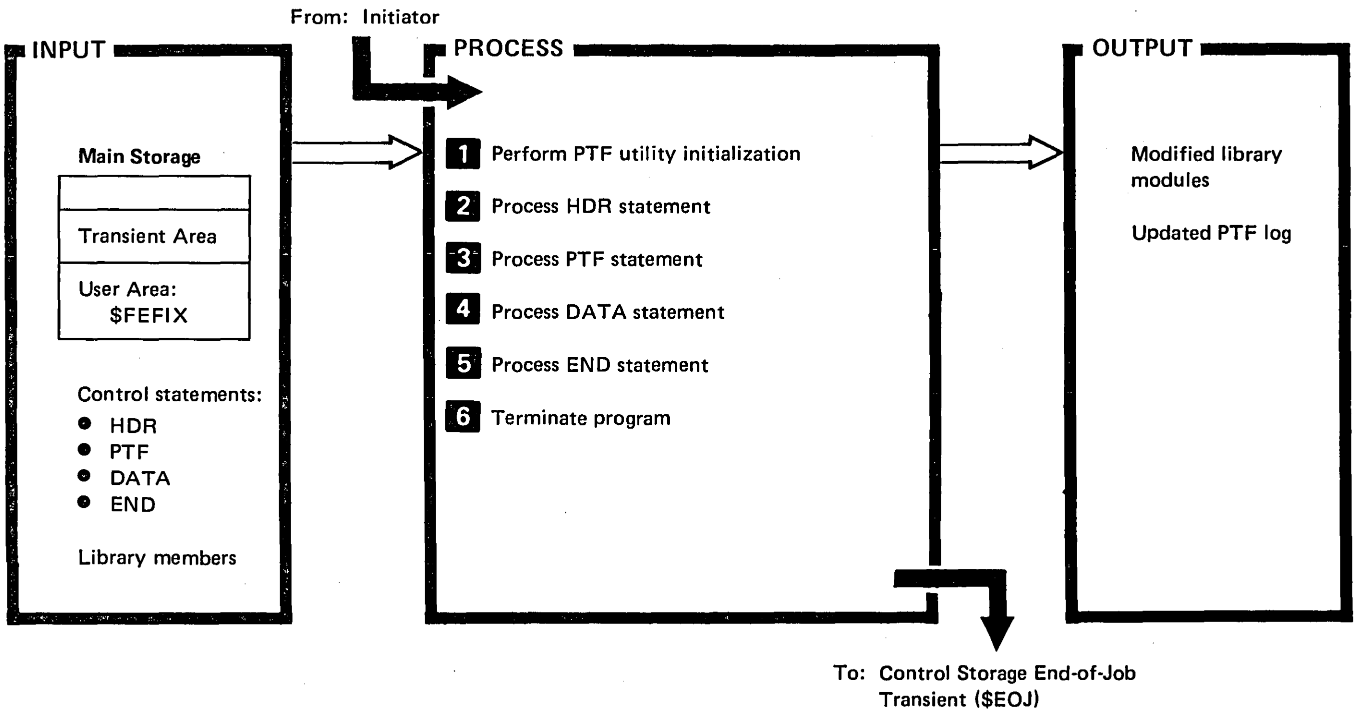
To: Control Storage End-of-Job Transient ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Use syntax checker (#USYX) to read and check control statements. If verb is DUMP, save keyword information. If requested function is: ● Dump main storage or control storage areas, go to 2. ● Dump selected disk storage areas, go to 3. ● Dump configuration record, trace file, or PTF log area, go to 4. ● Dump I/O controller storage area, go to 5. If verb is END, go to 6.	$FEDMP
2 Initialize display screen, printer, APARFILE, and disk as required. Display summary of storage dump information.	$FESTR

Diagram 7.2 (Part 1 of 2). Perform Dump Utility Function ($FEDMP)

DESCRIPTION	MODULE/ ROUTINE
If output to printer specified: ● Prompt for type of storage dump and address limits. ● Dump requested main storage or control storage area to printer. ● Prompt for another storage area to dump and if E entered, return to 1 to read another control statement. If output to display screen:	$FESTR
● Display first segment of main storage or control storage and TCB address of abnormally terminated task.	$FECRT
● Accept user request to display other portions of storage. ● If E entered, return to 1 to read another control statement.	$FESTR
3 Initialize display screen, printer, diskette, and disk fields as required. Ⓐ Prompt for disk or diskette sectors to be displayed/printed. Ensure valid prompt response given. If character E response, go to 1 to read next control statement. Read data from disk or diskette.	$FEDSK
Output disk or diskette sectors requested: ● If output to display screen specified, put data to screen and roll screen up or down as requested. ● If E entered, go to Ⓐ to accept new address or end display.	$FECRT
● If output to printer specified, dump requested sectors to printer. ● Go to Ⓐ to accept new address or end dump.	$FEDSK
4 Initialize display screen, printer, APARFILE, and disk as required. If list configuration record request: ● Read configuration record. ● Format configuration record for output to printer or display screen. ● Display or print selected fields from system configuration record. If list trace file request: ● Retrieve trace file data. ● Format trace file data for output to printer or display screen. ● If output to printer, start output with oldest sector entry. ● If output to display screen, start display with last set of entries and accept roll up and roll down keys to display additional entries. If list PTF log request: ● If input is disk, prompt for library name; otherwise, process #LIBRARY. ● Read PTF log. ● Format PTF log data for output to display screen or printer. ● If output to display screen, start display with first set of 40-byte PTF log entries and accept roll up and roll down keys to display additional entries. ● If output to printer, start output with first PTF log entry. When requested dump complete, return to 1 to read another control statement.	$FESYS
5 Initialize disk, APARFILE, and printer as required. Ⓑ Prompt for I/O controller storage dump area to output to printer; if character E entered, return to 1 to read another control statement. Read data from selected device storage dump area. Output selected device dump area data to printer. When requested I/O processor storage dump complete, go to Ⓑ to prompt for another dump request or end display.	$FEIOP
6 Pass control to end-of-job transient ($EOJ) to terminate program.	$FEDMP

Diagram 7.2 (Part 2 of 2). Perform Dump Utility Function ($FEDMP)

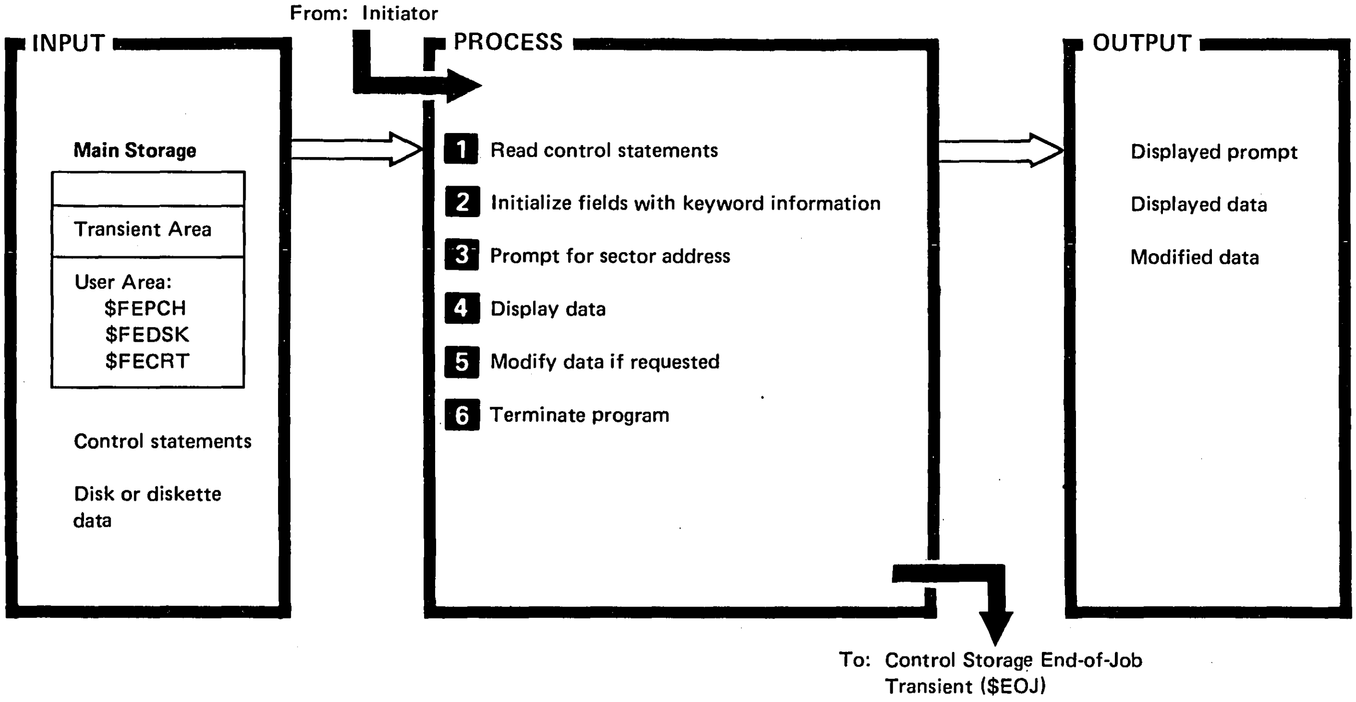


DESCRIPTION	MODULE/ ROUTINE
1 Allocate work file space. Determine SYSLOG device option. Read control statement from SYSIN device and go to appropriate processor: ● If HDR statement, go to **2**. ● If PTF statement, go to **3**. ● If DATA statement, go to **4**. ● If END statement, go to **5**.	$FEFIX
2 Syntax check HDR statement: ● If checksum not found (HDR statement blank), indicate field developed patch (ZAP) and save first four characters of system date and a '5' for the PTF ID. ● If checksum found, ensure PTF ID valid. Return to **1** to read next control statement.	
3 Syntax check PTF statement.	
If library parameter other than #LIBRARY given, find address of library format 1. Locate library module specified.	$MAFND
Read module from library and place in work file (if not already there).	Disk IOS
If module has overlays: ● Scan root RLDs to find RLD end, overlay table address, and number of root RLD sectors. ● If overlay request, read overlay table from module root and process any errors.	$FEFIX

Diagram 7.3 (Part 1 of 2). Perform PTF Installation Function ($FEFIX)

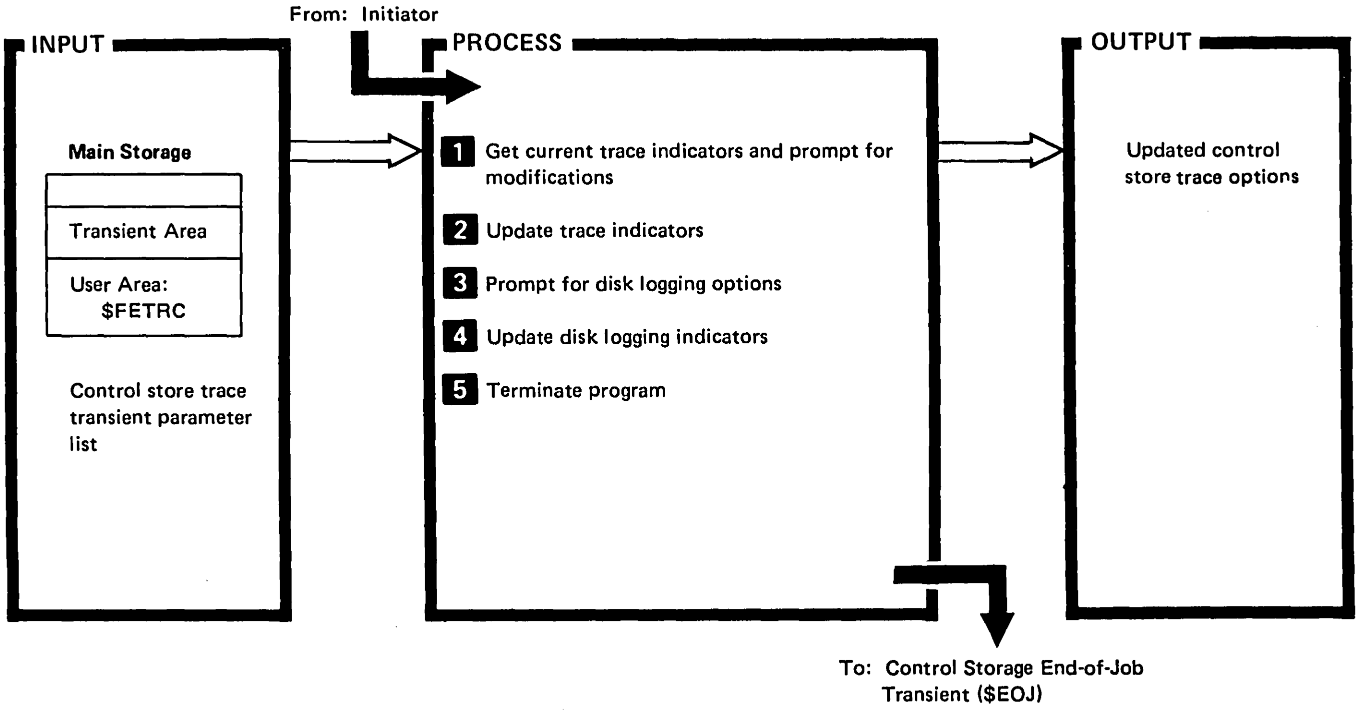
DESCRIPTION	MODULE/ ROUTINE
Store module information in PTF table.	$FEFIX
Return to 1 to read next control statement.	
4 Syntax check DATA statement.	
Read area to be patched in from work file.	
Delete RLDs in patch area.	
Merge RLDs from DATA statement into module RLDs.	
Insert patch data.	
Write patch data and RLDs back to work file.	
Return to 1 to read next control statement.	
5 Syntax check END statement.	
Set on SYSIN end of file.	
6 Log PTF (module name and ID — use system date and a 5 if ZAP) for each module.	$MAPTF
Set on PTF applied indicator in module directory entry.	$FEFIX
Transfer updated modules from work file back to library.	
Pass control to end-of-job transient ($EOJ) to terminate program.	

Diagram 7.3 (Part 2 of 2). Perform PTF Installation Function ($FEFIX)



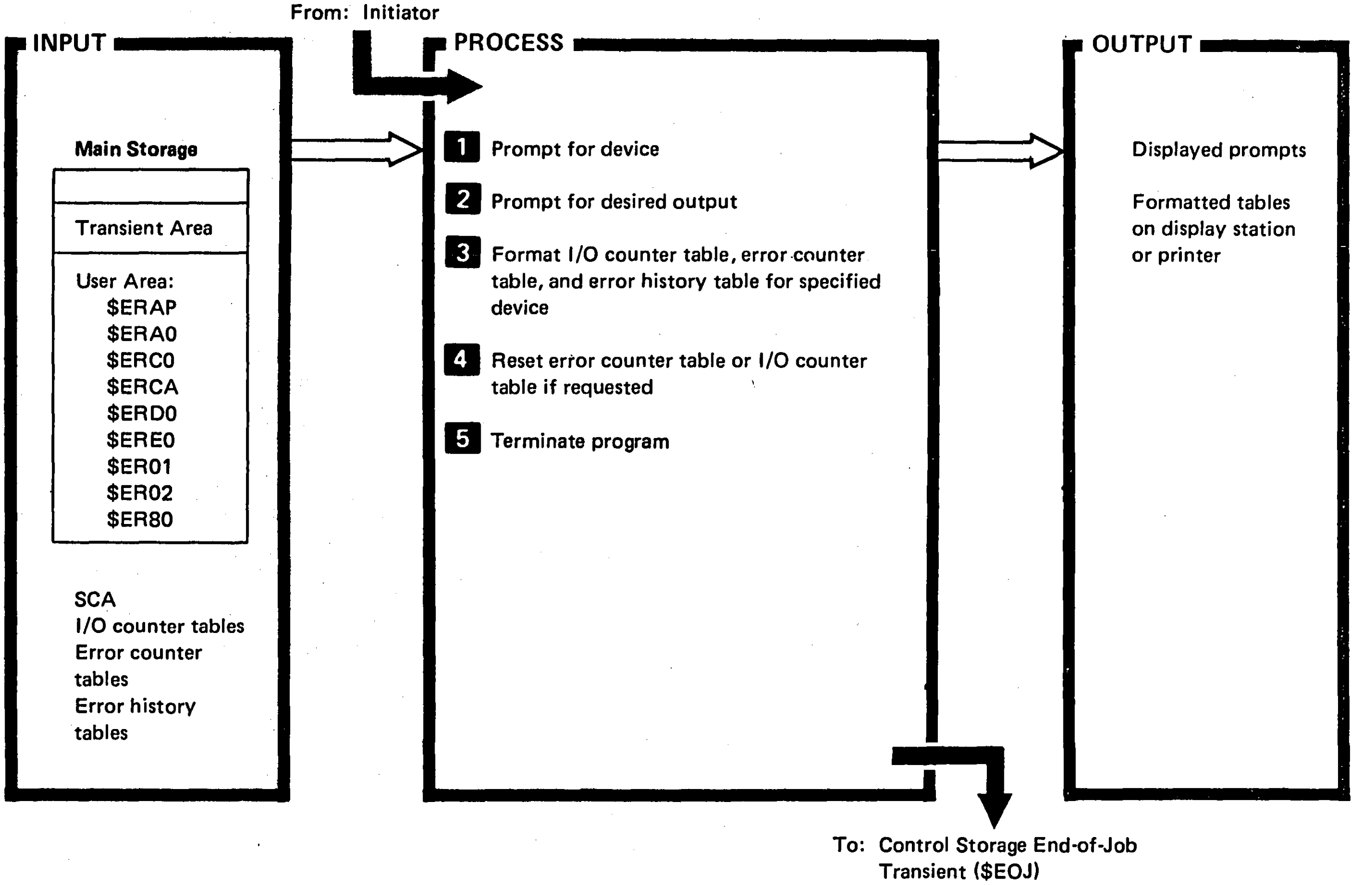
DESCRIPTION	MODULE/ ROUTINE
1 Use syntax checker (#USYX) to read and check control statements. Save keyword information if verb is PATCH. If END statement is read, go to **6** to terminate program.	$FEPCH
2 Initialize display screen, diskette, and disk fields as required.	$FEDSK
3 Prompt for disk/diskette sector to be patched. Ensure valid prompt response given. If character E response, go to **1** to read next control statement. Read data from disk or diskette.	
4 Display disk/diskette sector requested: ● Put data to display screen. ● Roll screen up or down as requested. ● If character E entered, go to **3** to prompt for new sector address. ● Update data area with modified data.	$FECRT
5 Write modified data back to disk/diskette. Return to **3** to prompt for new sector.	$FEDSK
6 Pass control to end-of-job control storage transient ($EOJ) to terminate program.	$FEPCH

Diagram 7.4. Perform Patch Utility Function ($FEPCH)


From: Initiator
INPUT
Main Storage
Transient Area
User Area:
$FETRC
Control store trace transient parameter list
PROCESS
1 Get current trace indicators and prompt for modifications
2 Update trace indicators
3 Prompt for disk logging options
4 Update disk logging indicators
5 Terminate program
OUTPUT
Updated control store trace options
To: Control Storage End-of-Job Transient ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Call control store trace transient to determine events now being traced. Display current trace options and prompt for new options.	$FETRC
2 Scan CRT buffer for new trace options just entered. Update trace control store transient parameter list with new options. Call control store trace transient to set new events to be traced.	
3 Prompt for: ● Start logging to disk . ● Stop logging to disk. ● No change.	
4 If STOP option, set stop logging indicator. IfSTART option: ● If old trace file exists, destroy it. ● Create new trace file with specified size. ● Obtain area in nucleus for disk logging IOB. Call control store trace transient to set new disk logging options for START or STOP request.	
5 Pass control to end-of-job control storage transient ($EOJ) to terminate program.	

Diagram 7.5. Perform Trace Select Function ($FETRC)



From: Initiator

INPUT

Main Storage

Transient Area

User Area:
$ERAP
$ERA0
$ERC0
$ERCA
$ERD0
$ERE0
$ER01
$ER02
$ER80

SCA
I/O counter tables
Error counter tables
Error history tables

PROCESS

1 Prompt for device

2 Prompt for desired output

3 Format I/O counter table, error counter table, and error history table for specified device

4 Reset error counter table or I/O counter table if requested

5 Terminate program

OUTPUT

Displayed prompts

Formatted tables on display station or printer

To: Control Storage End-of-Job Transient ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Issue prompt for device or end program: ● End. ● All. ● Main store processor. ● Control processor. ● Disk. ● Diskette. ● Line printer. ● Work station controller. ● Display stations. ● Serial matrix printer. ● BSC. If more than one device of a certain type, issue prompt for that specific device. If end option taken, go to **5**. **2** Issue prompt for desired output: ● Display. ● Print. ● Print and reset error counter table. ● Print and reset I/O counter table.	$ERAP

Diagram 7.6 (Part 1 of 2). Perform ERAP Utility Function

DESCRIPTION	MODULE/ ROUTINE
If print selected, prompt for desired printer. If display or print, go to 3. If reset, go to 4 .	$ERAP
3 Format I/O counter table, error counter table, and error history table for specified device. If C entered, go to 1. 4 Format error counter table and reset, or format I/O counter table and reset. Go to 1.	$ERA0 $ERC0 $ERCA $ERD0 $ERE0 $ER01 $ER02 $ER80
5 Pass control to control storage end-of-job transient ($EOJ — control storage).	$ERAP

Diagram 7.6 (Part 2 of 2). Perform ERAP Utility Function

Program Organization

Figures 7-1 through 7-6 show the control flow of the system maintenance programs.

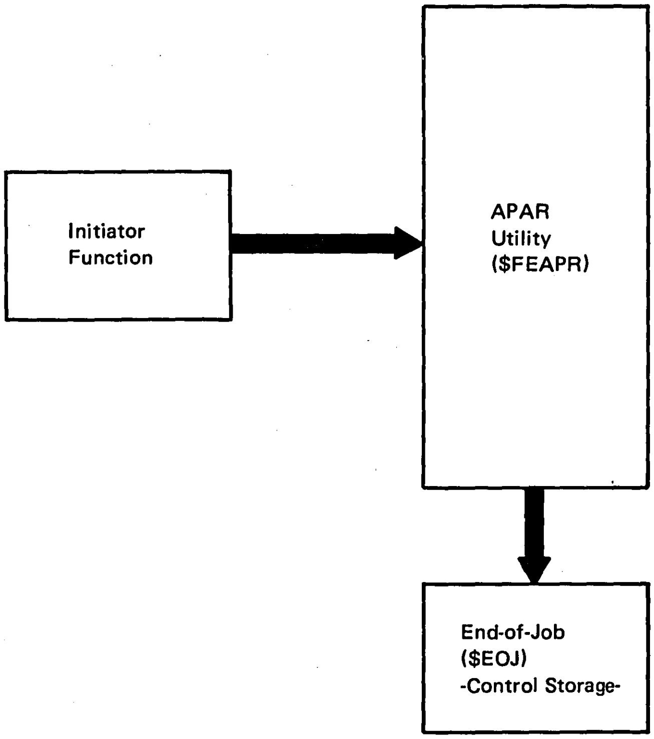


Figure 7-1. APAR Utility Control Flow ($FEAPR)

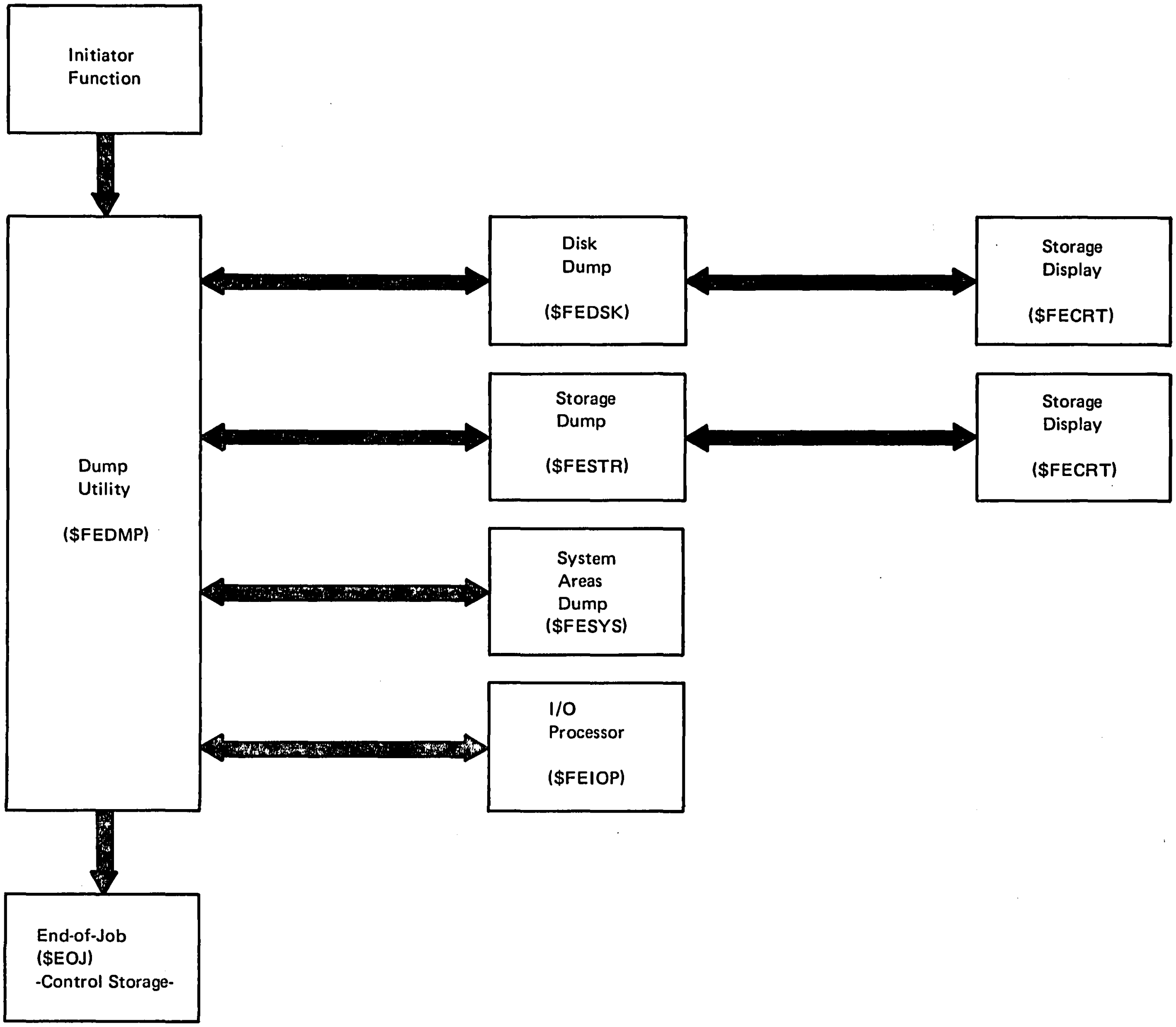


Figure 7-2. Dump Utility Control Flow ($FEDMP)

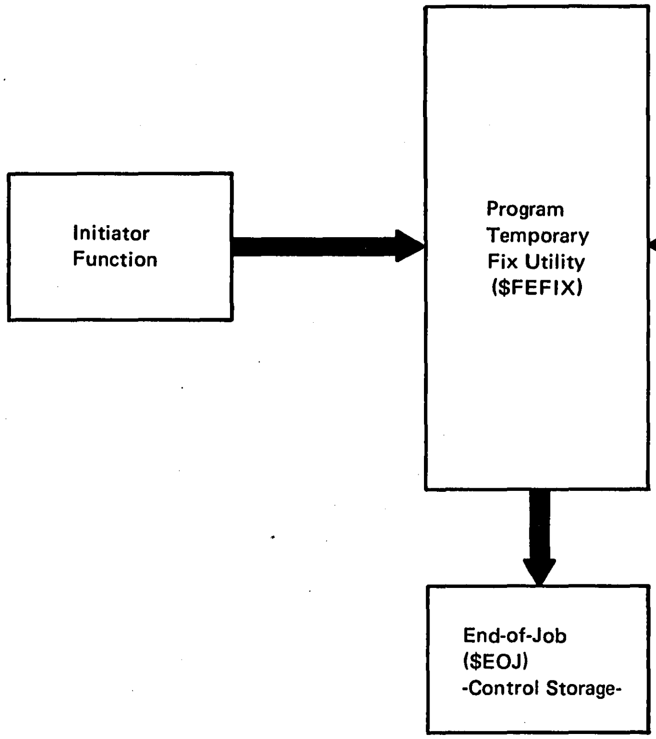


Figure 7-3. Program Temporary Fix Utility Control Flow ($FEFIX)

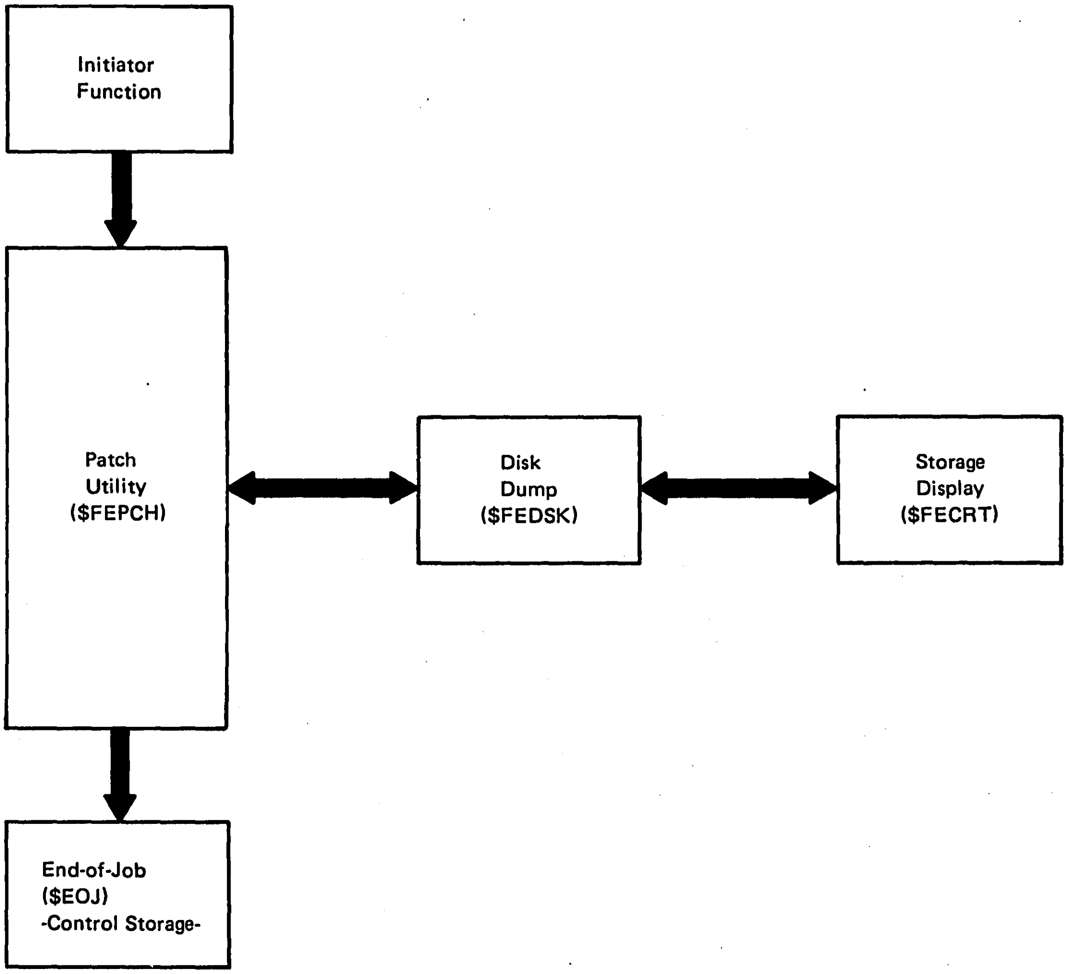


Figure 7-4. Patch Utility Control Flow ($FEPCH)

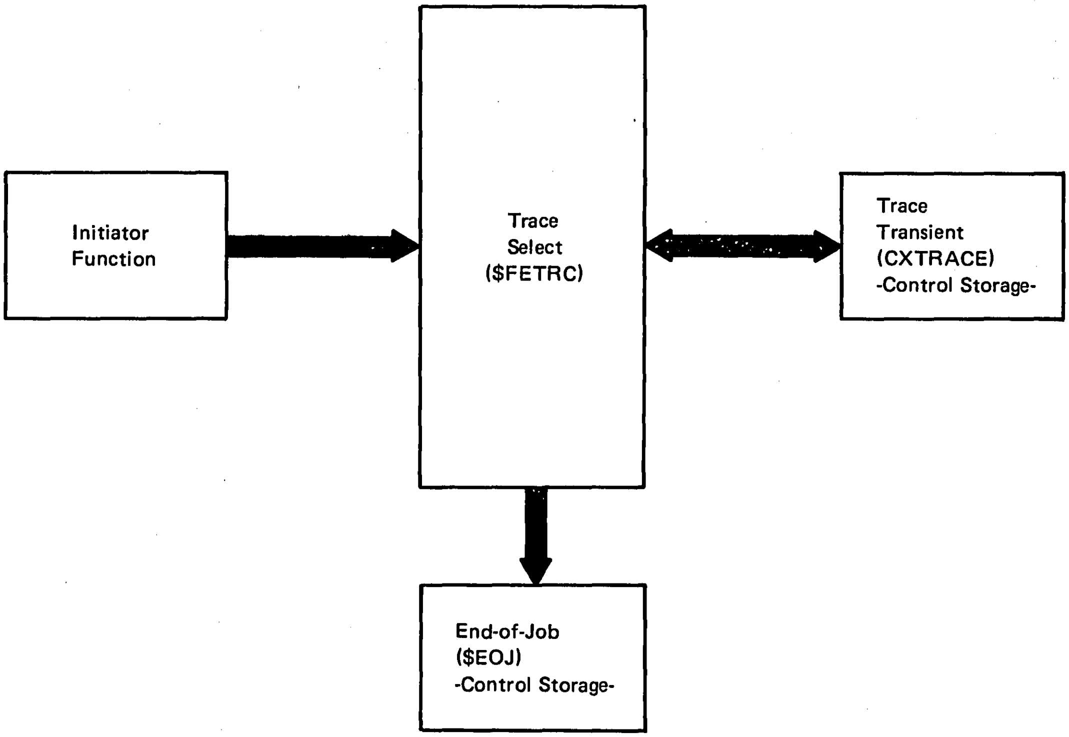


Figure 7-5. Trace Select Control Flow ($FETRC)

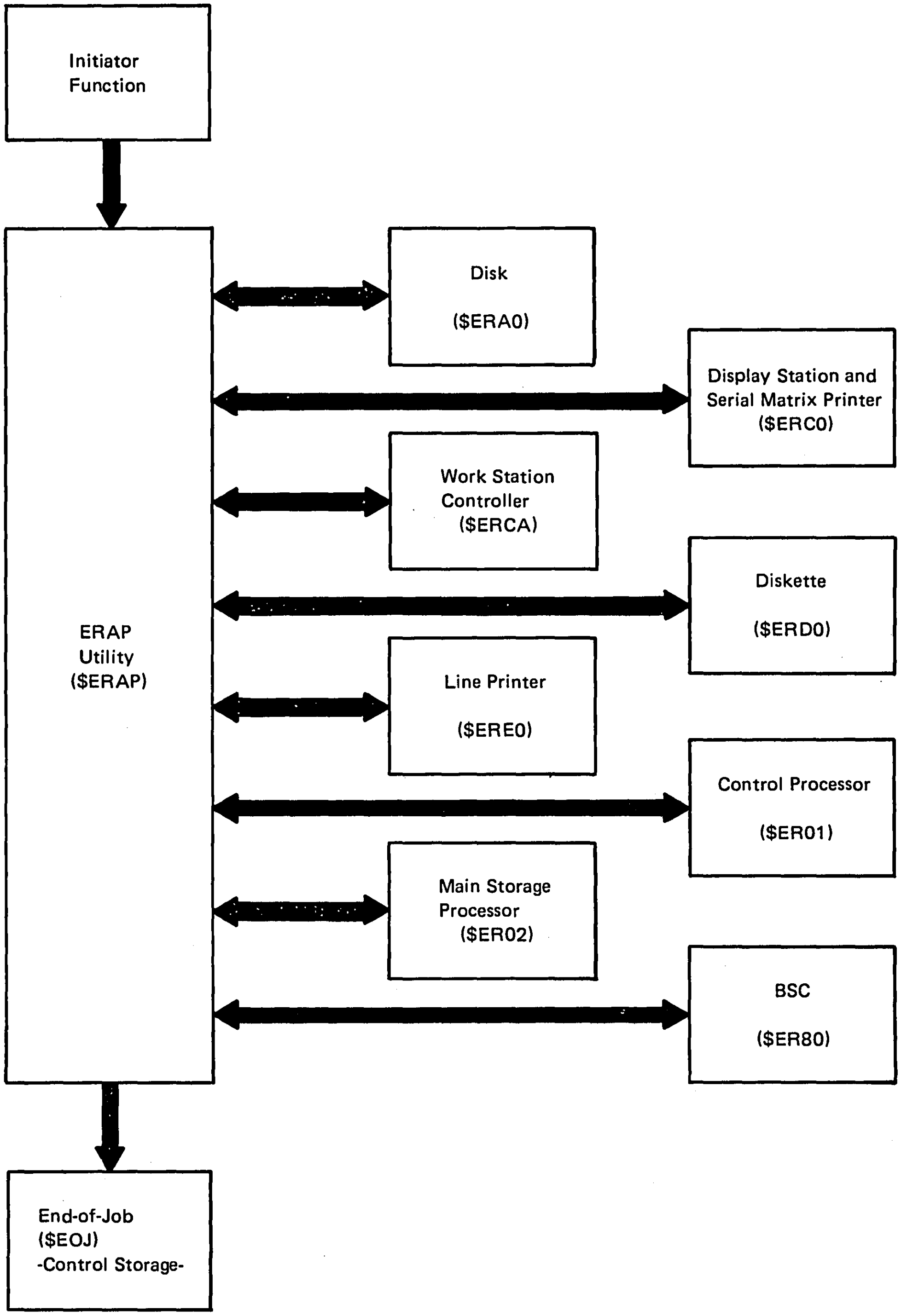

Initiator Function
Disk ($ERA0)
Display Station and Serial Matrix Printer ($ERC0)
Work Station Controller ($ERCA)
Diskette ($ERD0)
ERAP Utility ($ERAP)
Line Printer ($ERE0)
Control Processor ($ER01)
Main Storage Processor ($ER02)
BSC ($ER80)
End-of-Job ($EOJ) -Control Storage-

Figure 7-6. ERAP Utility Control Flow ($ERAP)

Introduction

The overlay linkage editor enables the user to manually determine overlays for programs. An automatic determination of overlays is also provided.

The overlay linkage editor can be entered two ways: directly from a language processor (compiler), or as a user-called program. The functions and method of operation is different depending on whether the entry is compiler or user entry.

COMPILER ENTRY

When entered directly from a compiler, the overlay linkage editor can perform either or both of the following functions:

- Catalog an object module as a subroutine member in the library on disk.
- Link object modules into an object program and catalog the program as a load member in the library on disk.

INPUT FOR COMPILER ENTRY

Input to the overlay linkage editor is in the $WORK file on disk. Each record in $WORK is 64 bytes long (Figure 8-1). The first record must be the options record; object modules follow the options record.

Each object module consists of external symbol list (ESL) fields (packed five to a 64-byte, S-type record) and text records. An END record follows the object modules. A /* record must be the last record in the compiler output.

Options Record: The options record tells the overlay linkage editor what functions to perform. The options record must be the first record in $WORK. Figure 8-1 shows the format of the options record.

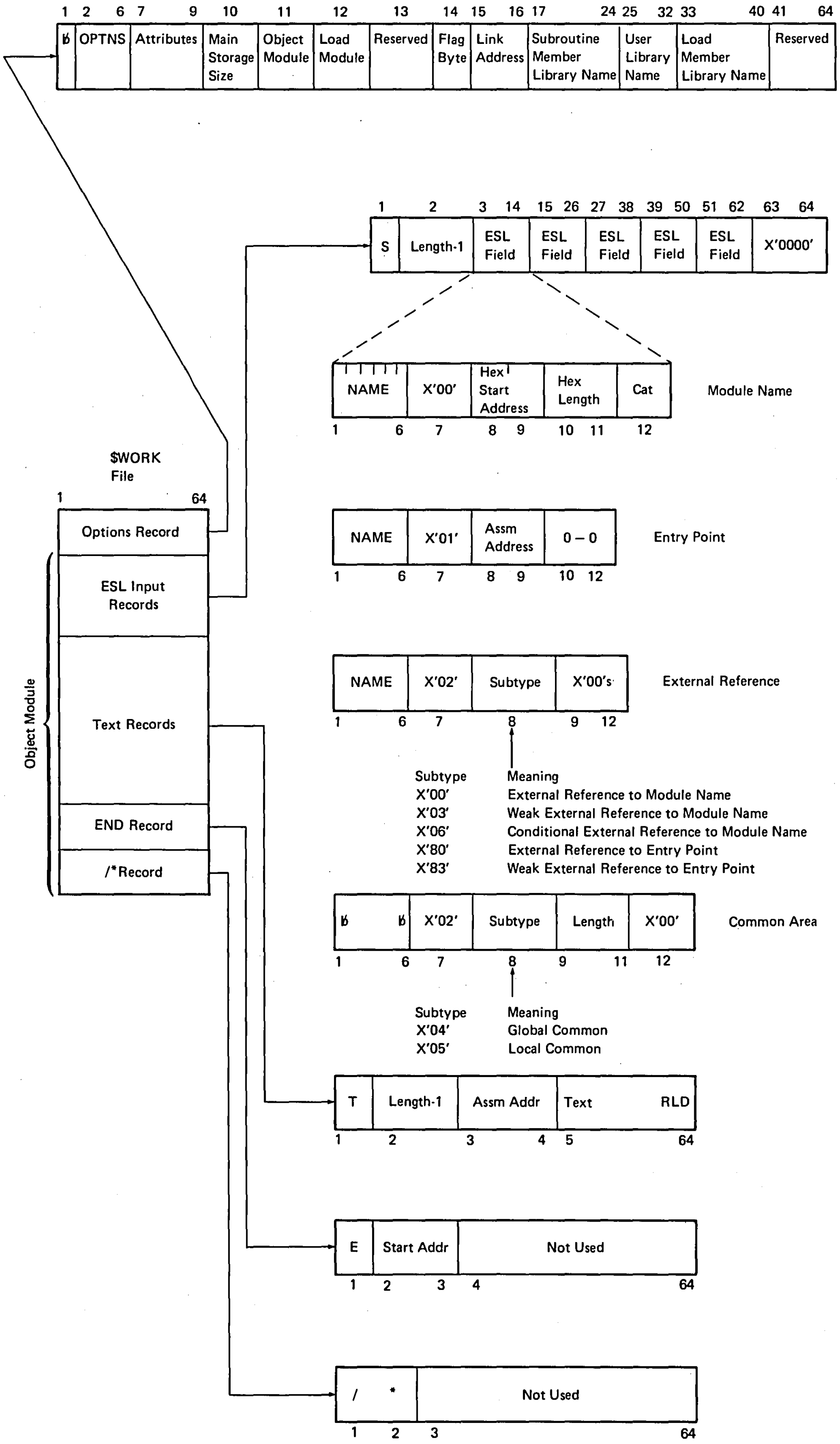
Object module: The object module consists of ESL fields packed into S-type records, text records, and an END record. Each 64-byte, S-type record can contain up to five 12-byte ESL fields. The S-type record must be X'0000' after the ESL fields.

Object modules are described in the *System/34 Overlay Linkage Editor Reference Manual,* SC21-7707.

OUTPUT FROM COMPILER ENTRY

Output from the overlay linkage editor is specified by the options record in $WORK. The object module in $WORK can be cataloged into the library as a subroutine member. If link-editing is specified, a load module is built from the input object module. The load module is then cataloged into the library as a load member.

A storage map and cross-reference list is printed unless the options record specifies otherwise (Figure 8-25).



1	2 6	7 9	10	11	12	13	14	15 16	17 24	25 32	33 40	41 64
ƀ	OPTNS	Attributes	Main Storage Size	Object Module	Load Module	Reserved	Flag Byte	Link Address	Subroutine Member Library Name	User Library Name	Load Member Library Name	Reserved

1	2	3 14	15 26	27 38	39 50	51 62	63 64
S	Length-1	ESL Field	ESL Field	ESL Field	ESL Field	ESL Field	X'0000'

Module Name

NAME	X'00'	Hex Start Address	Hex Length	Cat
1 6	7	8 9	10 11	12

Entry Point

NAME	X'01'	Assm Address	0 – 0
1 6	7	8 9	10 12

External Reference

NAME	X'02'	Subtype	X'00's
1 6	7	8	9 12

Subtype	Meaning
X'00'	External Reference to Module Name
X'03'	Weak External Reference to Module Name
X'06'	Conditional External Reference to Module Name
X'80'	External Reference to Entry Point
X'83'	Weak External Reference to Entry Point

Common Area

ƀ ƀ	X'02'	Subtype	Length	X'00'
1 6	7	8	9 11	12

Subtype	Meaning
X'04'	Global Common
X'05'	Local Common

T	Length-1	Assm Addr	Text RLD
1	2	3 4	5 64

E	Start Addr	Not Used
1	2 3	4 64

/	*	Not Used
1	2	3 64

Figure 8-1. Input For Compiler Entry

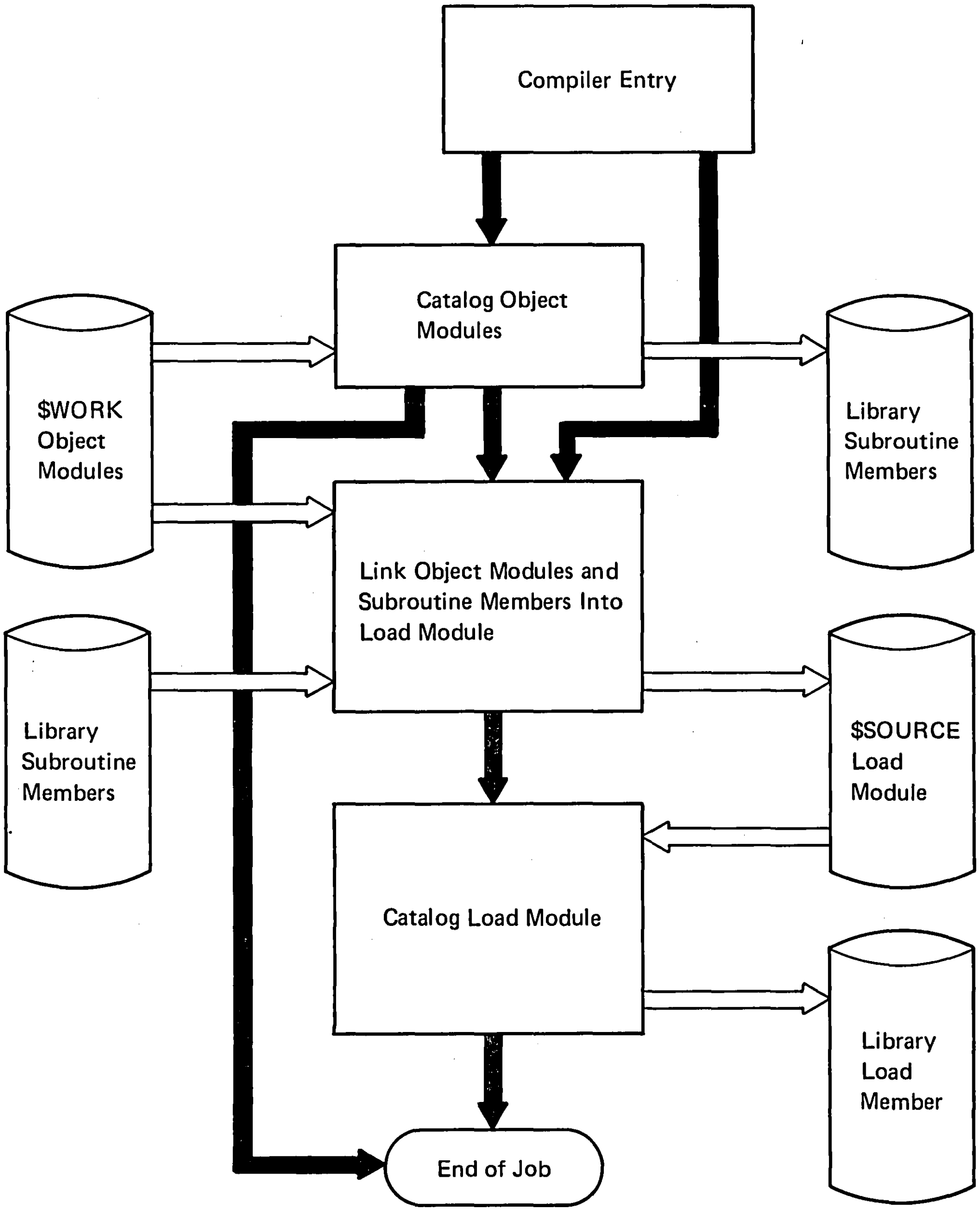


Figure 8-2. Overview of Overlay Linkage Editor Compiler Entry

1	2 6	7 9	10	11	12
b̸	OTPNS	Attributes	Main Storage Size	Object Module	Load Module

13	14	15 16	17 24	25 32
Reserved	Flag Byte	Link Address	Subroutine Member Library Name	User Library Name

33 40	41 64
Load Member Library Name	Reserved

Attributes: This 3 byte field describes the linked object program built by the overlay linkage editor.

Byte 7
Bit 0 On – SSP module; Off – not SSP module
1 Privileged module (O-only), proc with data (P-only)
2 Not inquirable mode
3 SFGR format load member
4 Source required
5 Not base SSP module
6 PTF applied (cannot be assigned through overlay linkage editor)
7 Module has overlays

Byte 8
Bit 0 Dedicated mode
1 NEP module
2 Module has OXRF format index table
3 Load module only from system console
4 Cannot load program with a LOAD statement
5 Program common
6 Program with utility control statements
7 Module has OXRF where-to-go table

Byte 9
Bit 0 $WORK2 file required
1 Do not swap this task
2 High level of dedication
3 Reserved
4 Reserved
5 Reserved
6 Reserved
7 Reserved

Figure 8-3 (Part 1 of 3). Options Record

Main storage size: This is the amount of main storage (in 1/4K increments) needed for object program execution. Example: X'12' = 18 (X'12') X 256 (1/4) = 4608 bytes

Object module: This byte specifies the disposition of the object module in $WORK.
Bit 0 Reserved
1 Reserved
2 Reserved
3 Reserved
4 Reserved
5 Catalog as subroutine member (RETAIN – R) in library
6 Catalog as subroutine member (permanent entry) in library
7 Reserved
If bits 0 – 7 are all zero, there is no object module

Load module: This byte specifies the disposition of the linked object program and the type of printed output from the overlay linkage editor.
Bit 0 Reserved
1 Reserved
2 Reserved
3 Reserved
4 Reserved
5 Do not print storage map
6 Do not print cross-reference list
7 Catalog program into object library
If bits 0 – 7 are all zero, there is no linked output. If neither object nor load module is specified, load is cataloged.

Flag Byte: This byte passes general information to the overlay linkage editor.
Bit 0 Reserved
1 Reserved
2 Reserved
3 Generate RLDs
4 Print messages
5 Reserved
6 Catalog as a load member (RETAIN – R) in library
7 Link edit address in byte 15 and 16

Link Address: These 2 bytes specify a link-edit address. If bit 7 of the flag byte is not on, the overlay linkage editor links the load module to address X'0000'.

Figure 8-3 (Part 2 of 3). Options Record

Subroutine Member Library Name: If an object module is requested (see bits 5 and 6 of the object module field), it is cataloged as a subroutine member in this library. If this field is blank, the default is CIBOTLB (compiler information block output library – from the COMPILE OUTLIB OCL statement).

User Library Name: If a load module is requested (see Load module and Flag Byte fields), the user subroutine members used to create the load module are found in this library. A subroutine name beginning with a # causes the overlay linkage editor to search #LIBRARY . If the field is blank, the default is CIBOTLB (compiler information block output library – from the COMPILE OUTLIB OCL statement).

Load Member Library Name: If a load member is requested (see Load module and Flag Byte fields), it is cataloged as a load member in this library. If this field is blank, the default is CIBOTLB (compiler information block output library – from the COMPILE OUTLIB OCL statement).

Figure 8-3 (Part 3 of 3). Options Record

USER ENTRY

The overlay linkage editor can be loaded by using a LOAD #OLINK OCL statement. The user must supply control statements.

INPUT FOR USER ENTRY

Input for the user entry is described in the *Overlay Linkage Editor Reference Manual.*

OUTPUT FROM USER ENTRY

Output of the overlay linkage editor for user entry is an object program cataloged as a load member in the library. A storage map and cross-reference list are printed depending on the MAP parameter of the OPTIONS statement.

OVERLAY LINKAGE EDITOR PHASES AND ROUTINES

The overlay linkage editor consists of the following phases and routines:

- Compiler entry phase (#OLYNX)
- User entry phase 1 (#OLINK)
- User entry phase 2 (#OLI1)
- User entry phase 3 (#OLI2)
- User entry phase 4 (#OLI3)
- Autolink segment list build (#OLAF)
- Cross-reference segment list build (#OLAH)
- Sort autolink segment list (#OLAJ)
- Overlay design (#OLAP)
- Overlay segment list build (#OLAR)
- Storage map phase (#OLAT)
- Relocate, resolve EXTRNs, and build load module phase (#OLBE)
- Library control phase (#OLBO)
- Specification module for the syntax checker (#OLISP)
- Error message print phase (#OLMSG)
- Error routine (#OLER)

Minimum Storage Requirements

The overlay linkage editor requires 14K of main storage for execution.

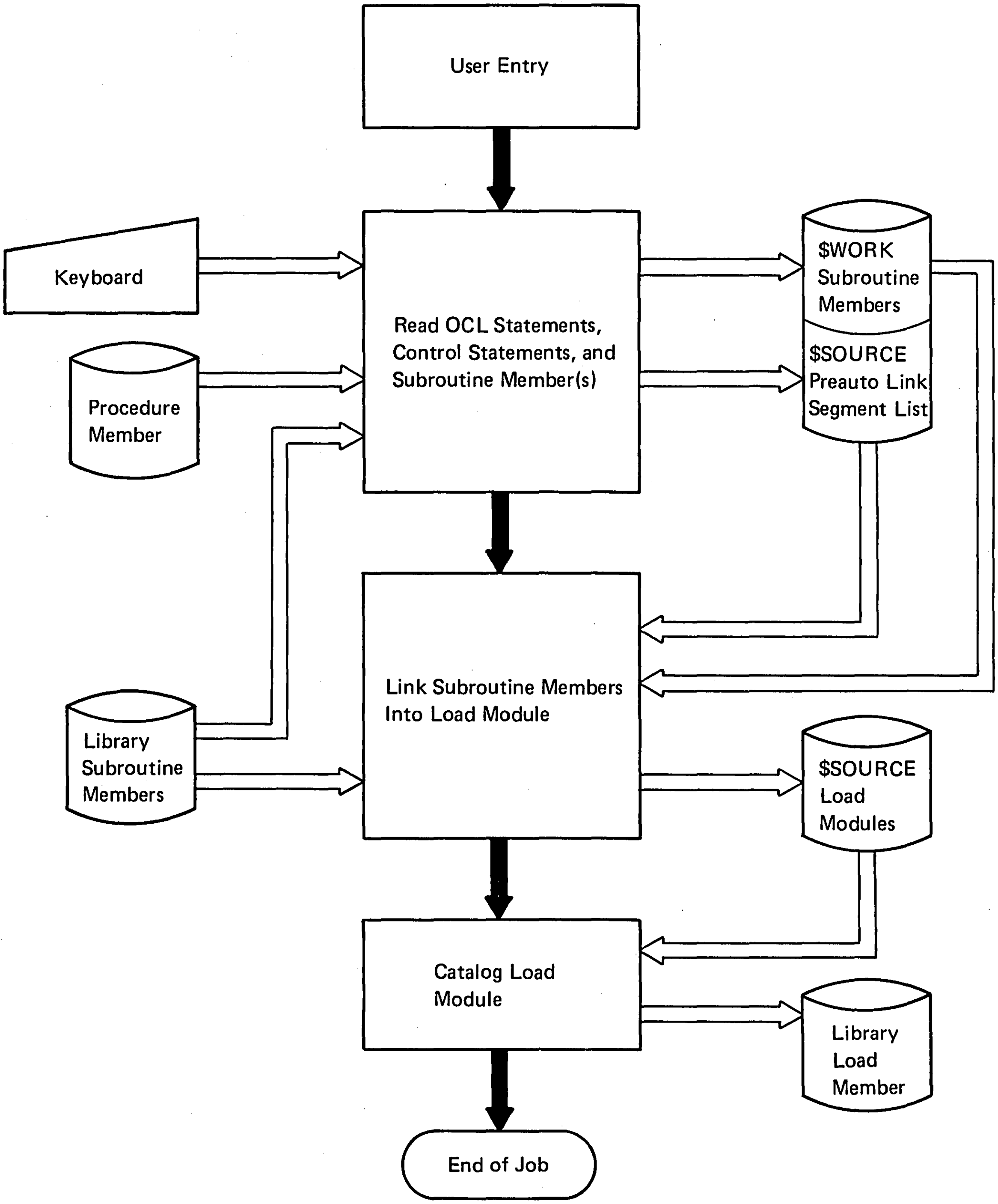

User Entry
Keyboard
Procedure Member
Read OCL Statements, Control Statements, and Subroutine Member(s)
$WORK Subroutine Members
$SOURCE Preauto Link Segment List
Library Subroutine Members
Link Subroutine Members Into Load Module
$SOURCE Load Modules
Catalog Load Module
Library Load Member
End of Job

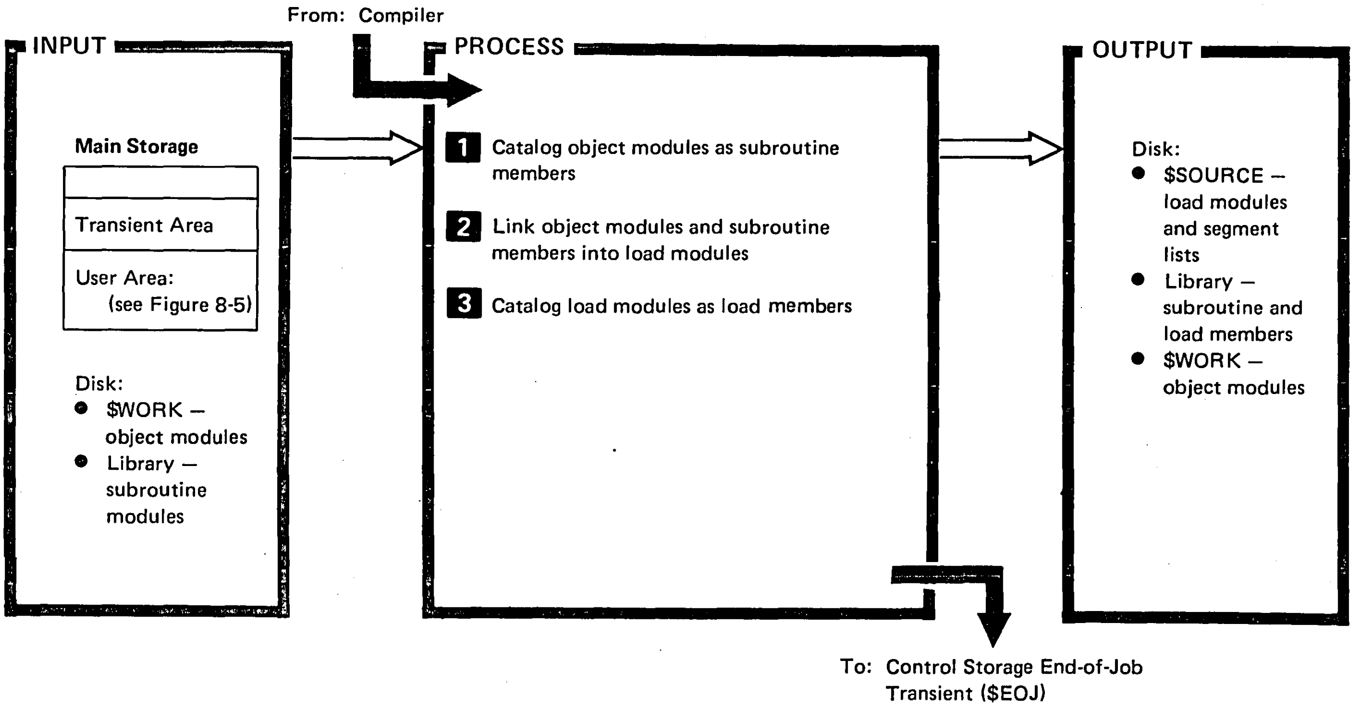
Figure 8-4. Overview of Overlay Linkage Editor User Entry

Method of Operation

The overlay linkage editor can be entered in two ways:

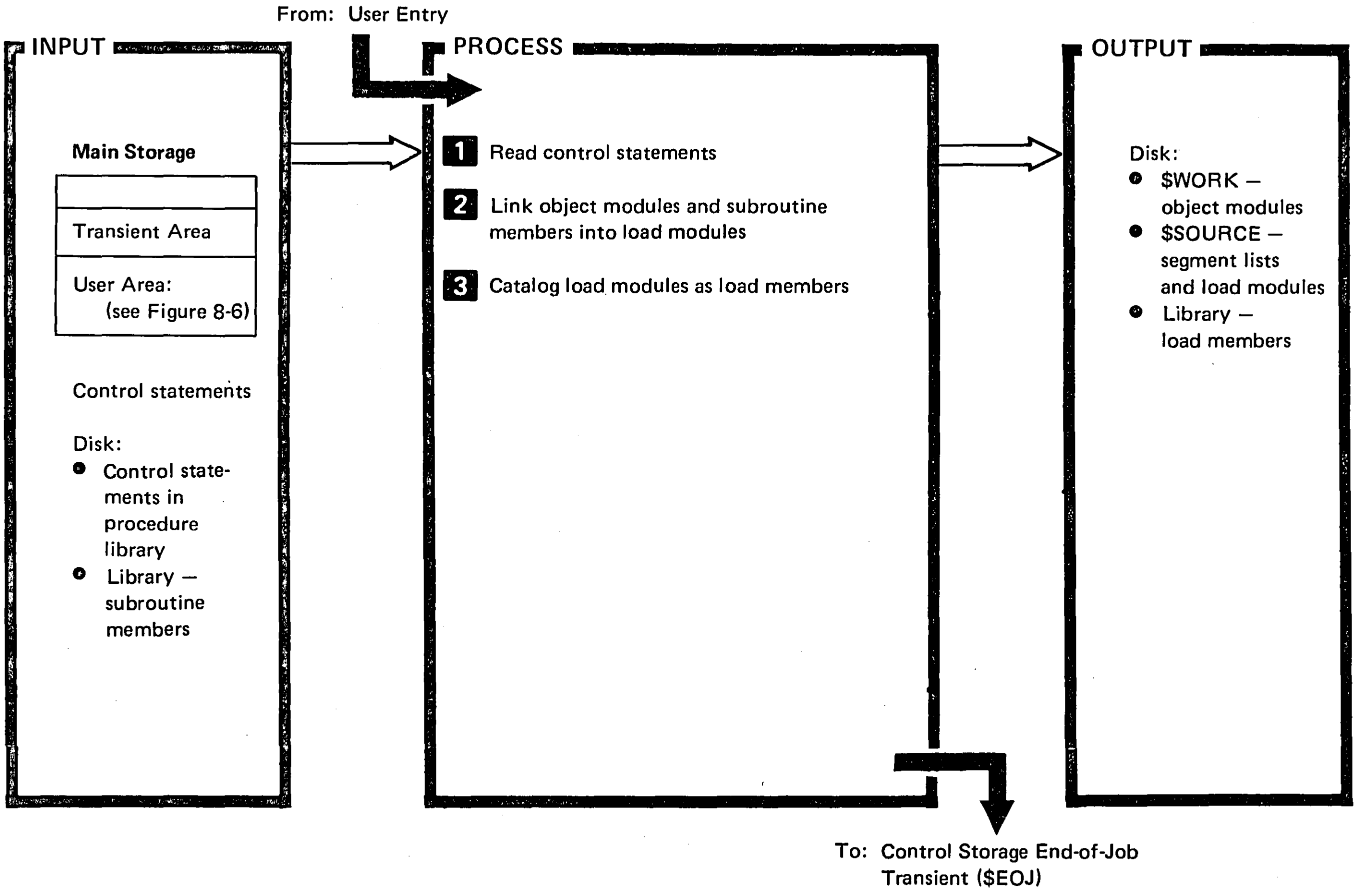
- Diagram 8.1 shows an overview of the compiler entry.
- Diagram 8.2 shows an overview of the user entry.

Diagrams 8.3 through 8.17 show the functions of the overlay linkage editor.



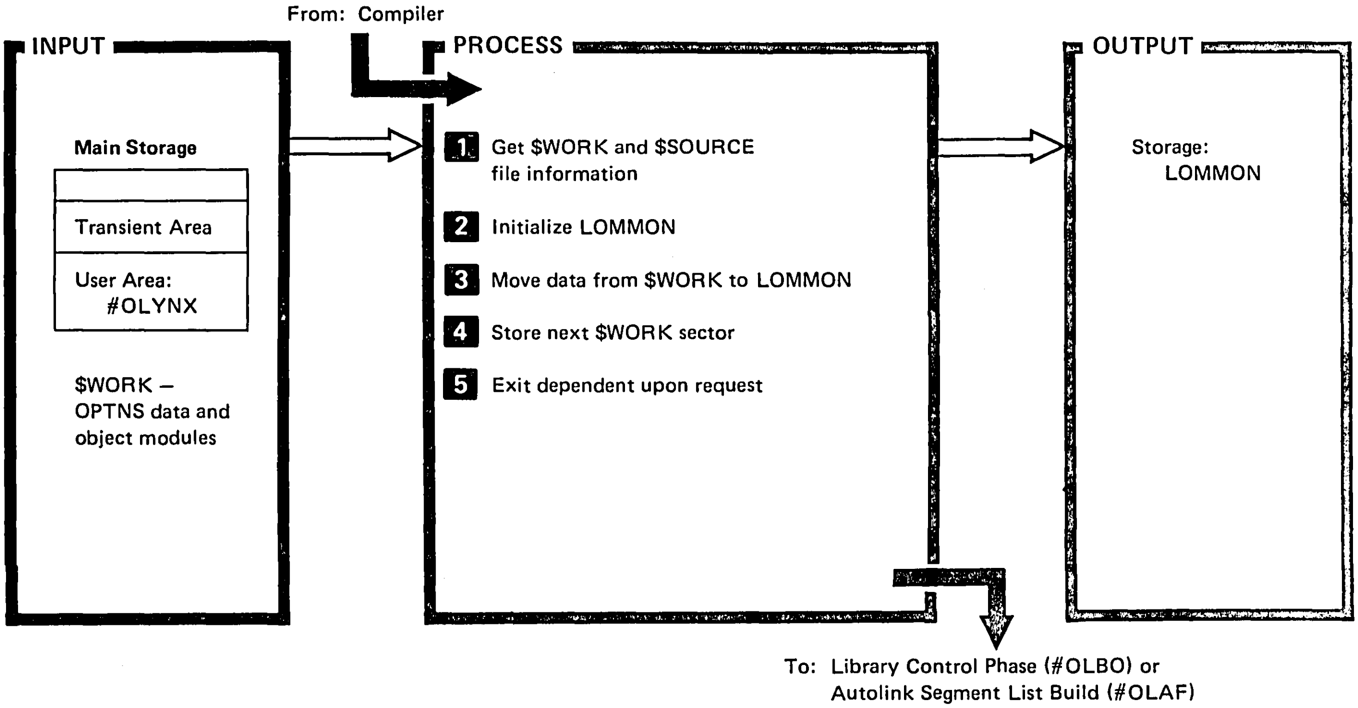
From: Compiler
INPUT
Main Storage
Transient Area
User Area:
(see Figure 8-5)
Disk:
● $WORK – object modules
● Library – subroutine modules
PROCESS
1 Catalog object modules as subroutine members
2 Link object modules and subroutine members into load modules
3 Catalog load modules as load members
OUTPUT
Disk:
● $SOURCE – load modules and segment lists
● Library – subroutine and load members
● $WORK – object modules
To: Control Storage End-of-Job Transient ($EOJ)

Diagram 8.1. Overview of Overlay Linkage Editor Compiler Entry



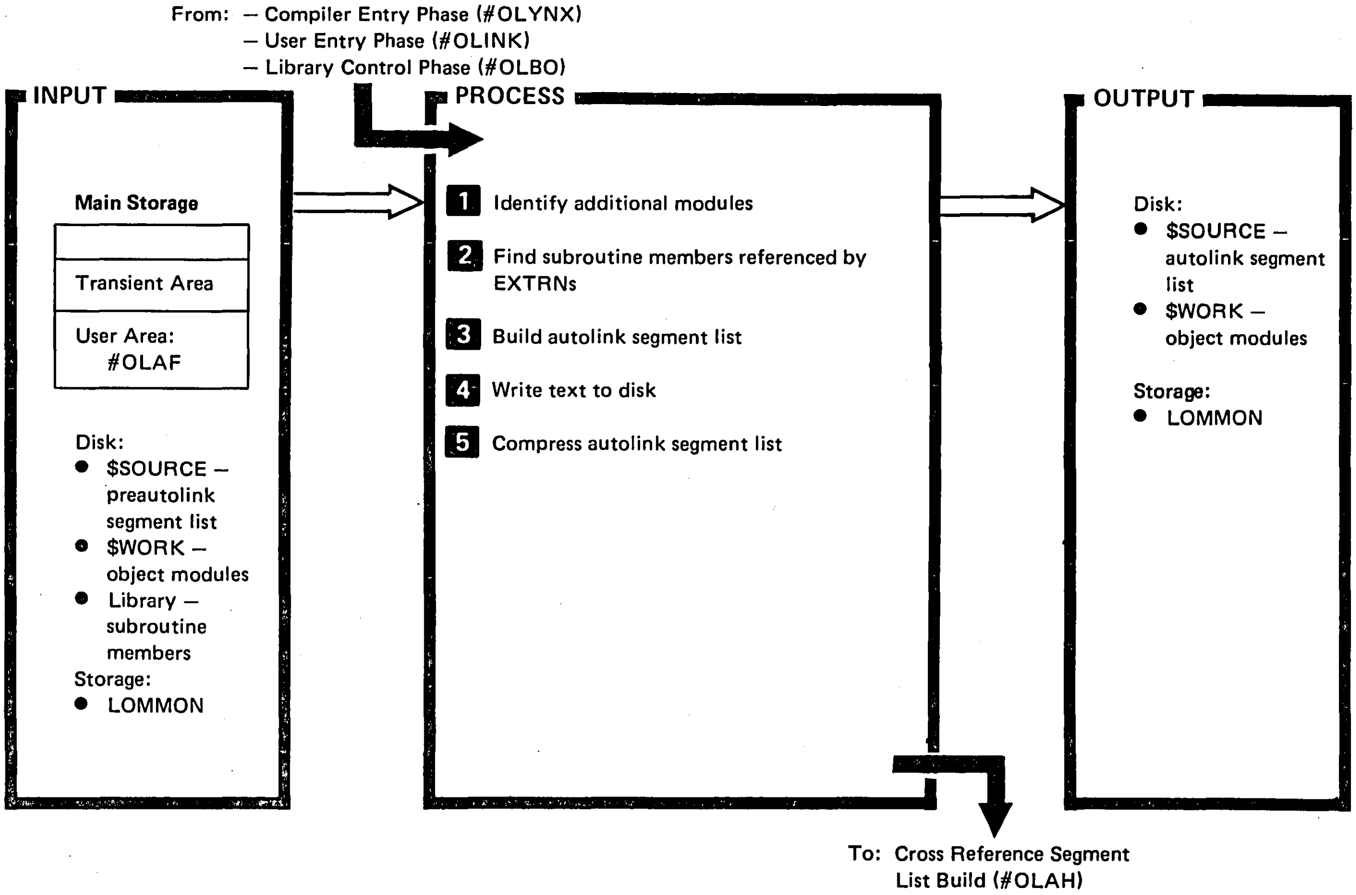
From: User Entry
INPUT
Main Storage
Transient Area
User Area:
(see Figure 8-6)
Control statements
Disk:
● Control statements in procedure library
● Library – subroutine members
PROCESS
1 Read control statements
2 Link object modules and subroutine members into load modules
3 Catalog load modules as load members
OUTPUT
Disk:
● $WORK – object modules
● $SOURCE – segment lists and load modules
● Library – load members
To: Control Storage End-of-Job Transient ($EOJ)

Diagram 8.2. Overview of Overlay Linkage Editor User Entry


From: Compiler
INPUT
Main Storage
Transient Area
User Area:
#OLYNX
$WORK –
OPTNS data and
object modules
PROCESS
1 Get $WORK and $SOURCE file information
2 Initialize LOMMON
3 Move data from $WORK to LOMMON
4 Store next $WORK sector
5 Exit dependent upon request
OUTPUT
Storage:
LOMMON
To: Library Control Phase (#OLBO) or
Autolink Segment List Build (#OLAF)

DESCRIPTION	MODULE/ ROUTINE
1 Get $WORK and $SOURCE file information from compiler information block.	#OLYNX
2 Initialize 800-byte work area used by OLE routines (LOMMON).	
3 Read OPTNS and name ESL records.	
Put records in LOMMON.	
4 Put relative number of next available $WORK sector in LOMMON.	
5 If object module to be cataloged as subroutine member, go to library control phase (#OLBO).	
If not, go to autolink segment list build (#OLAF).	
Issue error messages as needed.	#OLER

Diagram 8.3. Compiler Entry Phase (#OLYNX)



From: – Compiler Entry Phase (#OLYNX)
– User Entry Phase (#OLINK)
– Library Control Phase (#OLBO)

INPUT

Main Storage

Transient Area

User Area:
#OLAF

Disk:
- $SOURCE – preautolink segment list
- $WORK – object modules
- Library – subroutine members

Storage:
- LOMMON

PROCESS

1 Identify additional modules

2 Find subroutine members referenced by EXTRNs

3 Build autolink segment list

4 Write text to disk

5 Compress autolink segment list

OUTPUT

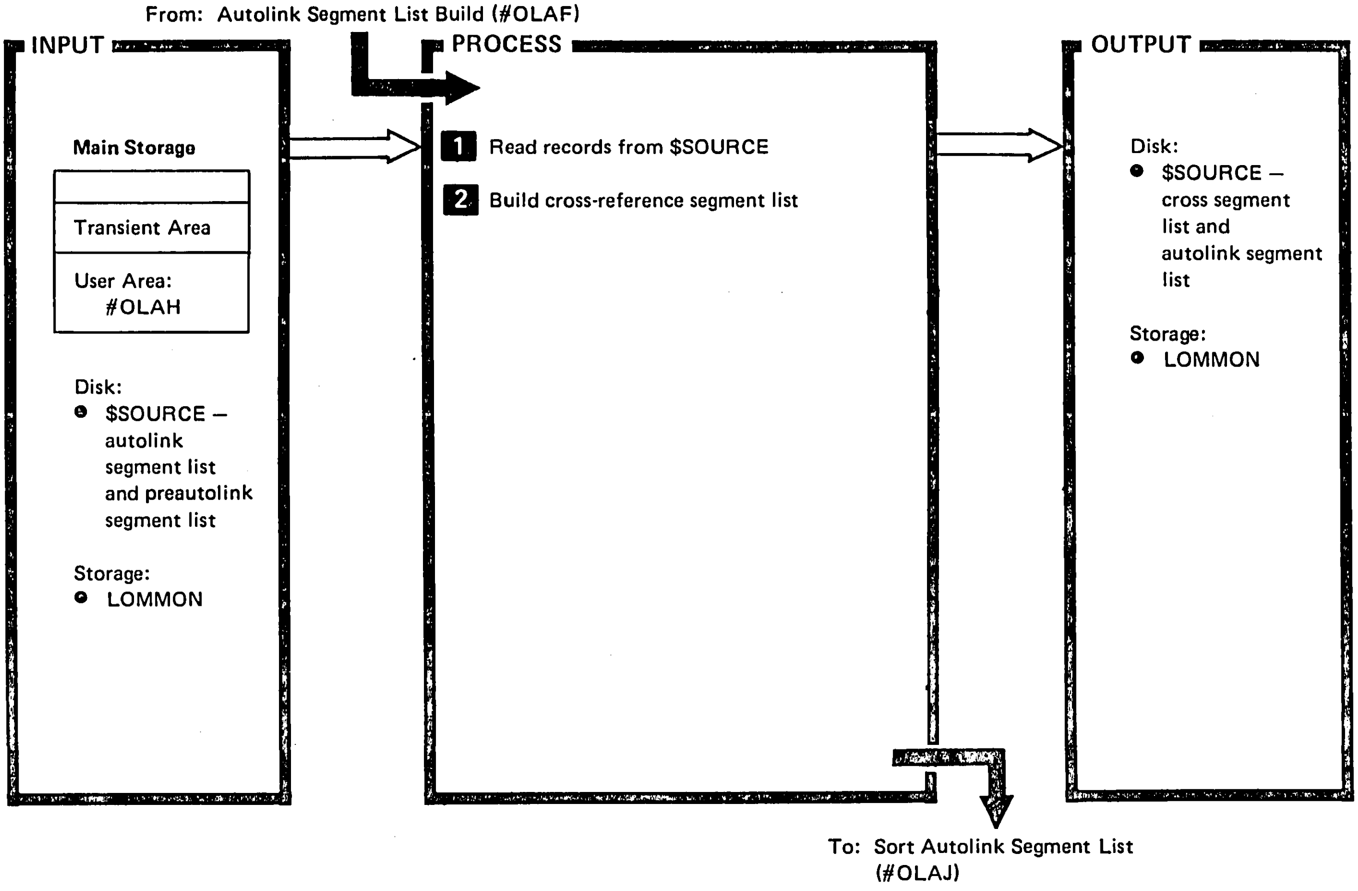
Disk:
- $SOURCE – autolink segment list
- $WORK – object modules

Storage:
- LOMMON

To: Cross Reference Segment
List Build (#OLAH)

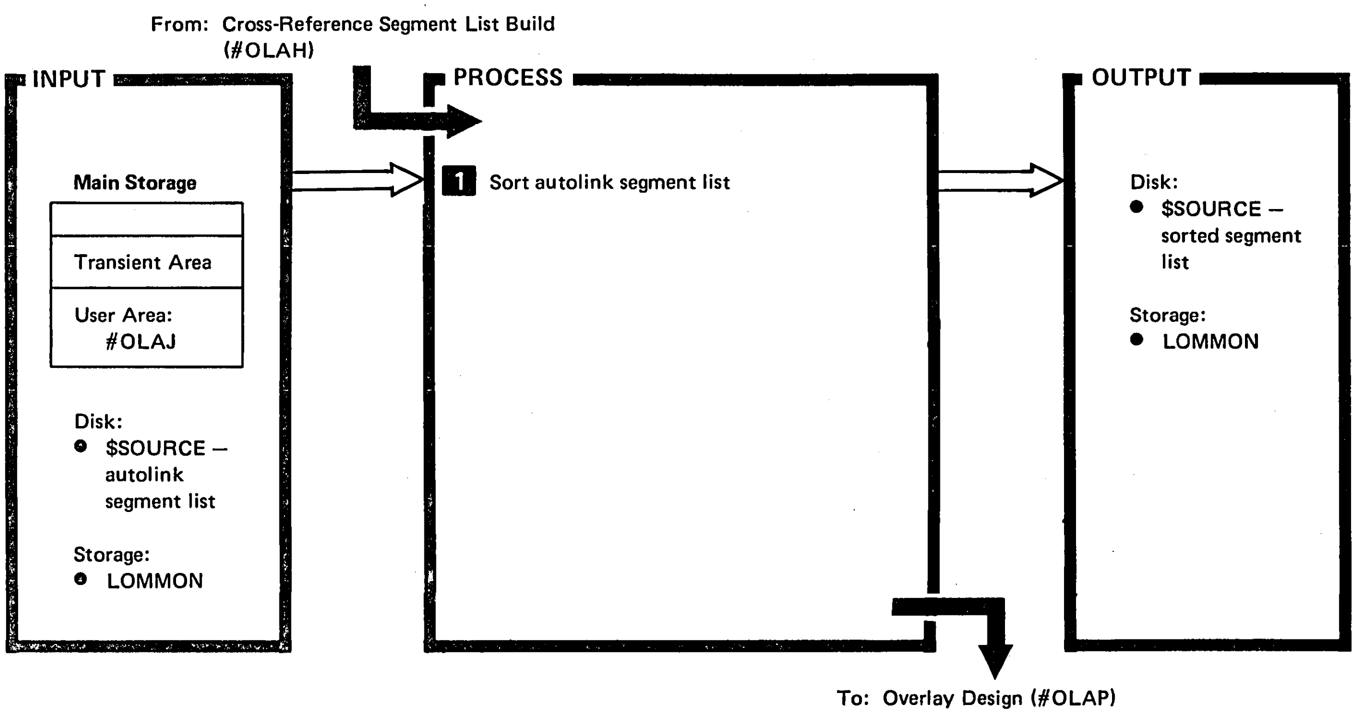
DESCRIPTION	MODULE/ ROUTINE
1 Read external segment list for each object module in $WORK to determine if any additional EXTRNs required.	#OLAF
2 Find additional modules (subroutine members) referenced by EXTRNs.	$MALFN
Put subroutine members in $WORK on disk. 3 Scan modules for EXTRNs. Assign EXTRN numbers to segments found (duplicate segments are given same number). Place all EXTRN segment elements in autolink segment list. 4 Write object module text records back to $WORK as they are encountered. Place an E in first byte of last text record of each module. 5 Delete all duplicate EXTRN numbers. Delete all but one COMMON entry from autolink segment list.	#OLAF
Issue error messages as needed: ● If message to be printed, use (#OLMSG).	#OLMSG
● If message to be displayed, use (#OLER).	#OLER

Diagram 8.4. Autolink Segment List Build (#OLAF)


From: Autolink Segment List Build (#OLAF)
INPUT
Main Storage
Transient Area
User Area: #OLAH
Disk:
● $SOURCE – autolink segment list and preautolink segment list
Storage:
● LOMMON
PROCESS
1 Read records from $SOURCE
2 Build cross-reference segment list
OUTPUT
Disk:
● $SOURCE – cross segment list and autolink segment list
Storage:
● LOMMON
To: Sort Autolink Segment List (#OLAJ)

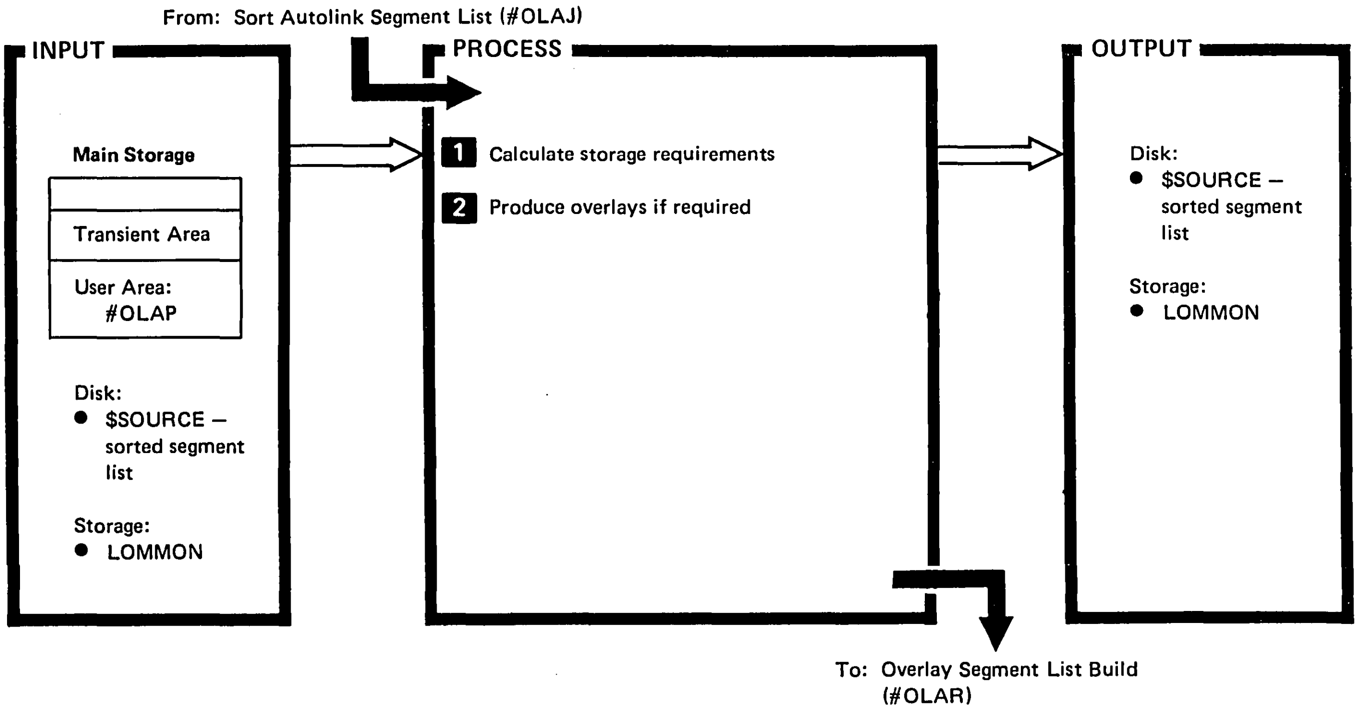
DESCRIPTION	MODULE/ ROUTINE
1 Read records from autolink segment list in $SOURCE.	#OLAH
2 Build cross-reference segment list containing module name, followed by an entry point, followed by modules referencing entry point. Scan autolink segment list for modules with EXTRNs. Add EXTRNs to cross-reference segment list after either module name or entry point.	
Issue error messages as needed: ● If message is to be displayed, use #OLER.	#OLER
● If message is to be printed, use #OLMSG.	#OLMSG

Diagram 8.5. Cross-Reference Segment List Build (#OLAH)


From: Cross-Reference Segment List Build (#OLAH)
INPUT
Main Storage
Transient Area
User Area: #OLAJ
Disk:
● $SOURCE – autolink segment list
Storage:
● LOMMON
PROCESS
1 Sort autolink segment list
OUTPUT
Disk:
● $SOURCE – sorted segment list
Storage:
● LOMMON
To: Overlay Design (#OLAP)

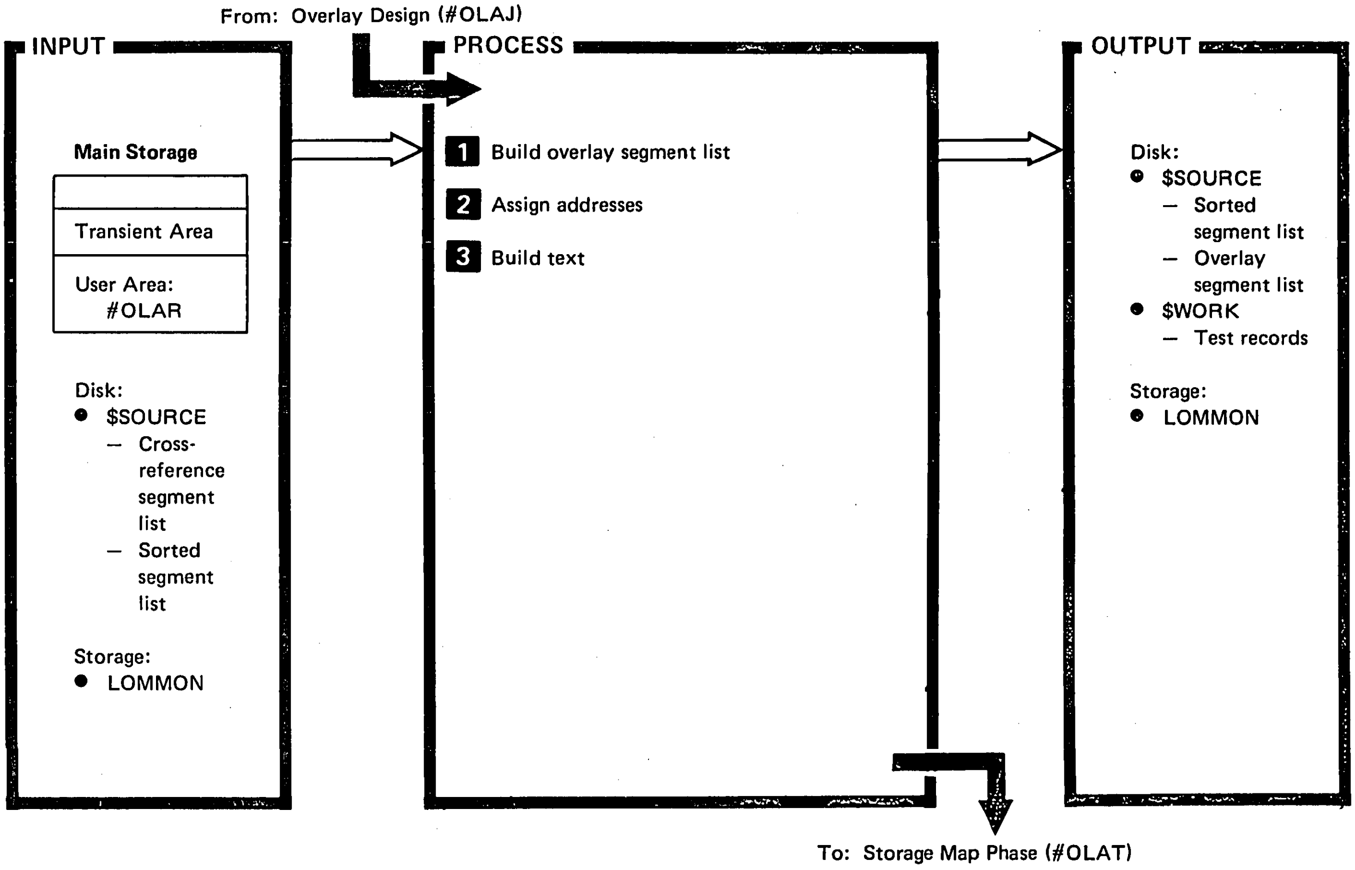
DESCRIPTION	MODULE/ ROUTINE
1 Group autolink segment list elements into sublists by: ● Common area (uses only largest common area on autolink segment list). ● Root mainline. ● Zero priority elements. ● Elements used by zero priority elements. ● Category. ● User overlay.	#OLAJ
Issue error messages as needed.	#OLER

Diagram 8.6. Sort Autolink Segment List (#OLAJ)


From: Sort Autolink Segment List (#OLAJ)
INPUT
Main Storage
Transient Area
User Area: #OLAP
Disk:
● $SOURCE – sorted segment list
Storage:
● LOMMON
PROCESS
1 Calculate storage requirements
2 Produce overlays if required
OUTPUT
Disk:
● $SOURCE – sorted segment list
Storage:
● LOMMON
To: Overlay Segment List Build (#OLAR)

DESCRIPTION	MODULE/ ROUTINE
1 Accumulate total storage size used by all segments in sorted segment list. Compare accumulated storage size to total usable storage size available to determine if overlay required. **2** Determine overlay structure dependent on storage size available. Identify duplicate or unused elements that can be eliminated based on overlay structure.	#OLAP
Issue error messages as needed.	#OLER

Diagram 8.7. Overlay Design (#OLAP)



From: Overlay Design (#OLAJ)

INPUT

Main Storage

Transient Area

User Area:
#OLAR

Disk:
- $SOURCE
  - Cross-reference segment list
  - Sorted segment list

Storage:
- LOMMON

PROCESS

1 Build overlay segment list

2 Assign addresses

3 Build text

OUTPUT

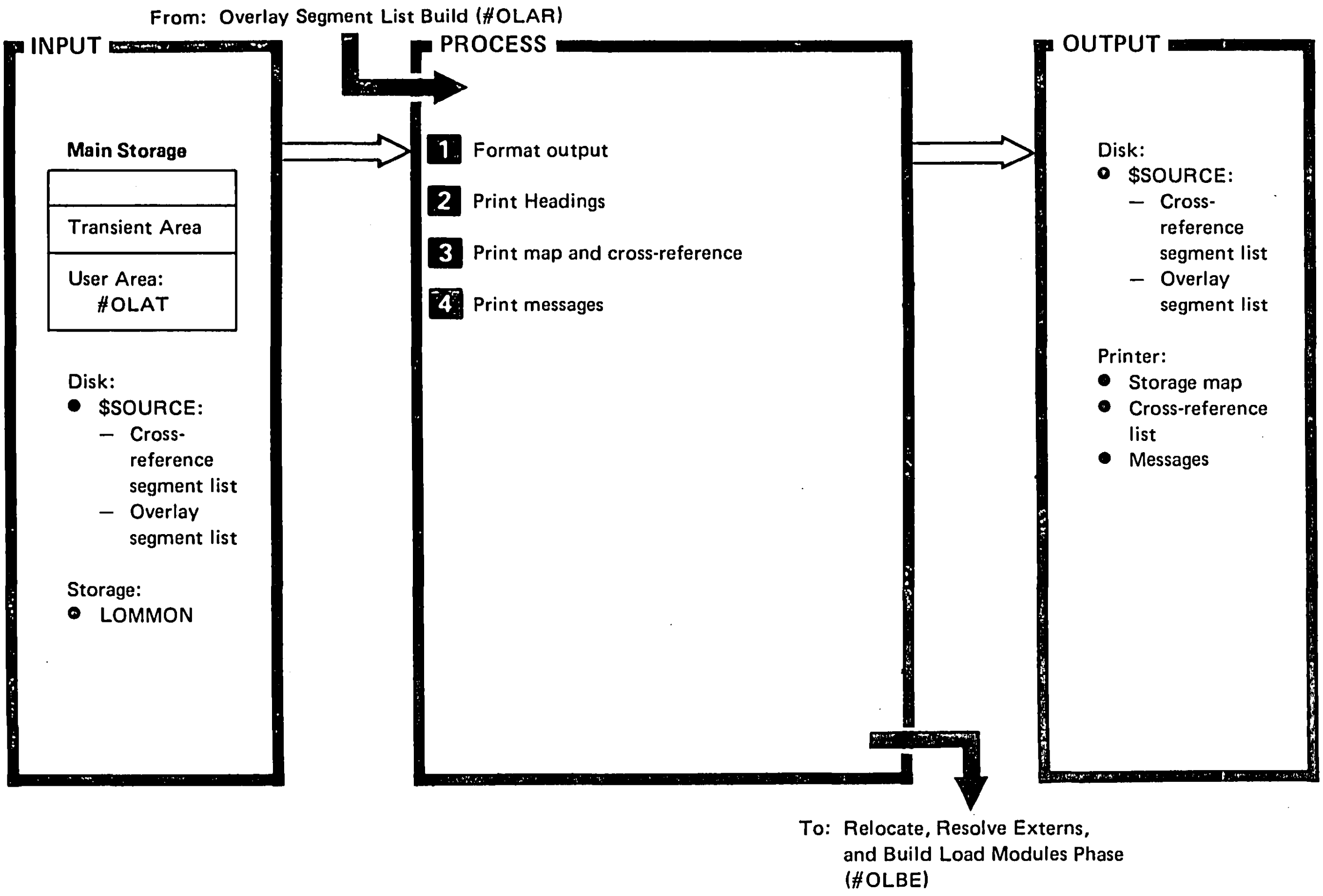
Disk:
- $SOURCE
  - Sorted segment list
  - Overlay segment list
- $WORK
  - Test records

Storage:
- LOMMON

To: Storage Map Phase (#OLAT)

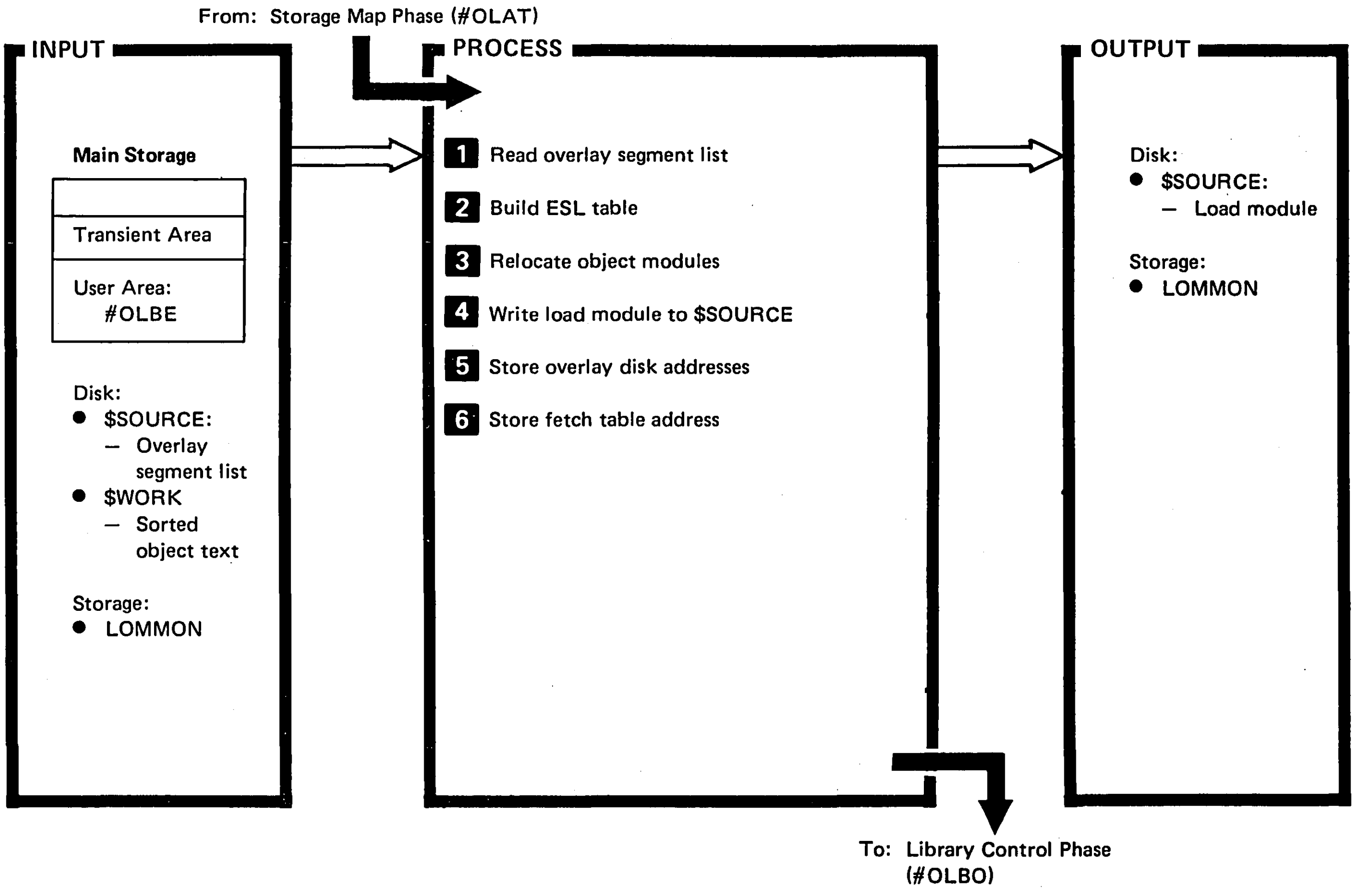
DESCRIPTION	MODULE/ ROUTINE
1 Read data into main storage from sorted segment list and cross-reference segment list. Reformat data to build overlay segment list. **2** Assign addresses to each module and EXTRN entry in overlay segment list. **3** Build object text for: ● Overlay fetch routine. ● Overlay fetch table. ● Overlay transfer vectors.	#OLAR
Issue error messages as needed.	#OLER

Diagram 8.8. Overlay Segment List Build (#OLAR)



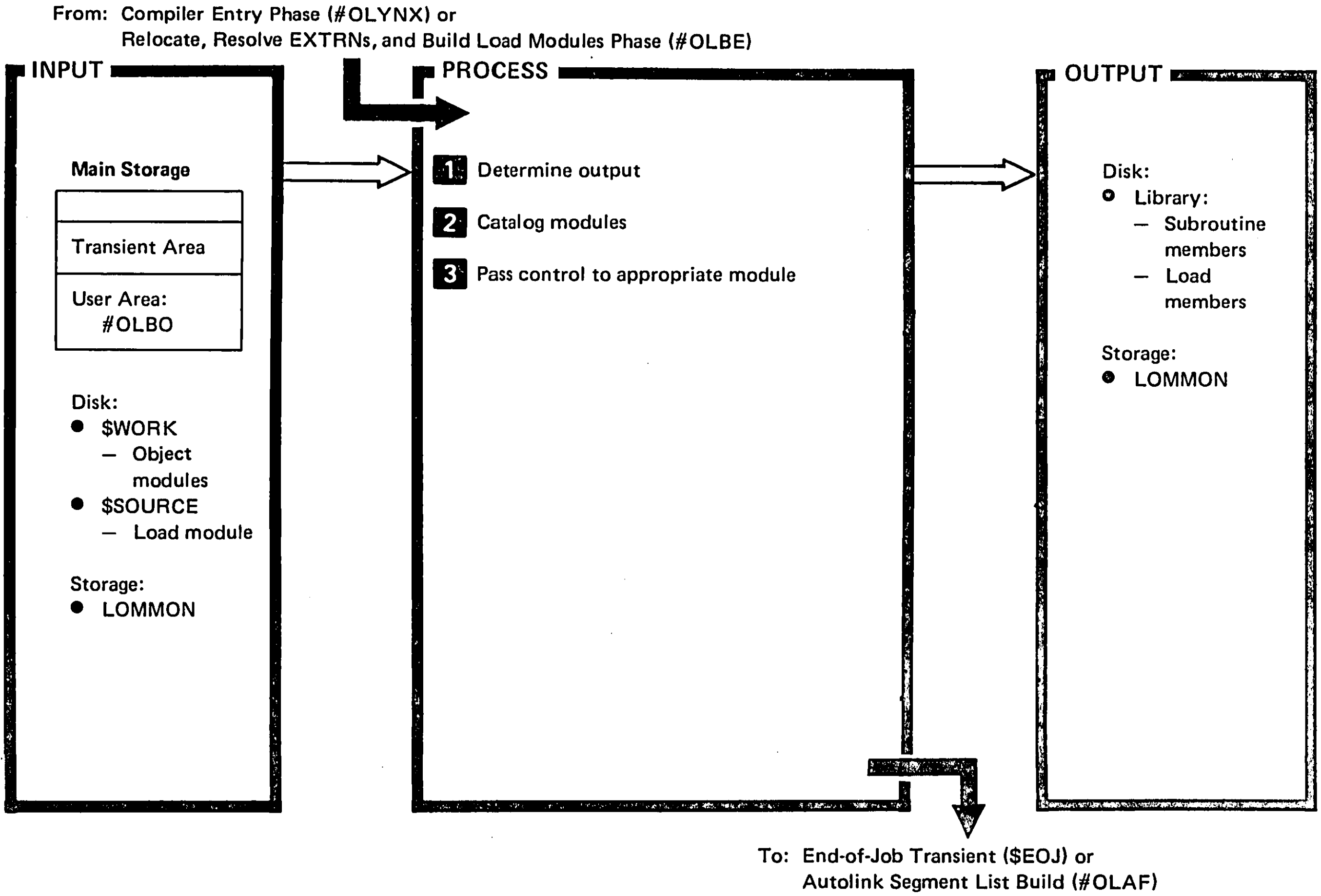
DESCRIPTION	MODULE/ ROUTINE
1 Format output according to option selected — output contains: ● Storage map. ● Cross-reference list. ● Messages.	#OLAT
2 Output heading to printer.	
3 Read overlay segment list. Print entry for each common area and module including cross-reference and entry point.	
4 Print informational messages and error messages (if any) according to option selected.	
Issue error messages as needed.	#OLER

Diagram 8.9. Storage Map Phase (#OLAT)


From: Storage Map Phase (#OLAT)
INPUT
Main Storage
Transient Area
User Area:
#OLBE
Disk:
● $SOURCE:
– Overlay segment list
● $WORK
– Sorted object text
Storage:
● LOMMON
PROCESS
1 Read overlay segment list
2 Build ESL table
3 Relocate object modules
4 Write load module to $SOURCE
5 Store overlay disk addresses
6 Store fetch table address
OUTPUT
Disk:
● $SOURCE:
– Load module
Storage:
● LOMMON
To: Library Control Phase (#OLBO)

DESCRIPTION	MODULE/ ROUTINE
1 Read data into main storage from overlay segment list.	#OLBE
2 Build ESL table entry for each external reference in overlay segment list.	
3 Read object module for each overlay.	
Place RLDs in buffer to be written to load module.	
Adjust and place text records addresses in buffer to be written to load module.	
4 Place RLDs and text record addresses in load module.	
Write load module to $SOURCE.	
5 Put relative disk address of each overlay into overlay fetch table in root phase.	
6 Put overlay fetch table address following last RLD in root phase.	
Issue error messages as needed: ● If message is to be displayed, use #OLER.	#OLER
● If message is to be printed, use #OLMSG.	#OLMSG

Diagram 8.10. Relocate, Resolve EXTRNs, and Build Load Module Phase (#OLBE)



From: Compiler Entry Phase (#OLYNX) or
Relocate, Resolve EXTRNs, and Build Load Modules Phase (#OLBE)

INPUT

Main Storage

Transient Area

User Area:
#OLBO

Disk:
- $WORK
  - Object modules
- $SOURCE
  - Load module

Storage:
- LOMMON

PROCESS

1 Determine output

2 Catalog modules

3 Pass control to appropriate module

OUTPUT

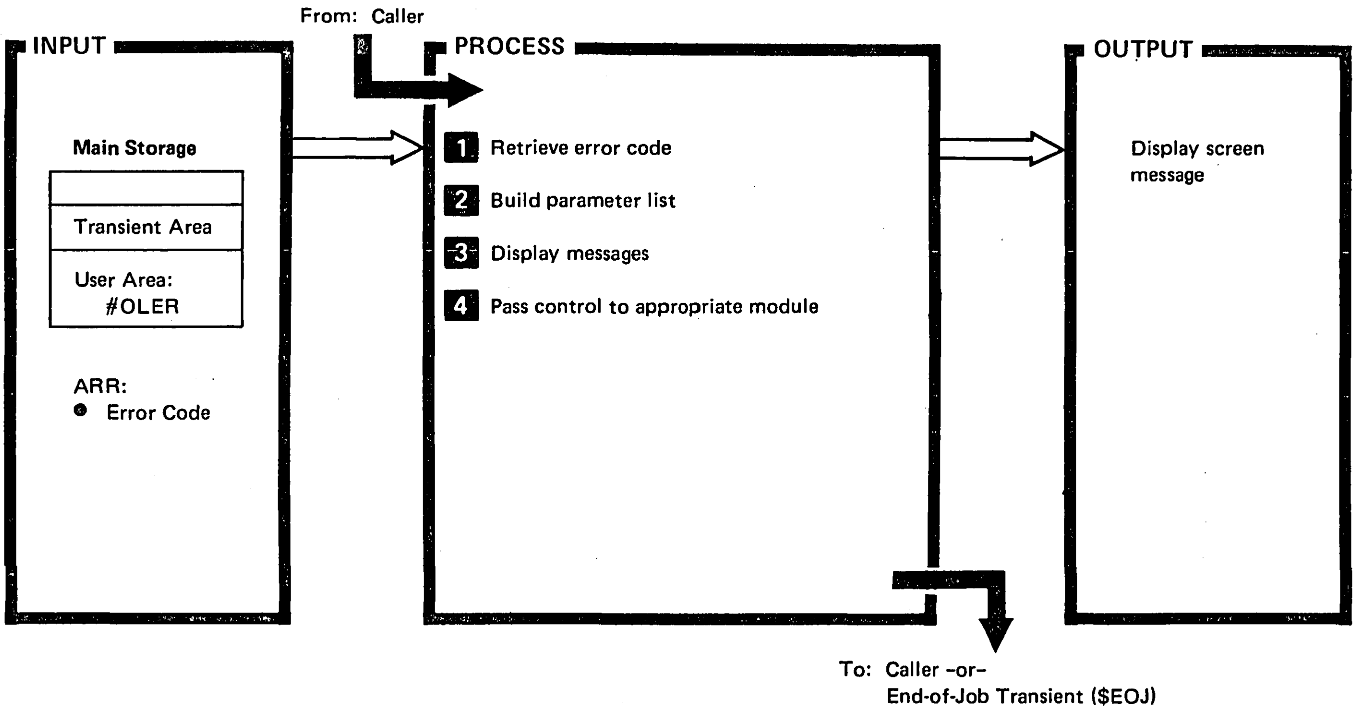
Disk:
- Library:
  - Subroutine members
  - Load members

Storage:
- LOMMON

To: End-of-Job Transient ($EOJ) or
Autolink Segment List Build (#OLAF)

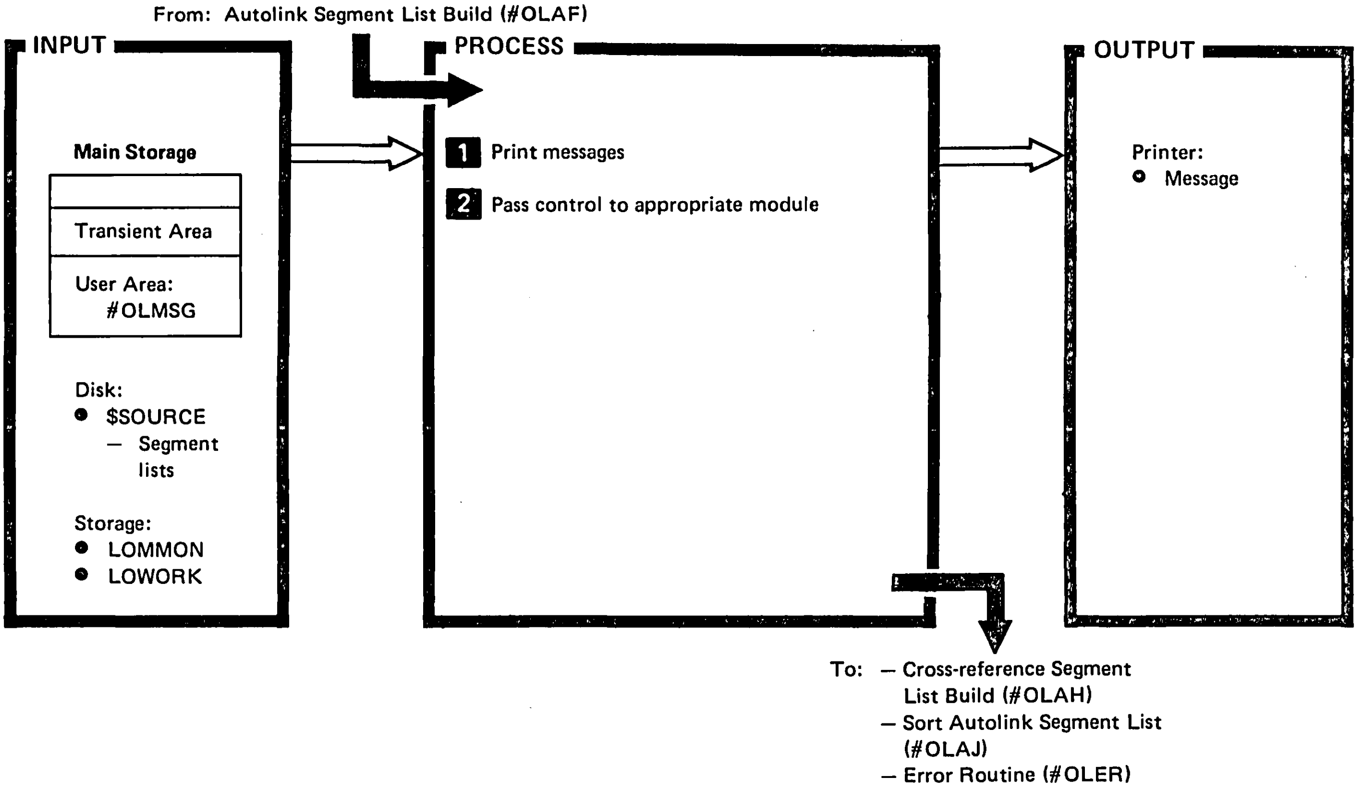
DESCRIPTION	MODULE/ ROUTINE
1 Check LOMMON to determine if object module or load module to be cataloged in system library.	#OLBO
2 Create entry for module in library directory containing: ● Module type. ● Name. ● Relative sector address. ● Other library directory information.	$MAPGS
3 Branch to autolink segment list build (#OLAF) if user program requests that object module cataloged be linked into load module. Otherwise pass control to end-of-job transient ($EOJ).	#OLBO
Issue error messages as needed; if message is to be displayed, use #OLER.	#OLER

Diagram 8.11. Library Control Phase (#OLBO)



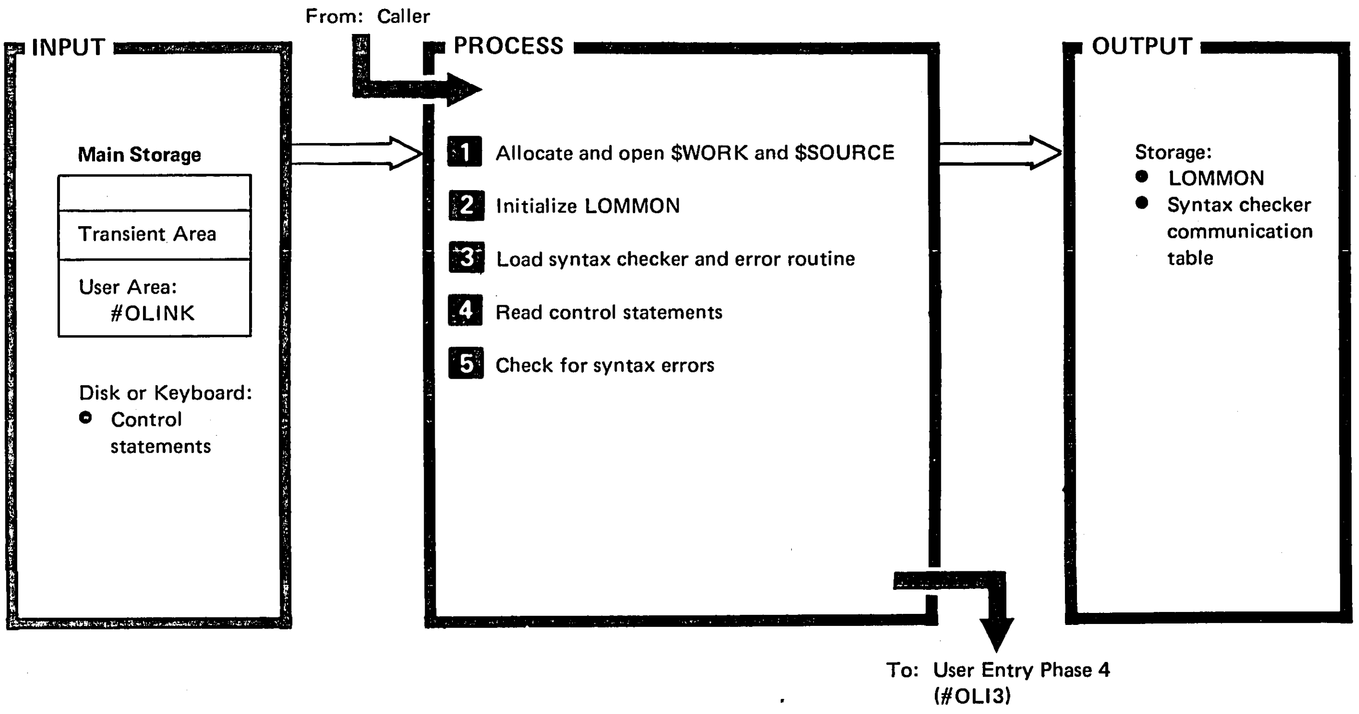
DESCRIPTION	MODULE/ ROUTINE
1 Address recall register (ARR) contains address of error code used as displacement into error table.	#OLER
2 Build and pass parameter list to SYSLOG.	
3 Display messages on SYSLOG device.	#CLSG
4 If option 3 response to error, #CLSG passes control to end-of-job transient ($EOJ).	#OLER
If option 0, 1, or 2 response to error, control returns to caller.	

Diagram 8.12. Error Routine (#OLER)



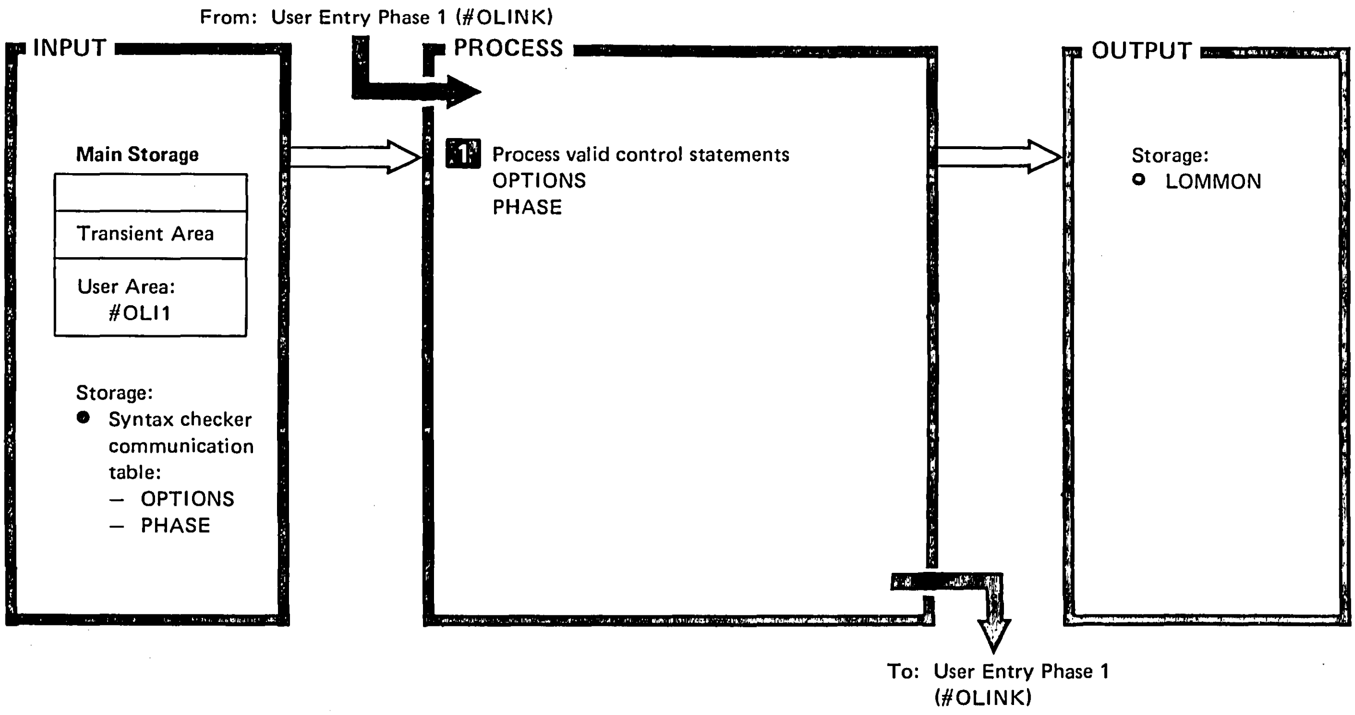
DESCRIPTION	MODULE/ ROUTINE
1 Print message on system printer.	#DPDM
2 Exit to error routine (#OLER) for terminal errors. Exit to sort autolink segment list (#OLAJ) if message 3109 or 3111 issued. Otherwise exit to cross-reference segment list build (#OLAH).	#OLMSG

Diagram 8.13. Error Message Print Phase (#OLMSG)



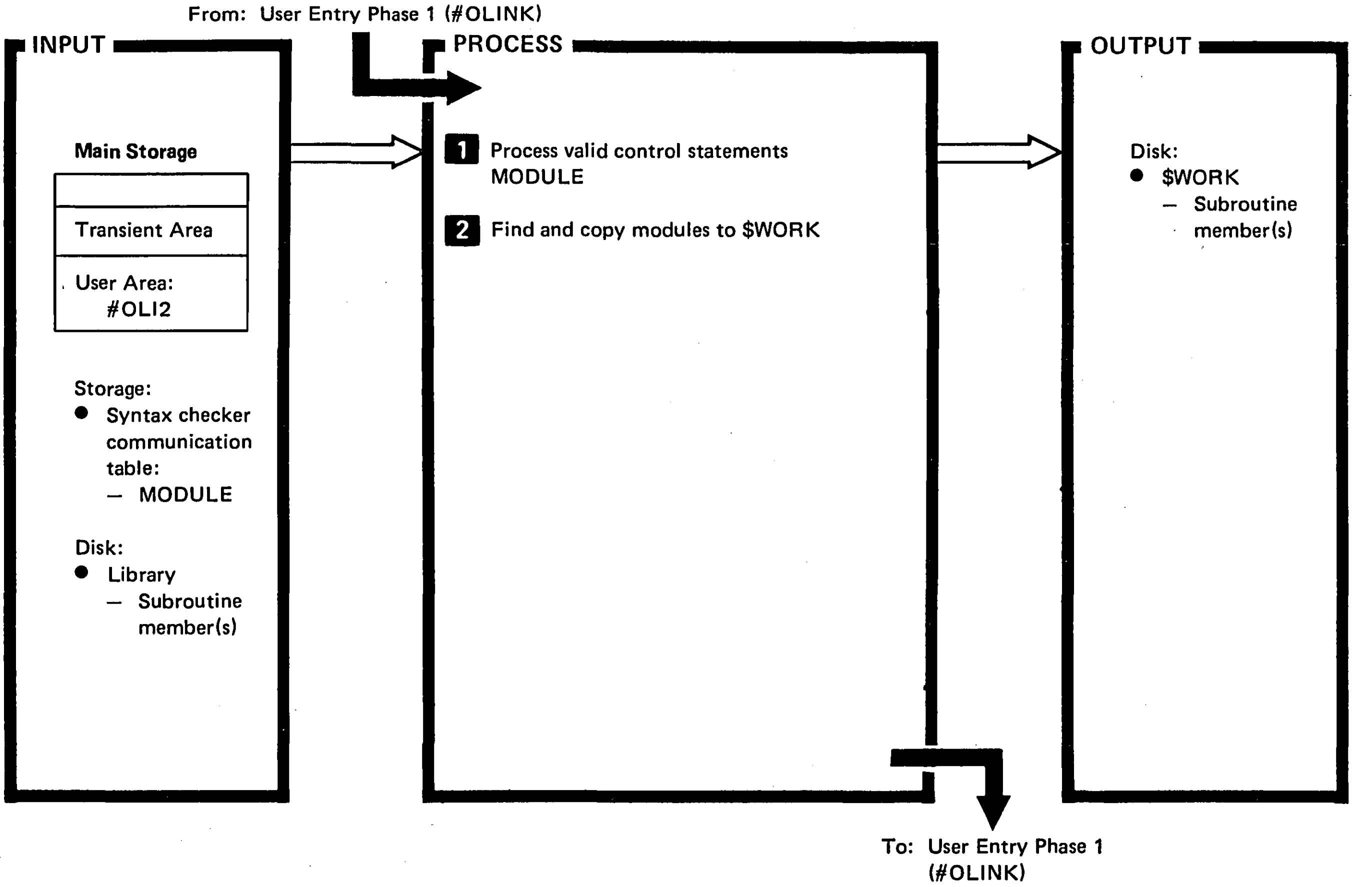
DESCRIPTION	MODULE/ ROUTINE
1 Allocate $WORK and $SOURCE files.	#OLINK
Open $WORK and $SOURCE files.	
2 Initialize 800-byte work area used by overlay linkage editor routines (LOMMON).	
3 Load syntax checker (#USYX) to compare control statement parameters with #OLISP module.	
Load error routine (#OLER) to display error messages (if any).	
4 Read control statements:	
○ // PHASE	#OLI1
○ // OPTIONS	
▷ ○ // MODULE	#OLI2 ◀
○ // GROUP	#OLI3
○ // CATEGORY	
○ // EQUATE	
○ // END	
5 Syntax check control statement parameters.	#USYX
Issue error messages as needed; if message is to be displayed, use #OLER.	#OLER

Diagram 8.14. User Entry Phase 1 (#OLINK)



DESCRIPTION	MODULE/ ROUTINE
1 Retrieve OPTIONS and PHASE data from syntax checker communication table. Put data in LOMMON.	#OLI1
Issue error messages as needed; if message is to be displayed, use #OLER.	#OLER

Diagram 8.15. User Entry Phase 2 (#OLI1)



From: User Entry Phase 1 (#OLINK)

INPUT

Main Storage

Transient Area
User Area: #OLI2

Storage:
- Syntax checker communication table:
  - MODULE

Disk:
- Library
  - Subroutine member(s)

PROCESS

1. Process valid control statements MODULE
2. Find and copy modules to $WORK

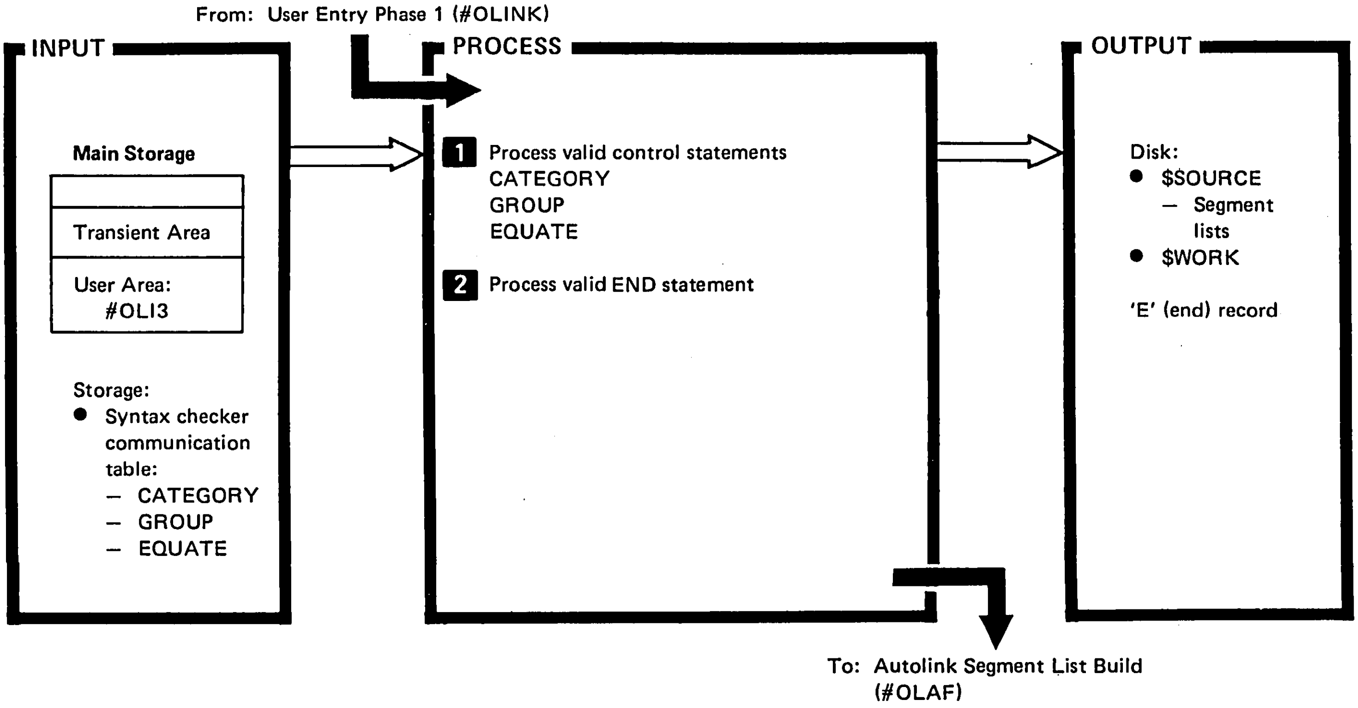
OUTPUT

Disk:
- $WORK
  - Subroutine member(s)

To: User Entry Phase 1 (#OLINK)

DESCRIPTION	MODULE/ ROUTINE
1 Retrieve names of modules to be included in load module from syntax checker communication table.	#OLI2
2 Find modules to be included and put modules in $WORK on disk.	$MALFN
Issue error messages as needed: ● If message is to be displayed, use #OLER.	#OLER
● If message is to be printed, use #OLMSG.	#OLMSG

Diagram 8.16. User Entry Phase 3 (#OLI2)


From: User Entry Phase 1 (#OLINK)
INPUT
Main Storage
Transient Area
User Area:
#OLI3
Storage:
● Syntax checker communication table:
– CATEGORY
– GROUP
– EQUATE
PROCESS
1 Process valid control statements
CATEGORY
GROUP
EQUATE
2 Process valid END statement
OUTPUT
Disk:
● $SOURCE
– Segment lists
● $WORK
'E' (end) record
To: Autolink Segment List Build (#OLAF)

DESCRIPTION	MODULE/ ROUTINE
1 Process valid CATEGORY, GROUP, or EQUATE control statement: ● Retrieve control statement data from syntax checker communication table. ● Build preautolink segment list. **2** Retrieve END statement from syntax checker communication table. Perform final write to $WORK and $SOURCE. Pass control to autolink segment list build (#OLAF).	#OLI3
Issue error messages as needed; if message is to be displayed, use #OLER.	#OLER

Diagram 8.17. User Entry Phase 4 (#OLI3)

Program Organization

The overlay linkage editor is divided into self-overlaying routines. The sequence in which routines are loaded and which routines are used depends on whether the compiler entry or the user entry is used and which functions are required. Figures 8-5 and 8-6 are the storage maps for the compiler entry and user entry overlay linkage editor modules.

Figures 8-7 and 8-8 show the overlay linkage editor compiler entry and user entry control flow.

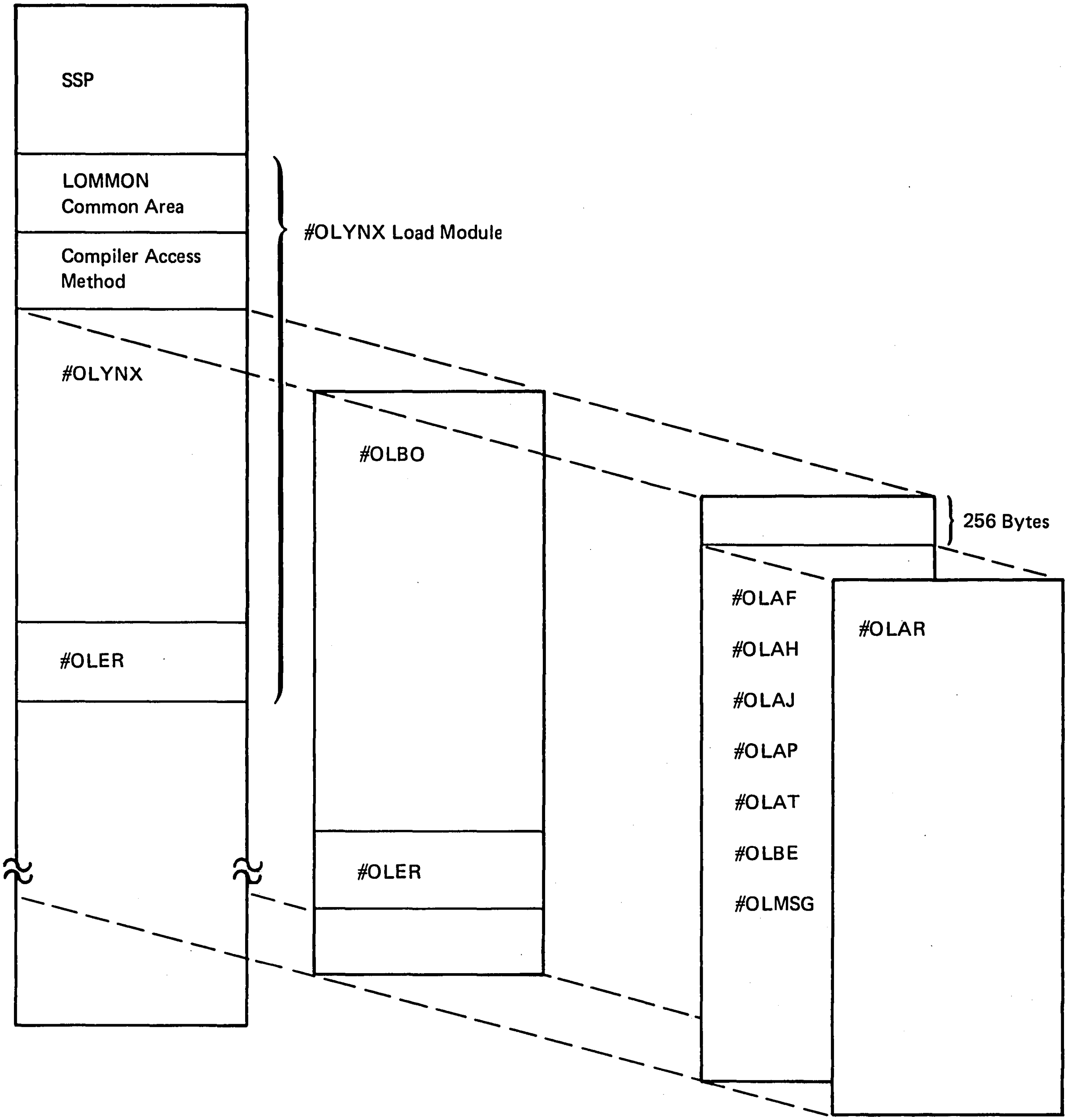


Figure 8-5. Compiler Entry Storage Map

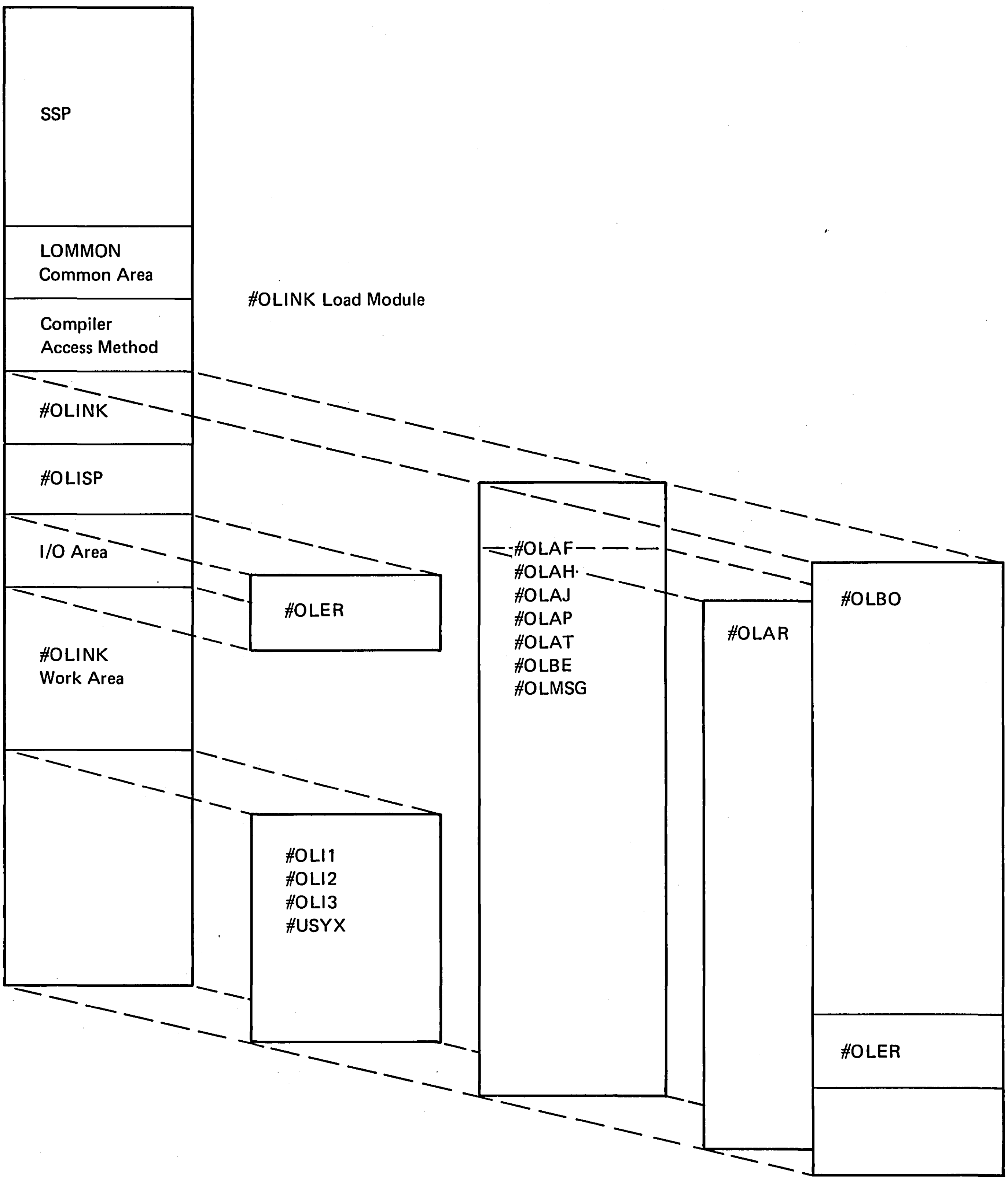


Figure 8-6. User Entry Storage Map

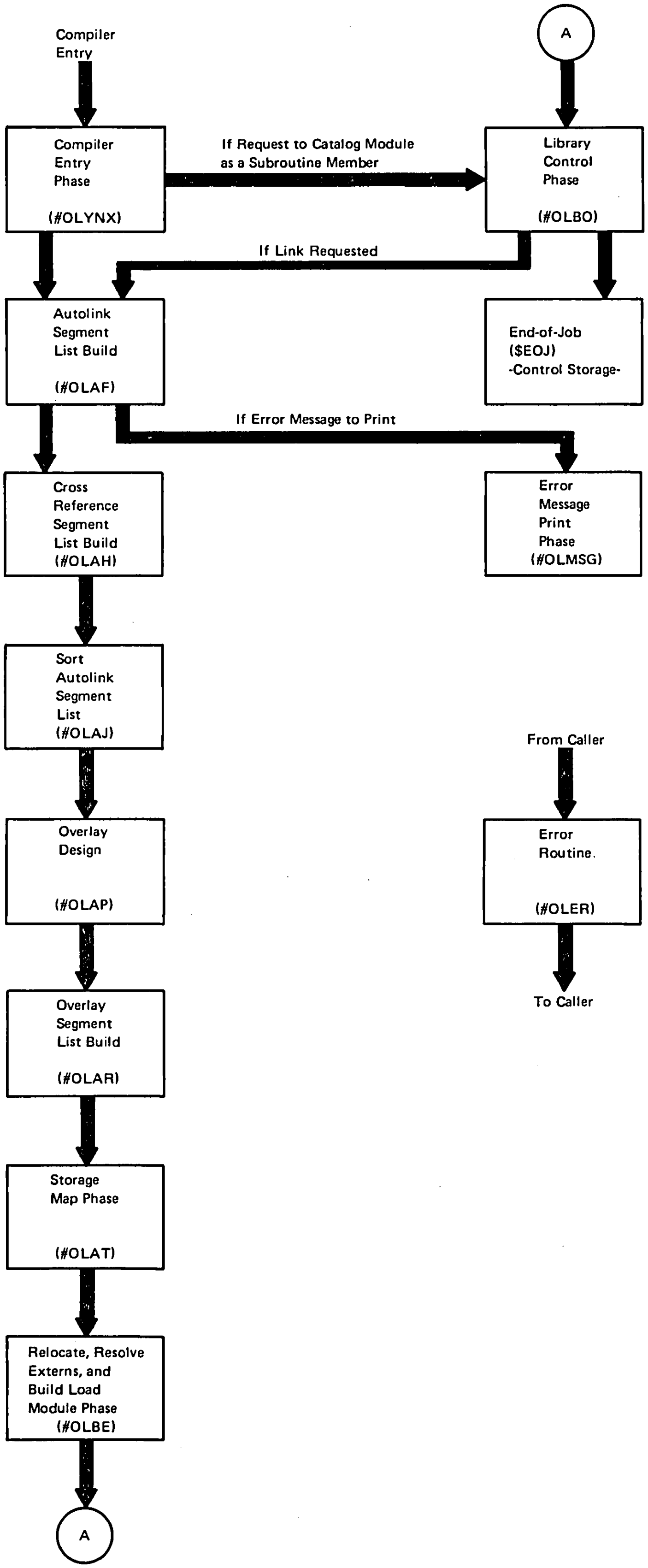

Compiler Entry
A
Compiler Entry Phase (#OLYNX)
If Request to Catalog Module as a Subroutine Member
Library Control Phase (#OLBO)
If Link Requested
Autolink Segment List Build (#OLAF)
End-of-Job ($EOJ) -Control Storage-
If Error Message to Print
Cross Reference Segment List Build (#OLAH)
Error Message Print Phase (#OLMSG)
Sort Autolink Segment List (#OLAJ)
From Caller
Overlay Design (#OLAP)
Error Routine. (#OLER)
To Caller
Overlay Segment List Build (#OLAR)
Storage Map Phase (#OLAT)
Relocate, Resolve Externs, and Build Load Module Phase (#OLBE)
A

Figure 8-7. Overlay Linkage Editor Compiler Entry Control Flow

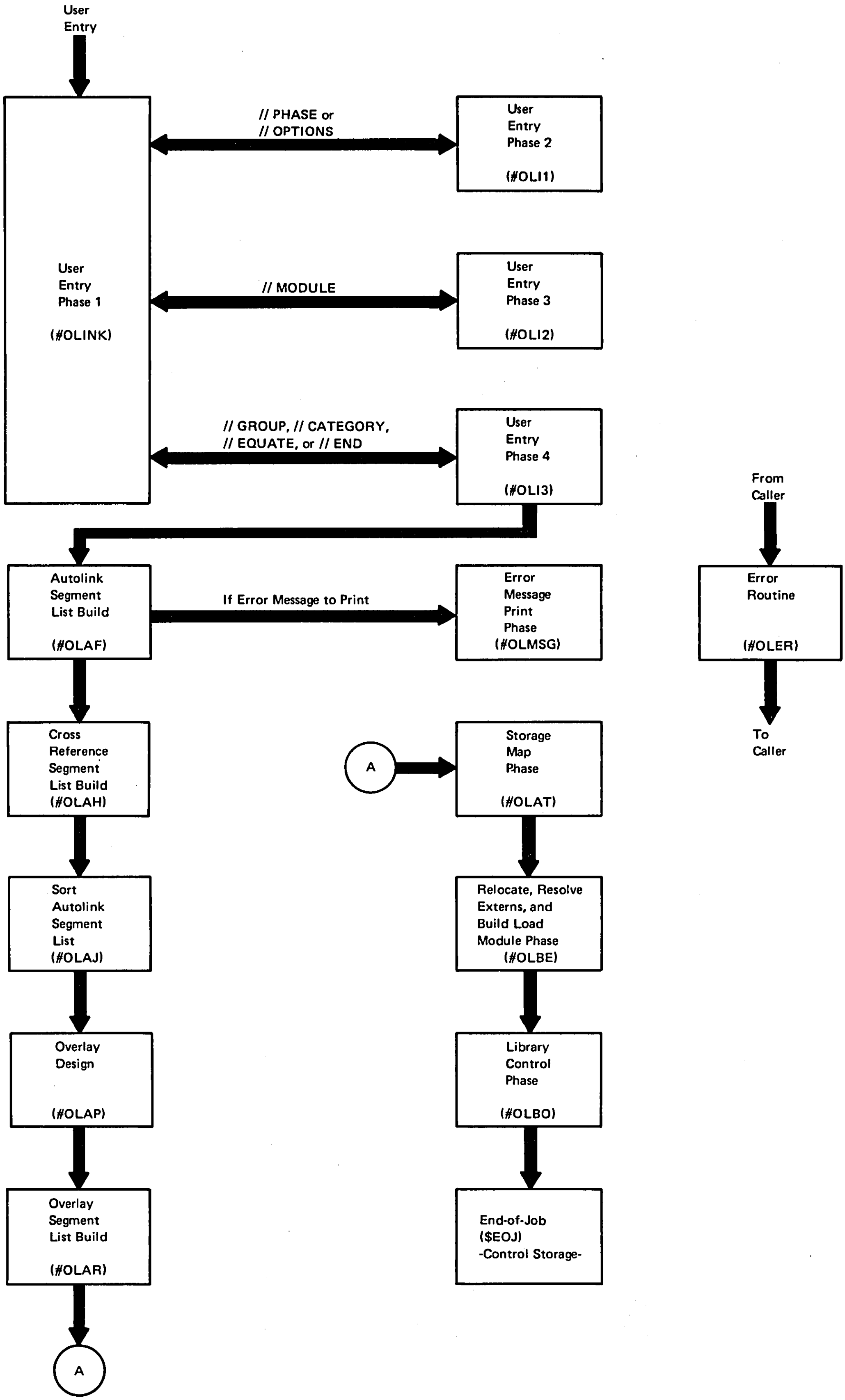


Figure 8-8. Overlay Linkage Editor User Entry Control Flow

Data Areas

This section describes the data areas that pass information between routines of the overlay linkage editor.

OVERLAY LINKAGE EDITOR COMMON (LOMMON)

The overlay linkage editor common area (Figure 8-9) passes control information between the various routines. Most of LOMMON is initially set to zero by user entry phase 1 (#OLINK) or compiler entry phase (#OLYNX).

VERB LIST (OVERBS)

OVERBS is a list of constants passed to the syntax checker (#USYX) each time the syntax checker is called by #OLINK. The third and fourth bytes of the syntax checker parameter list contain the address of the verb list. Figure 8-10 shows the format and contents of the verb list.

SEGMENT LIST ENTRIES

The various routines of the overlay linkage editor build a series of segment lists. These segment lists are built in the $SOURCE work file (Figure 8-11).

Each entry is 16 bytes long. The format of entries varies between and within segment lists, depending on the type of entry. (See Figures 8-17 through 8-22 for segment list formats.) All data fields in the segment list entries are not used for all types of entries, the column heading *Applies to Segment Type* in each segment list indicates which types of entries contain the data. Figure 8-17 contains a list of all the segment types.

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes in Decimal	Description	Routines that Change Data (#OLxxx)
0	LORTYP	1	Object module information X'80' = Reserved X'40' = Reserved X'20' = Reserved X'10' = Reserved X'08' = Reserved X'04' = Catalog the object module as a subroutine member with a RETAIN-R (replace) in the library X'02' = Catalog the object module as a subroutine member with a RETAIN-P (permanent) in the library X'01' = Reserved If this byte is X'00', there is no object module	YNX, BO
1	LOOTYP	1	Load module information X'80' = Reserved X'40' = Reserved X'20' = Reserved X'10' = Reserved X'08' = Reserved X'04' = No storage map option X'02' = No cross-reference list option X'01' = Catalog the load module as a load member in the library X'00' = Do not catalog the load module (If neither an object or a load module is specified, a load member is cataloged)	INK, YNX, I2, I3, AT
2	LOSWT1	1	Overlay linkage editor switch 1 X'80' = Segment list is in $SOURCE X'40' = User call X'20' = User specified overlays X'10' = Entry point to be resolved X'08' = Groups in segment list X'04' = Reserved X'02' = Print messages X'01' = Retain-R (replace)	INK, YNX, AF, I1, I2, I3

Figure 8-9 (Part 1 of 5). Common Area (LOMMON)

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes in Decimal	Description	Routines that Change Data (#OLxxx)
3	LOSWT2	1	Overlay linkage editor error switch X'80' = System category calls another category X'40' = DTF in last 1K of storage X'20' = All elements in group are category 1-7 X'10' = No module for group element X'08' = Entry point not in common program X'04' = Entry on options not found X'02' = Storage size on options statement X'02' = Program will not fit X'01' = Terminal error	AF, AH, AJ, AR, AT, I1, YNX
4	LOUSLB	2	User library format 1 address	INK, YNX
6	LOSYLB	2	System library format 1 address	INK, YNX
8	LORLIB	2	Subroutine member library format 1 address	INK, YNX
A	LOOLIB	2	Load member library F1 address	INK, YNX
C	LONOVL	1	Number of overlays	AP
D	LOEND@	2	End of storage address	INK, YNX
F	LOCRSZ	2	Actual storage size for storage map execution	AP, YNX, BO
11	LOOVNO	2	Low, high system overlay numbers	AR
13	LOFTBL	2	Displacement of overlay fetch table	AR
15	LOAUTO	2	Relative entry number of auto segment list	I3
17	LOXREF	2	Relative entry number of cross-reference segment list	AF, BE
19	LOSORT	2	Relative entry number of sort segment list	AH, BE
1B	LOOVER	2	Relative entry number of overlay segment list	AJ, AP, BE
1D	LOLIMT	2	Relative entry number of last delimiter	AR, BE
1F	LOWKCS	2	Relative sector number of the next sector in $WORK	I3, I2, INK, YNX, AF, AR
21	LOLCSB	3	Sector address of data start	INK, YNX, BE

Figure 8-9 (Part 2 of 5). Common Area (LOMMON)

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes in Decimal	Description	Routines that Change Data (#OLxxx)
24	LOLHDR	1	Library type; R (subroutine) or O (load member)	INK, YNX, BE
25	LOLNAM	6	Module name	IN1, YNX, I2, BO
2B	LOLLCS	2	Sector address of library entry	BO
2D	LOLTXS or	1	Number of text sectors in load member	BE, BO
2D	LOLCAT	1	Overlay category of subroutine member X'00' = Root category X'01' through X'7E' = Overlay category value	YNX
2E	LOLLEA	2	Link edit addresses	INK, YNX, AR, I1
30	LOLRLD	1	RLD displacement	BE
31	LOLSCA	2	Start control address	YNX, I2, AJ, AR
33	LOLCSZ	1	Main storage size in hexadecimal (in 1/4 K increments)	INK, YNX, I1, BO
34	LOATB1	1	1st attribute byte X'80' = SSP module X'40' = Privileged module — O only X'40' = Do not log OCL — P only X'20' = Non-inquirable module X'10' = SFGR format load module — O only X'10' = Procedure with data — P only X'08' = Source required X'04' = Non-base SSP module X'02' = PTF applied bit X'01' = Module has overlays	R/O I1, I2, AF, BE, BO, YNX, AJ
35	LOATB2	1	2nd attribute byte X'80' = Dedicated module X'40' = Never-ending program module X'20' = Module has OXRF format index table X'10' = Module can only be loaded from system console X'08' = Cannot load program via // LOAD X'04' = Program common X'02' = Program with utility control statements X'01' = Module has OXRF WTG table	R/O I1, I2, AF, BE, BO, YNX, AJ

Figure 8-9 (Part 3 of 5). Common Area (LOMMON)

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes in Decimal	Description	Routines that Change Data (#OLxxx)
36	LOATB3	1	3rd attribute byte X'80' = $WORK2 file required X'40' = Do not swap this task X'20' = High level of dedication X'10' = Reserved X'08' = Reserved X'04' = Reserved X'02' = Reserved X'01' = Reserved	IN1, IN2, AF, BE, BO, YNX, AJ
37	LOMRTX	1	MRTMAX	YNX, I1, BO
38	LOJCB@	2	JCB address	YNX, I1, AT
3A	LOLLVL	1	Release level	IN1, INK, YNX
3B	LOLTSC	2	Total sector count	YNX, BE
3D	LOWORK	30	Phase work area *Note:* The phase work area can be used by each phase. Information can be passed from one phase to the next in this area, but the phase work area is not to be used for passing information from one phase to phases that are two or three loads away.	any
5C	LOCZER	2	Constant of zero	INK, YNX
5E	LOCONE	1	Constant of one	INK, YNX
5F	LOCHFF or	2	Constant X'FFFF'	INK, YNX
5F	LOCM1	2	Constant of minus one	INK, YNX
61	LOSCAT	1	System category	AH
62	LOERCD	1	Error code	AT, AF
63	LOENTR	6	Entry point name	INK, I1, AH, YNX
69	LOERR	1	#OLER error code	–
69	LOAF	5	Load list for #OLAF	YNX
6E	LOAH	5	Load list for #OLAH	AF
73	LOAJ	5	Load list for #OLAJ	AF

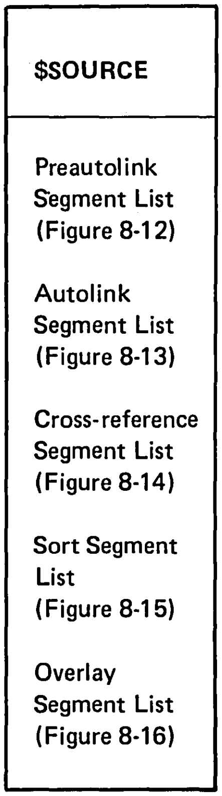
Figure 8-9 (Part 4 of 5). Common Area (LOMMON)

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes in Decimal	Description	Routines that Change Data (#OLxxx)
78	LOAP	6	Load list for #OLAP	AF
7D	LOAR	6	Load list for #OLAR	AF
82	LOAT	6	Load list for #OLAT	AF
87	LOBE	6	Load list for #OLBE	AF
8C	LOBO	6	Load list for #OLBO	AF
91	LOER	5	Load list for #OLER	INK, YNX
96	LOMS	5	Load list for #OLMSG	AF
9C	LOSWT3	1	Flag byte: X'80' = RLDs for load member	INK, YNX, MSG, AT
AD	LOEND	17	End of LOMMON	–
AE	LOIOBS	32	IOB for $SOURCE	–
CE	LOIOBW	32	IOB for $WORK	–
EE	LOCAMS	27	$SOURCE CAM (compiler access method) control block	–
109	LOCAMW	27	$WORK CAM control block	–
124	LODTFP	41	DTF for printer	–
14D	CAM	182	Start of CAM	–
203	LOPRCA	132	Printer logical record	–
288	LOPIOA	152	Printer physical area	–
320	LOPHSE	(variable)	phase area	–

Figure 8-9 (Part 5 of 5). Common Area (LOMMON)

Displacement of Leftmost Byte	Label	Length in Bytes	Description
0	OVERBS	1	PHASE verb ID
1		1	OPTIONS verb ID
2		1	MODULE verb ID
3		1	GROUP verb ID
4		1	CATEGORY verb ID
5		1	EQUATE verb ID
6		1	END verb ID
7		1	X'FF' end of verb list

Figure 8-10. OVERBS Verb List


$SOURCE
Preautolink Segment List (Figure 8-12)
Autolink Segment List (Figure 8-13)
Cross-reference Segment List (Figure 8-14)
Sort Segment List (Figure 8-15)
Overlay Segment List (Figure 8-16)

Segment List Entry Types

00	Module name
01	Entry point
02	EXTRN
03	Weak EXTRN
04	Global common
05	Local common
06	Conditional EXTRN
0B	EQUATE entry
0C	Transfer vector
0D	Reference a previous name or entry point
0E	GROUP entry
0F	CATEGORY entry
FE	Nulled entry
FF	End of segment list

Figure 8-11. Segment Lists in $SOURCE and Segment List Entry Types

Displacement of Leftmost Byte in Hexadecimal	Applies to Segment Type	Length in Bytes	Description	Routines that Change Data (#OLxxx)
0	0E, 0F	1	Bits 0-3 = Reserved	I3
0	0E, 0F, 0B	1	Bits 4-7 = segment type[1]	I3
1	0E	1	Group number	I3
1	0E	1	Category override number	I3
2	0E	—	Work area — original category	AH
2	0E, 0F	4	Reserved	I3
3	0E	1	Bit 7 = User area specified for module	I3
6	0E, 0F	2	Reference number — pointer to module element in autolink segment list[1]	AF
6	0E	2	Reference number — pointer to lead element in last overlay[2]	AJ
8	0E, 0F	2	ESL sequence number	AF
A	0E, 0F, 0B	6	Module name	I3
A	0E	2	Reserved	AF
C	0E	2	Module element pointer (moved from bytes 6 through 7)	AJ
E	0E	2	Reserved	AF

[1] See Figure 8-11

[2] Displacement within $SOURCE

Figure 8-12. Preautolink Segment List

Displacement of Leftmost Byte in Hexadecimal	Applies to Segment Type	Length in Bytes	Description	Routines that Change Data (#OLxxx)
0	–	1	This byte indicates the following	
	00		X'80' = Reserved	AF
	01, 02, 03, 04, 05		X'80' = Entry point or references an entry point	AF
	02, 03		X'40' = Resolved to module and/or entry	AF
	00, 01		X'40' = This module or entry point has an EXTRN referencing it	AF
	00, 01, 02, 03		X'20' = Work area must be OFF at phase end	AF
	00		X'20' = Used – do not place in structure	AJ
	00		X'10' = Calls a user routine or requires a transfer vector	AF
	02, 04, 05		X'10' = Delete this element when compressing list	AF
	00		X'10' = Module already placed in root	AJ
	All		Bits 4-7 = Segment type[1]	AF
1	00, 01, 02, 03	1	Category	AF, AH
2	00	2	$WORK address of object code	AF
2	01	2	Entry displacement from start of module	AF
2	00	1	Number of entry points	AH
3	00	1	Module information	
	00		X'80' = Module requires boundary alignment	AH
	00		X'40' = Module calls a user routine	AH
	00		X'20' = Module has I/O dependency	AH
	00		X'10' = Module already in an overlay	AJ
	00		X'10' = Substructure pointer already built	AJ

[1] See Figure 8-11

Figure 8-13 (Part 1 of 2). Autolink Segment List

Displacement of Leftmost Byte in Hexadecimal	Applies to Segment Type	Length in Bytes	Description	Routines that Change Data (#OLxxx)
4	00	2	Object code length	AF
6	00, 01, 02, 03, 04	2	Reference number — pointer to equal ESL number in autolink segment list	AF
8	00, 01, 02, 03, 04, 05	2	ESL number	AF
A	00, 01, 02, 03, 04, 05	6	ESL name	AF, INK
A	00	2	Reference number — pointer to equal 00 type in cross-reference segment list	AH
C	00	2	Work area	AJ
E	00	2	Reserved	AH

Figure 8-13 (Part 2 of 2). Autolink Segment List

Displacement of Leftmost Byte in Hexadecimal	Applies to Segment Type	Length in Bytes	Description	Routines that Change Data (#OLxxx)
0	—	1	This byte indicates the following	
	00, 01, 0D		X'80' = Reserved	AH
	00, 01		X'40' = Reserved	AH
	00, 01, 0D		X'20' = Reserved	AH
	00, 01, 0D		X'10' = Reserved	AH
	00, 01, 0D		Bits 4-7 = Segment type[1]	AH
1	00, 01, 0D	1	Category	AH

[1] See Figure 8-11

Figure 8-14 (Part 1 of 2). Cross-Reference Segment List

Displacement of Leftmost Byte in Hexadecimal	Applies to Segment Type	Length in Bytes	Description	Routines that Change Data (#OLxxx)
2	00, 01	2	Entry point displacement from start of module	AH
4	00	1	Work area = number of entry points on original category	AH
5	–	1	This byte indicates the following:	
	00		X'80' = Module requires boundary alignment	AH
	0D		X'80' = Categories make this call a potential program failure	AH
	00		X'40' = Module calls a user routine	AH
	00		X'20' = Module has I/O dependency	AH
	00		X'10' = Work area – OFF at end of phase	AH
	00		X'08' = Work area – OFF at end of phase	AH
	00		X'04' = No reference made to this module	AH
	01		X'04' = Same name as module name	AH
	00, 01		X'02' = Duplicate name	AH
	00, 01		X'01' = Start control label	AH
6	00	2	Location of object text in $WORK (X'FFFF' = nontext)	AH
8	00, 01, 0D	2	ESL number	AH
A	00, 01, 0D	6	Module or entry point name	AH

Figure 8-14 (Part 2 of 2). Cross-Reference Segment List

Displacement of Leftmost Byte in Hexadecimal	Applies to Segment Type	Length in Bytes	Description	Routines that Change Data (#OLxxx)
0	–	1	This byte indicates the following:	
	0C		Bits 0-3 = Set of modules already summed	AP
	0C		X'40' = Set of modules contains a boundary alignment module	AP
	00, 02, 04, 05, 0C		Bit 0-3 = Reserved	AJ
	00, 02, 04, 05, 0C		Bits 4-7 = Segment type[1]	AJ
1	00, 02	1	Category	AJ
1	00, 0C	1	Overlay number	AP
2	00	1	Number of entry points this module	AJ
2	0C	1	Number of entry points this overlay	AP
3	–	1	This byte indicates the following:	
	00		X'80' = Module requires boundary alignment	AJ
	00		X'40' = Module calls a user routine	AJ
	00		X'20' = Module has I/O dependency	AJ
	00		X'20' = Work area	AP
	00		Bits 4-7 = Reserved	AJ
3	0C	1	Overlay area used by this set of modules at execution time	AJ

[1] See Figure 8-11

Figure 8-15 (Part 1 of 2). Sort Segment List

Displacement of Leftmost Byte in Hexadecimal	Applies to Segment Type	Length in Bytes	Description	Routines that Change Data (#OLxxx)
4	00, 04, 05	2	Length of object area associated with this ESL	AJ
4	0C	2	Length of object area for this overlay candidate	AP
6	00	2	Reference number – pointer to equal module	AJ
6	02	2	Pointer to module	AJ
8	00, 02, 04, 05	2	ESL number	AJ
A	00	2	Pointer to module name element in cross-reference list	AJ
A	0C	2	Pointer to next set of modules in same overlay	AP
C	00	2	Chain to substructure referencing this module	AJ
C	02	2	Chain to other substructure and module	AJ
C	0C	2	Chain to last previous transfer vector element	AJ
E	02, 04, 05, 0C	2	Reserved	AJ
E	00	2	Boundary alignment adjustment factor	AP

Figure 8-15 (Part 2 of 2). Sort Segment List

Displacement of Leftmost Byte in Hexadecimal	Applies to Segment Type	Length in Bytes	Description	Routines that Change Data (#OLxxx)
0	–	1	This byte has the following meaning:	
	02		X'80' = Work area = resolve to transfer vector	AR
	00, 02, 04, 05		Bits 4-7 = Segment type[1]	AR
1	00, 02, 04, 05	1	Overlay number	AR
2	00, 02, 04, 05	2	Object time address for this ESL	AR
4	00, 04, 05	2	Object time length for this ESL	AR
4	02	2	Corresponding module type ESL number	AR
6	00	2	Address of module's first transfer vector	AR
6	00	2	$WORK location of object text	AT
6	02	2	3-byte RLD object time address for this ESL	AR
8	00, 02, 04, 05	2	ESL number	AR
A	00	2	Pointer to equal 00 type in cross-reference segment list. X'FFFF' designates overlay fetch routine	AR
B	02	1	Relative entry point position	AR
C	00	2	Overlay size – first 00 type of overlay only	AR
C	02	2	Pointer to 00 type entry in sort list	AR
E	00, 02, 04, 05	2	Reserved	AR

[1] See Figure 8-11

Figure 8-16. Overlay Segment List

OVERLAY FETCH ROUTINE

The overlay fetch routine is added to the root segment of every program that has overlays. It is built by routine #OLAR. When an overlay segment is needed during program execution, the overlay fetch routine is called. It fetches overlay segments from access devices and places them in the overlay regions in main storage. Bits are set in the overlay fetch table telling which overlay region is used. The fetch table contains one 7-byte entry for each overlay in the program. Figure 8-17 shows the overlay fetch table entry format.

The overlay fetch routine requires three parameters as input:

1. Overlay number (1 byte)

2. Entry address of the overlay (2 bytes)

3. Return address from the overlay (2 bytes)

A transfer vector is built for each overlay in an object program. Transfer vectors provide input parameters for the overlay fetch routine. Overlay Linkage Editor routine #OLAR builds transfer vectors. Figure 8-18 shows the format of transfer vectors.

The overlay fetch routine checks to see if the requested overlay segment is already in main storage. If it is, the routine·branches to the entry address of the overlay; if not, the overlay fetch table entries are checked to see if they use the same main storage. If they do, the overlay is flagged as not being in main storage.

After the overlay fetch routine checks all entries in the overlay fetch table, it sets the overlay-in-storage bit in the overlay fetch table entry for the requested overlay. The overlay fetch routine then loads the overlay segment and branches to its entry address.

Relative Sector Address	Number of Sectors of Text	Main Storage Load Address	RLD	Flag Byte
0 1	2	3 4	5	6

Bytes **Contents**

0-1 — Relative sector address of the overlay segment. This is the number of sectors past the SSS@ of the root segment of the overlay program as given in the object library directory entry for the program.

2 — Number of sectors of text in the load module. (Does not include the number of related RLD sectors.)

3-4 — Relative main storage load address of where the overlay segment is to be placed in main storage by the system loader. (Relative to the end of the supervisor address.)

5 — RLD start displacement

6 — Flag byte — used at execution time by the root segments overlay fetch routine.

X'80'	Overlay in storage
X'60'	System overlay area (category values 1, 2, 3, 5, 6, and 7)
X'40'	Coresident area (user I/O-independent modules, category values 8-126)
X'20'	System overlay area (category value 4)
X'10'	User overlay area (user I/O-dependent modules, category values 8-126)
X'0F'	Reserved

Figure 8-17. Overlay Fetch Table Entry Format

ST	OVFRS1,ARR	Save the return address
B	OVFR	Call the overlay fetch routine
DC	XL1'NN'	One byte containing the overlay number
DC	AL2 (entry)	Two-byte entry address

Figure 8-18. Transfer Vector Format

HOW TO FIND AN OVERLAY

When a process check occurs, the following steps can determine which overlays are in main storage and where to find them.

1. Locate the address of the overlay fetch routine on the storage usage map of the source listing (Figure 8-19).

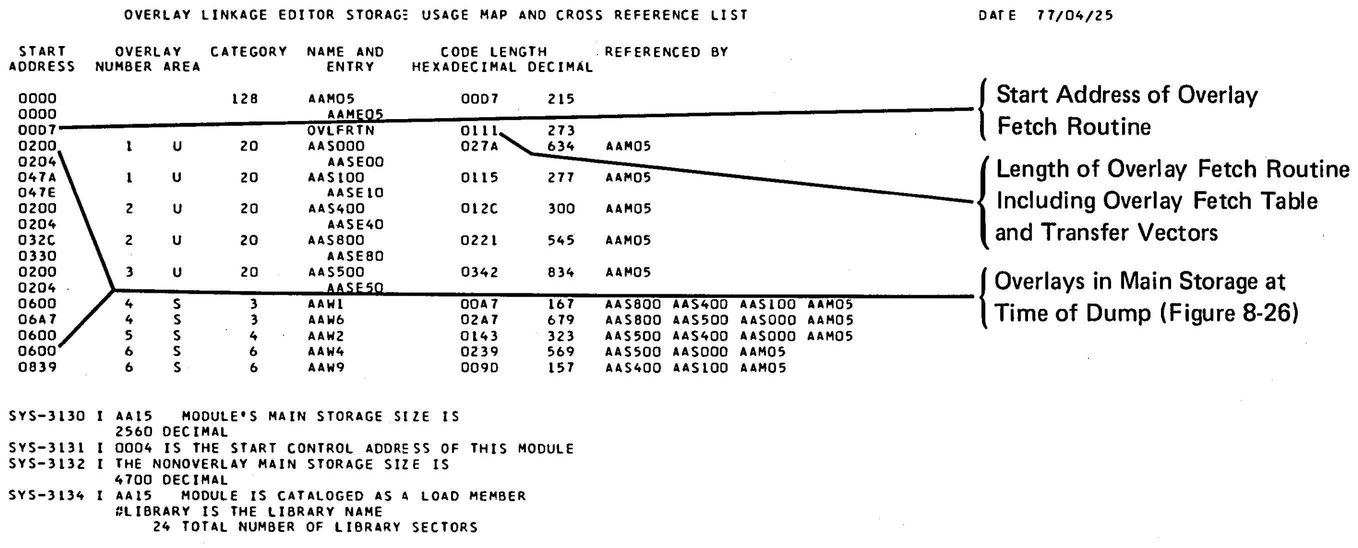
2. Locate the overlay fetch table in the storage dump (see Figure 8-20). The overlay fetch table is 120 bytes past the start address of the overlay fetch routine. It can be obtained by this hexadecimal formula: Address of overlay fetch routine +X'78' = overlay fetch table.

3. Mark off every 7-byte entry in the overlay fetch table until the last entry is reached. The last entry is X'FF' (see Figure 8-20).

4. Number each entry left to right, starting with 1. Each entry refers to an overlay (see Figure 8-20).

5. Look at the seventh byte in each entry. This is the flag byte. The first bit will be on for every overlay in storage at the time of the dump (see Figure 8-20).

6. Compare the numbers you gave the overlays in storage at the time of the dump with the number of the overlays in the storage usage map (Figure 8-19). This gives the names and addresses of the segments within the overlays that were in storage at the time of the dump (see Figure 8-20).



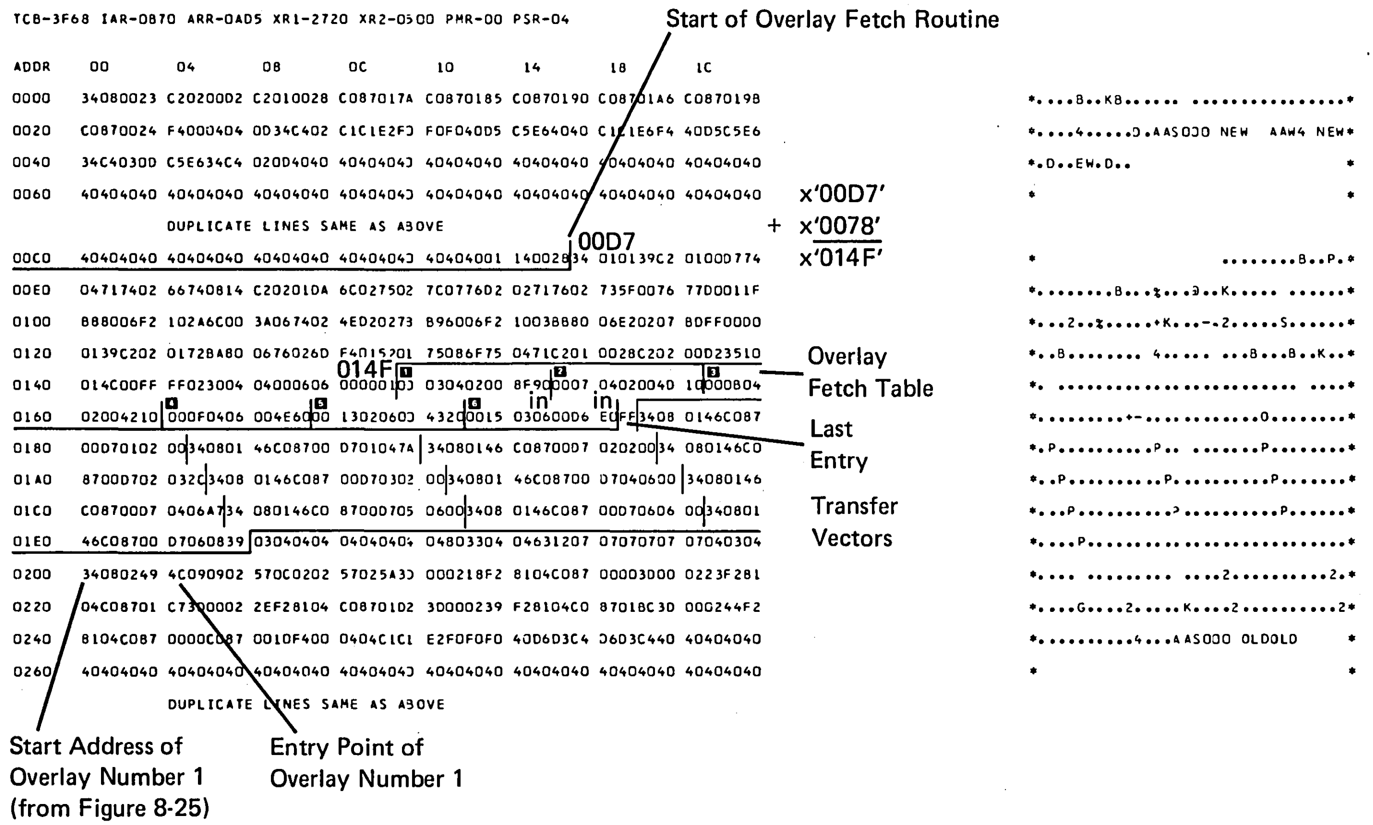
```
          OVERLAY LINKAGE EDITOR STORAG: USAGE MAP AND CROSS REFERENCE LIST                                  DATE  77/04/25

 START     OVERLAY   CATEGORY  NAME AND      CODE LENGTH      REFERENCED BY
ADDRESS  NUMBER AREA              ENTRY   HEXADECIMAL DECIMAL

 0000                   128     AAM05          0007      215
 0000                             AAME05
 0007                           OVLFRTN        0111      273
 0200       1     U      20     AAS000         027A      634   AAM05
 0204                             AASE00
 047A       1     U      20     AAS100         0115      277   AAM05
 047E                             AASE10
 0200       2     U      20     AAS400         012C      300   AAM05
 0204                             AASE40
 032C       2     U      20     AAS800         0221      545   AAM05
 0330                             AASE80
 0200       3     U      20     AAS500         0342      834   AAM05
 0204                             AASE50
 0600       4     S       3     AAW1           00A7      167   AAS800 AAS400 AAS100 AAM05
 06A7       4     S       3     AAW6           02A7      679   AAS800 AAS500 AAS000 AAM05
 0600       5     S       4     AAW2           0143      323   AAS500 AAS400 AAS000 AAM05
 0600       6     S       6     AAW4           0239      569   AAS500 AAS000 AAM05
 0839       6     S       6     AAW9           009D      157   AAS400 AAS100 AAM05

SYS-3130 I AA15   MODULE'S MAIN STORAGE SIZE IS
           2560 DECIMAL
SYS-3131 I 0004 IS THE START CONTROL ADDRESS OF THIS MODULE
SYS-3132 I THE NONOVERLAY MAIN STORAGE SIZE IS
           4700 DECIMAL
SYS-3134 I AA15   MODULE IS CATALOGED AS A LOAD MEMBER
           #LIBRARY IS THE LIBRARY NAME
               24 TOTAL NUMBER OF LIBRARY SECTORS
```

Start Address of Overlay Fetch Routine

Length of Overlay Fetch Routine Including Overlay Fetch Table and Transfer Vectors

Overlays in Main Storage at Time of Dump (Figure 8-26)

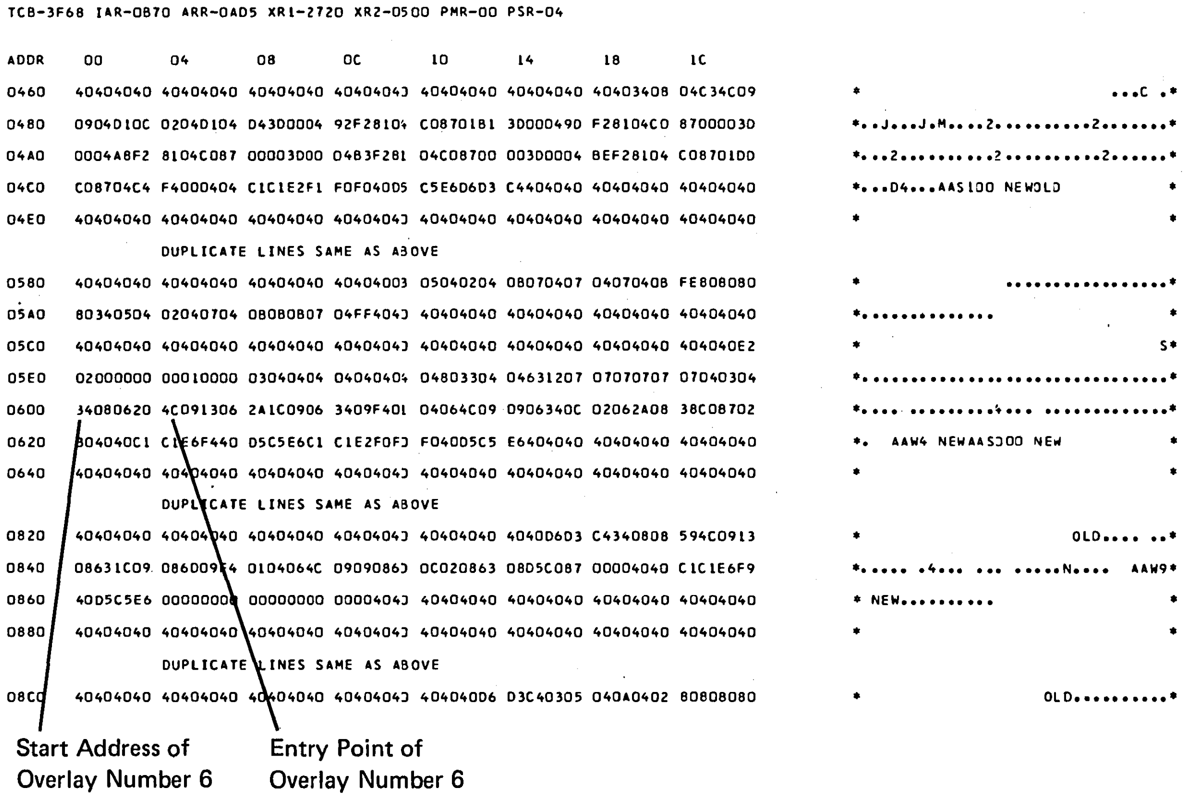
Figure 8-19. Storage Usage Map and Cross-Reference List



```
TCB-3F68 IAR-0870 ARR-0AD5 XR1-2720 XR2-0500 PMR-00 PSR-04

ADDR    00       04       08       0C       10       14       18       1C

0000    34080023 C2020002 C2010028 C087017A C0870185 C0870190 C08701A6 C0870198      *....B..KB...... ...............*
0020    C0870024 F4000404 0D34C402 C1C1E2F0 F0F040D5 C5E64040 C1C1E6F4 40D5C5E6      *....4......D.AASOOO NEW  AAW4 NEW*
0040    34C4030D C5E634C4 020D4040 40404040 40404040 40404040 40404040 40404040      *.D..EW.D..                     *
0060    40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040      *                               *
                 DUPLICATE LINES SAME AS ABOVE
00C0    40404040 40404040 40404040 40404040 40404001 14002834 010139C2 01000774      *                ........B..P.*
00E0    04717402 66740814 C20201DA 6C027502 7C0776D2 02717602 735F0076 77D0011F      *........B...%...@..K..... ......*
0100    888006F2 102A6C00 3A067402 4ED20273 B96006F2 10038880 06E20207 8DFF0000      *...2..%......+K...-.2.....S......*
0120    0139C202 01728A80 0676026D F4015201 75086F75 0471C201 0028C202 00D23510      *..B........ 4..... ...B...B..K..*
0140    014C00FF FF023004 04000606 00000100 03040200 8F900007 04020040 10000804      *. ......................... ....*
0160    02004210 000F0406 004E6000 13020600 43200015 03060006 E0FF3408 0146C087      *.........+-............O........*
0180    00070102 00340801 46C08700 D701047A 34080146 C0870007 02020034 080146C0      *.P..........P.. .......P........*
01A0    8700D702 032C3408 0146C087 00D70302 00340801 46C08700 D7040600 34080146      *..P..........P..........P.......*
01C0    C08700D7 0406A734 080146C0 87000705 06003408 0146C087 00070606 00340801      *...P..........?..........P......*
01E0    46C08700 D7060839 03040404 04040404 04803304 04631207 07070707 07040304      *....P...........................*
0200    34080249 4C090902 570C0202 57025A30 000218F2 8104C087 00003000 0223F281      *.... ......... ....2..........2.*
0220    04C08701 C7300002 2EF28104 C08701D2 30000239 F28104C0 87018C30 000244F2      *....G....2.....K....2..........2*
0240    8104C087 0000C087 0010F400 0404C1C1 E2F0F0F0 40D6D3C4 D6D3C440 40404040      *...........4....AASOOO OLDOLD   *
0260    40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040      *                               *
                 DUPLICATE LINES SAME AS ABOVE
```

Start of Overlay Fetch Routine
00D7
x'00D7'
+ x'0078'
x'014F'
014F
1
2
3
4
5
6
in
in
Overlay Fetch Table
Last Entry
Transfer Vectors
Start Address of Overlay Number 1 (from Figure 8-25)
Entry Point of Overlay Number 1

Figure 8-20 (Part 1 of 3). Sample Core Dump

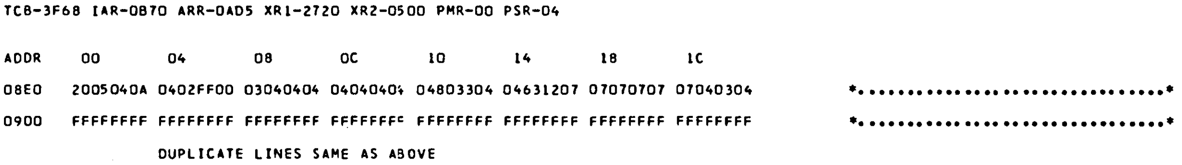


```
TCB-3F68 IAR-0870 ARR-0AD5 XR1-2720 XR2-0500 PMR-00 PSR-04

ADDR    00       04       08       0C       10       14       18       1C

0460    40404040 40404040 40404040 40404040 40404040 40404040 40403408 04C34C09      *                          ...C .*
0480    0904D10C 0204D104 D43D0004 92F28104 C0870181 3D000490 F28104C0 8700003D      *..J...J.M....2..........2.......*
04A0    0004A8F2 8104C087 00003D00 04B3F281 04C08700 003D0004 BEF28104 C08701DD      *...2..........2..........2......*
04C0    C08704C4 F4000404 C1C1E2F1 F0F04005 C5E6D6D3 C4404040 40404040 40404040      *...D4...AAS100 NEWOLD           *
04E0    40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040      *                                *
                 DUPLICATE LINES SAME AS ABOVE
0580    40404040 40404040 40404040 40404003 05040204 08070407 04070408 FE808080      *               .................*
05A0    80340504 02040704 08080807 04FF4040 40404040 40404040 40404040 40404040      *..............                  *
05C0    40404040 40404040 40404040 40404040 40404040 40404040 40404040 404040E2      *                               S*
05E0    02000000 00010000 03040404 04040404 04803304 04631207 07070707 07040304      *................................*
0600    34080620 4C091306 2A1C0906 3409F401 04064C09 0906340C 02062A08 38C08702      *.... .........4... .............*
0620    804040C1 C1E6F440 D5C5E6C1 C1E2F0F0 F04005C5 E6404040 40404040 40404040      *.  AAW4 NEWAAS000 NEW           *
0640    40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040      *                                *
                 DUPLICATE LINES SAME AS ABOVE
0820    40404040 40404040 40404040 40404040 40404040 4040D6D3 C4340808 594C0913      *                      OLD.... ..*
0840    08631C09 086009F4 0104064C 09090860 0C020863 08D5C087 00004040 C1C1E6F9      *..... .4... ... .....N....  AAW9*
0860    40D5C5E6 00000000 00000000 00004040 40404040 40404040 40404040 40404040      * NEW..........                  *
0880    40404040 40404040 40404040 40404040 40404040 40404040 40404040 40404040      *                                *
                 DUPLICATE LINES SAME AS ABOVE
08C0    40404040 40404040 40404040 40404040 40404006 D3C40305 040A0402 80808080      *                   OLD..........*
```

Start Address of
Overlay Number 6

Entry Point of
Overlay Number 6

Figure 8-20 (Part 2 of 3). Sample Core Dump

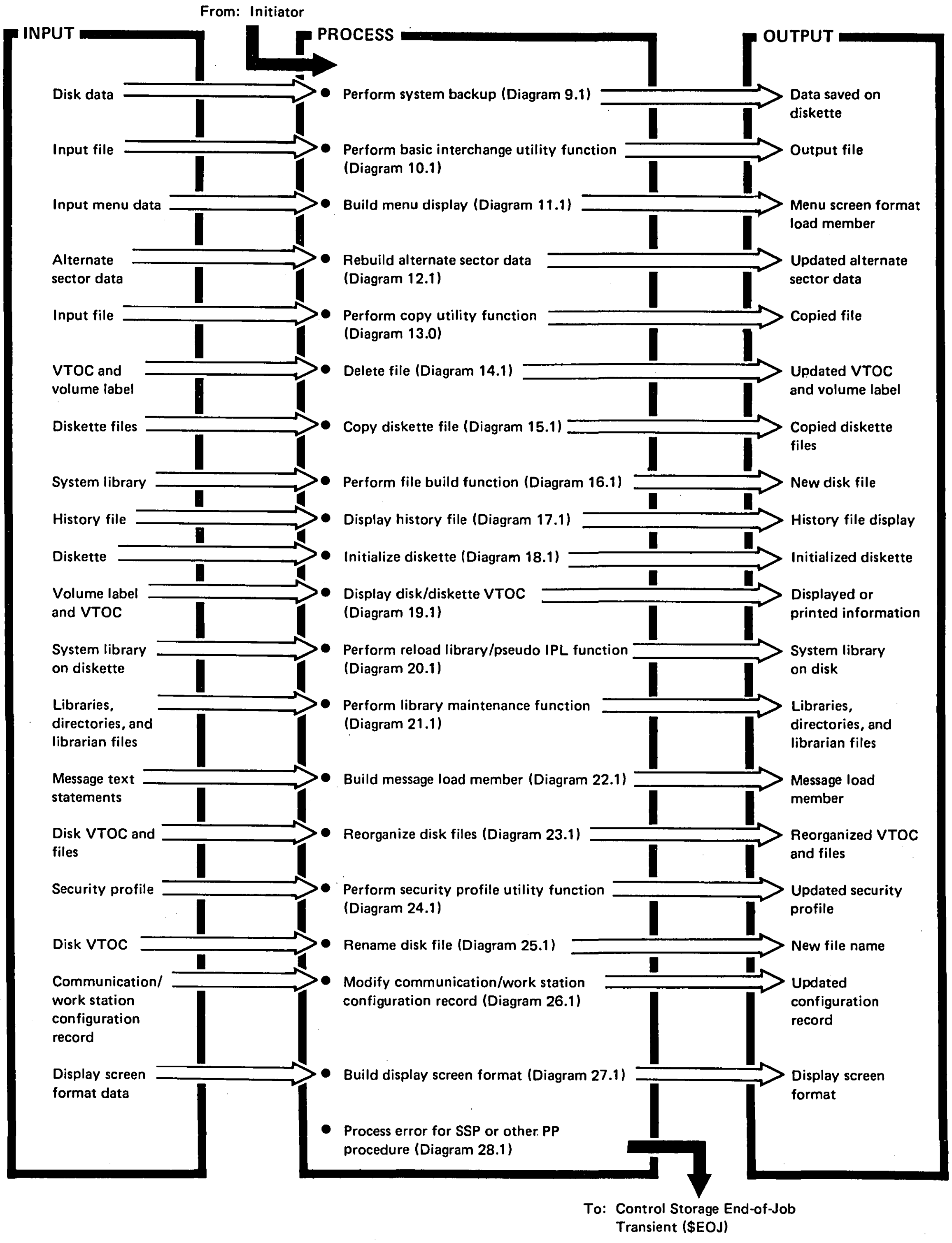


```
TCB-3F68 IAR-0870 ARR-0AD5 XR1-2720 XR2-0500 PMR-00 PSR-04

ADDR    00       04       08       0C       10       14       18       1C

08E0    2005040A 0402FF00 03040404 04040404 04803304 04631207 07070707 07040304      *................................*
0900    FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF      *................................*
                 DUPLICATE LINES SAME AS ABOVE
```

Figure 8-20 (Part 3 of 3). Sample Core Dump

Part 4. System Utility Programs

System utility programs are supplied by IBM as part of the SSP. When a system utility program is run, OCL statements identify the program and supply to the SSP any information that it requires about the program. Diagram 9.0 shows the functions performed by the system utility programs.



From: Initiator

INPUT	PROCESS	OUTPUT
Disk data	● Perform system backup (Diagram 9.1)	Data saved on diskette
Input file	● Perform basic interchange utility function (Diagram 10.1)	Output file
Input menu data	● Build menu display (Diagram 11.1)	Menu screen format load member
Alternate sector data	● Rebuild alternate sector data (Diagram 12.1)	Updated alternate sector data
Input file	● Perform copy utility function (Diagram 13.0)	Copied file
VTOC and volume label	● Delete file (Diagram 14.1)	Updated VTOC and volume label
Diskette files	● Copy diskette file (Diagram 15.1)	Copied diskette files
System library	● Perform file build function (Diagram 16.1)	New disk file
History file	● Display history file (Diagram 17.1)	History file display
Diskette	● Initialize diskette (Diagram 18.1)	Initialized diskette
Volume label and VTOC	● Display disk/diskette VTOC (Diagram 19.1)	Displayed or printed information
System library on diskette	● Perform reload library/pseudo IPL function (Diagram 20.1)	System library on disk
Libraries, directories, and librarian files	● Perform library maintenance function (Diagram 21.1)	Libraries, directories, and librarian files
Message text statements	● Build message load member (Diagram 22.1)	Message load member
Disk VTOC and files	● Reorganize disk files (Diagram 23.1)	Reorganized VTOC and files
Security profile	● Perform security profile utility function (Diagram 24.1)	Updated security profile
Disk VTOC	● Rename disk file (Diagram 25.1)	New file name
Communication/ work station configuration record	● Modify communication/work station configuration record (Diagram 26.1)	Updated configuration record
Display screen format data	● Build display screen format (Diagram 27.1)	Display screen format
	● Process error for SSP or other PP procedure (Diagram 28.1)	

To: Control Storage End-of-Job Transient ($EOJ)

Diagram 9.0. Overview of System Utility Program

Chapter 9. Backup Library Utility ($BACK)

Introduction

The backup library utility ($BACK) copies the entire system library (#LIBRARY) to one or more diskettes. When the library is copied to the diskettes, library members are shifted to remove gaps (unused space between members is collected at the end of the library).

The compressed library on diskette is in the following order:

- Reload IPL program ($LOADI) — 30 sectors
- Reload format load module (##FLOD) — 5 sectors
- $LOADI heading messages — 4 sectors
- Nucleus initialization program ($MSNIP) — 6 sectors
- System configuration records (2) — 2 sectors
- Library control sector (LCS) — 1 sector
- Reload communication area (at X'C0' into LCS)
- System library directory (addresses updated)
- System library members (compressed)

The backup library utility is called by the BACKUP procedure or appropriate OCL statements. (See *System Support Reference Manual* for information about calling $BACK.)

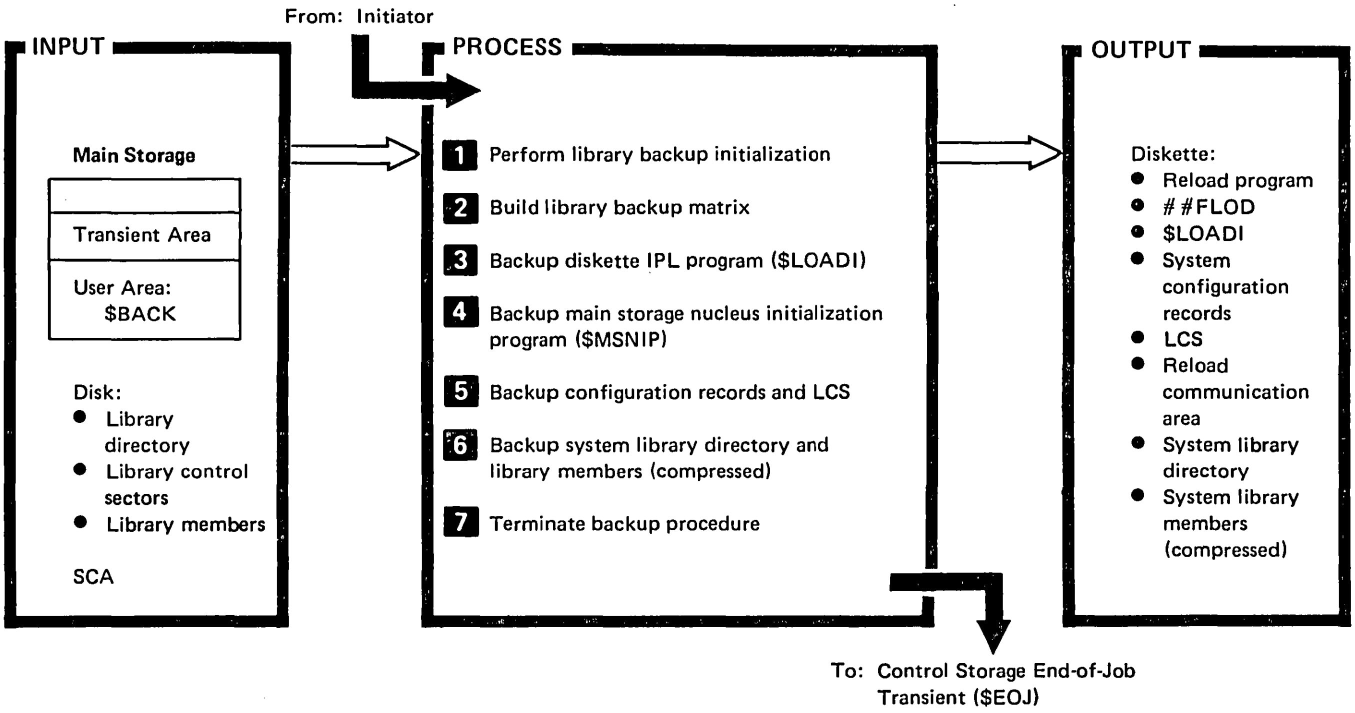
The main functions involved in executing $BACK are:

- Call the library directory compactor ($MACMP) to compress the library member directory.
- Allocate and open the backup file on diskette.
- Create strings of bits in main storage (matrix) representing library members as they appear on disk.
- Copy the reload library utility load module ($LOADI) from disk to diskette.
- Copy reload screen format load module (##FLOD) and $LOADI level 1 messages from disk to diskette.
- Copy main storage nucleus initialization program ($MSNIP) from disk to diskette.
- Copy the two configuration records and the library control sector (LCS) from disk to diskette.
- Update the library directory entries' sector addresses to reflect the compressed library and copy the library directory to diskette.
- Copy the library members from disk to diskette using the matrix to eliminate holes in the library.
- Close the diskette backup file.
- Return to the system by way of the end-of-job transient ($EOJ).

$BACK requires 14K bytes of main storage, exclusive control of the #LIBRARY file, and must run dedicated.

Method of Operation

Diagram 9.1 shows the function of the backup library utility.



From: Initiator

INPUT

Main Storage

Transient Area

User Area:
$BACK

Disk:
- Library directory
- Library control sectors
- Library members

SCA

PROCESS

1 Perform library backup initialization

2 Build library backup matrix

3 Backup diskette IPL program ($LOADI)

4 Backup main storage nucleus initialization program ($MSNIP)

5 Backup configuration records and LCS

6 Backup system library directory and library members (compressed)

7 Terminate backup procedure

OUTPUT

Diskette:
- Reload program
- ##FLOD
- $LOADI
- System configuration records
- LCS
- Reload communication area
- System library directory
- System library members (compressed)

To: Control Storage End-of-Job Transient ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Move directory and library sector addresses from system library format 1 to disk IOB.	$BACK
Read system library control sector (LCS).	Disk IOS
Compact system library directory and update in-core LCS.	$MACMP
▶ Calculate directory and library space now required (in sectors).	$BACK ◀
Verify and diagnose backup diskette.	#CAML
Return.	$BACK
Open diskette DTF.	#DMOP
Move file creation date from format 1 to reload-communication area.	$BACK
Move file name from format 1 to assign/free area.	
Set block size to track size calculated at open time.	
Calculate physical buffer end (CKPHYEND).	
Set maximum buffer sector count (CKBUFREC).	
Set dynamic buffer address (CKBUFFR@).	

Diagram 9.1 (Part 1 of 3). Perform System Backup ($BACK)

DESCRIPTION	MODULE/ ROUTINE
2 Set library matrix (4K) and hole table (2K) to zeros.	$BACK
Build library matrix from directory data:	
● Read directory sectors from disk.	Disk IOS
● Turn matrix bit on for each library sector allocated (matrix starts at X'3FFF' and goes backwards to X'3000').	$BACK
If any SSP modules deleted (SSP decreased in size) pass old SSP size to reload communications area (CKSCPSAV).	
3 Find $LOADI address.	
If $LOADI not found, issue error message and exit.	#CLXS
Copy reload program ($LOADI) to diskette:	$BACK
● Read 30 sectors of data into buffer.	Disk IOS
● Write data to diskette as buffer becomes full.	#DRSM
▶ Find reload screen formats (# #FLOD) location.	$BACK ◀
If # #FLOD not found, issue error message and exit.	#CLXS
Copy # #FLOD to diskette:	$BACK
● Read data from disk into buffer.	Disk IOS
● Write data to diskette as buffer becomes full.	#DRSM
● Return.	$BACK
Read heading messages into I/O buffer. (Sixteen messages, MICs 3900 through 3915, are loaded.)	#MGRET
Return.	$BACK
If any heading messages not found, issue error message and exit.	#CLXS
Write data to diskette as buffer becomes full.	#DRSM
4 Find main storage nucleus initialization program ($MSNIP).	$BACK
If $MSNIP not found, issue error message and exit.	#CLSG
Copy $MSNIP to diskette:	$BACK
● Read data into I/O buffer (buffer will not be filled for any buffer size).	Disk IOS
● Write data to diskette.	#DRSM
Initialize hole table for directory/library copy:	$BACK
● Scan library backup matrix (built in **2**) to find holes.	
● Set bit and hole values in hole table (to be used for directory backup).	
5 Set up disk IOB to read two configuration records.	
Read configuration records into I/O buffer.	Disk IOS
Return.	$BACK
Write configuration records to diskette.	#DRSM
Set up disk IOB to read library control sector (LCS).	$BACK
Read LCS into I/O buffer (1 sector).	Disk IOS
Return.	$BACK
Write LCS to diskette.	#DRSM

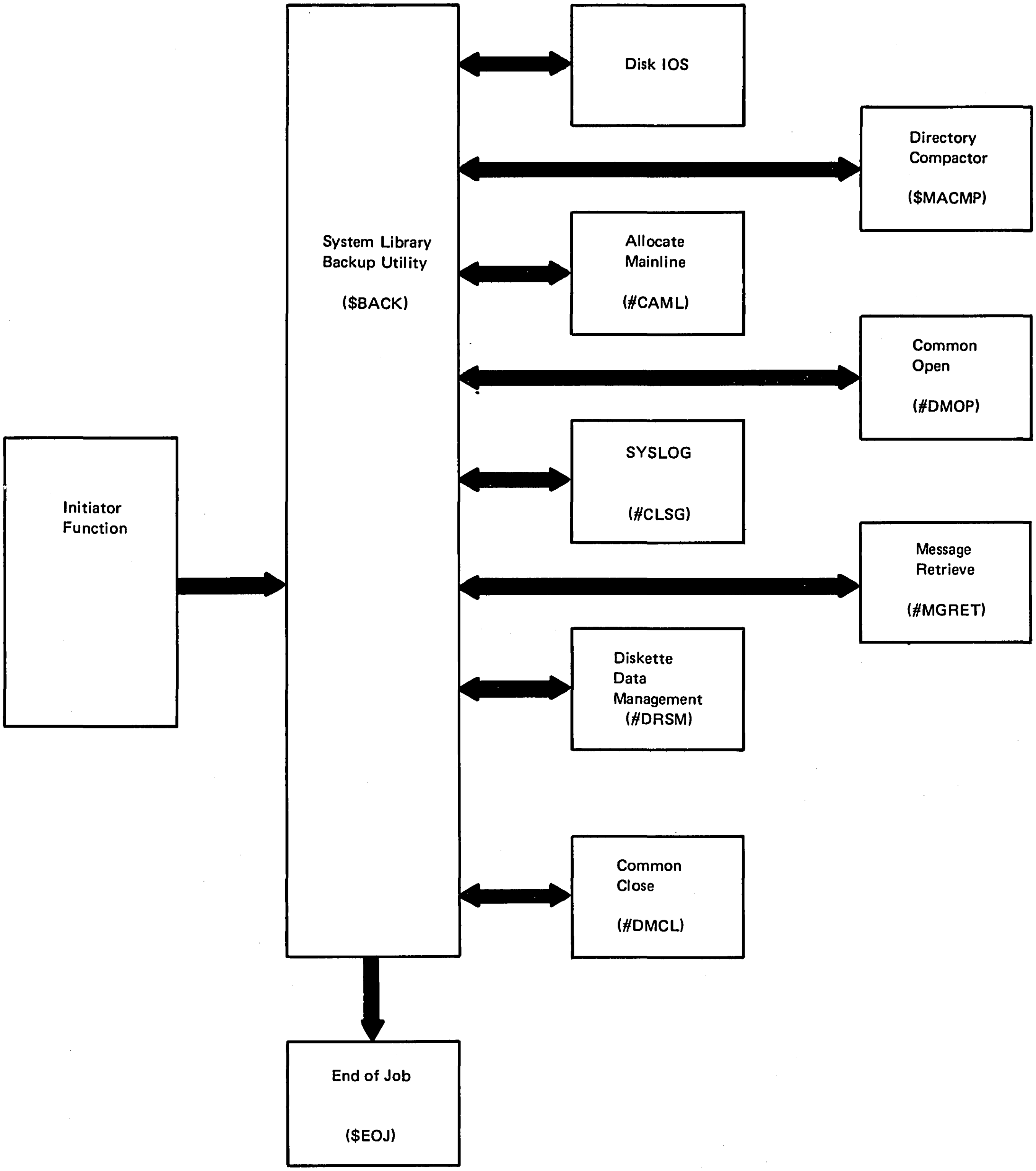
Diagram 9.1 (Part 2 of 3). Perform System Backup ($BACK)

DESCRIPTION	MODULE/ ROUTINE
6 Set up disk IOB to read system library directory.	$BACK
Read library directory into I/O buffer.	Disk IOS
Update hole table.	$BACK
Write library directory to diskette.	#DRSM
Scan library backup matrix until hole is found (number of bits scanned is number of sectors to transfer to diskette). Start scan again with next allocated sector (creates string of sectors to be transferred to diskette).	$BACK
Use string just built to read library sectors into I/O buffer (buffer contains compressed library sectors — no holes).	Disk IOS
Return.	$BACK
Write library sectors to diskette as buffer becomes full.	#DRSM
Return.	$BACK
7 Close diskette DTF.	#DMCL
Pass control to control storage end-of-job transient ($EOJ).	$BACK

Diagram 9.1 (Part 3 of 3). Perform System Backup ($BACK)

Program Organization

Figure 9-1 shows the control flow for the backup library utility.



Disk IOS
Directory Compactor ($MACMP)
System Library Backup Utility ($BACK)
Allocate Mainline (#CAML)
Common Open (#DMOP)
SYSLOG (#CLSG)
Initiator Function
Message Retrieve (#MGRET)
Diskette Data Management (#DRSM)
Common Close (#DMCL)
End of Job ($EOJ)

Figure 9-1. System Library Backup Utility Control Flow ($BACK)

Chapter 10. Basic Exchange Utility ($BICR)

Introduction

The basic exchange utility ($BICR) provides a way to convert a disk file to a basic exchange file on diskette, to convert a diskette basic exchange file to a sequential or indexed disk file, to add a basic exchange file to a sequential disk file, or display a diskette basic exchange file onto the SYSLIST device. All diskette files that provide input to $BICR must be in basic exchange format. All diskette files created by $BICR are also in basic exchange format. Figure 10-1 shows the diskette basic exchange format.

The utility program consists of a mainline phase ($BICR), a disk-to-diskette copy phase ($BICFI), a diskette-to-disk copy phase ($BICIF), a basic exchange file display phase ($BICDI), and a syntax specification module ($BITAB). The program resides in the system library.

The basic exchange utility is called by the TRANSFER procedure or appropriate OCL statements. (See the *System Support Reference Manual* for more information about calling and executing $BICR.)

The basic exchange utility requires 14K bytes of main storage for program execution.

Figure 10-1 shows the diskette basic exchange format.

Sector	Use	Initialized To
01	Reserved	
02	Reserved	
03	Reserved	
04	Reserved	
05	Positions 1 through 13 are used to record the identity of error tracks. Positions 1-5 = ERMAP (ERMAP identifies the sector as an error map.) Position 6 is reserved. Positions 7 and 8 hold the identification of the only bad track, or, if two tracks are bad, the identification of the lower-numbered bad track. If no bad tracks are identified, positions 7 and 8 = ƀ. Position 9 = ƀ if no bad track has been identified in this field. If at least one bad track has been identified, position = 0. Position 10 is reserved. Positions 11 and 12 contain the number of the higher-numbered bad track if two bad tracks have been identified; otherwise, these positions contain blanks. Position 13 = 0 if two bad tracks have been identified; otherwise, position 13 contains a blank. Positions 14 through 22 are reserved. Position 23 is a defect flag position, which is normally initialized to ƀ. A 3540 using IBM programming support puts a D in this position if the 3540 detects a surface defect in the data field of any sector of any track. All other positions on the sector are reserved.	1-5 = ERMAP 7-8 = ƀ 9 = ƀ 11-12 = ƀ 13 = ƀ 23 = ƀ

Figure 10-1 (Part 1 of 4). Diskette Basic Exchange Format

Sector	Use	Initialized To
06	Reserved	
07	This sector is called the volume label. Various fields in this sector identify the diskette, the diskette format, diskette owner identification, and whether or not the diskette uses standard labels. Positions 1-4 VOL1 (VOL1 identifies the sector as a volume label.) Positions 5-10 are called the volume ID field. This field can contain the data written on the permanent diskette label to identify the diskette. The ID consists of one to six numeric digits or letters. These characters must be left-justified in the field (that is, the first character must be in position 5 of the sector), and any unused positions in the field to the right of the ID data must contain space characters (blanks). No blanks are allowed between digits or letters in this field. Position 11 is the volume accessibility field. A blank (space character) in this field permits access to the disk. Any non-blank character in this field means that the disk is not accessible or has restricted access per system definition. Positions 12-37 are reserved. Positions 38-51 are called the owner ID field. This field is not used by all systems. Positions 52-76 are reserved. Positions 77-78 are called the record sequence field. This field holds the sector sequence code assigned to the tracks on this diskette. Position 79 is reserved. Position 80 is the standard label version field. The W Character indicates that IBM standard labels are used on the diskette. All diskettes used on the IBM 3540, 3741, 3742, and 3747 use standard labels, so this field should contain a W.	1-4 = VOL1 11 = ƀ 38-51 = ƀ 77-78 = ƀ 80 — W
08 through 26	These sectors are used to record the data set labels that define data sets (files of information) recorded on tracks 01 through 73 of the diskette.	

Position by Position Representation of Data on the Index Track in Any One Sector of Sectors 8-26.			Field in Unused New Diskette Contains this Data	
Field Name	Position	Purpose	Sector 08	Sectors 09 through 26
Label ID (Identifier)	1-4	Label identifier for system application; must be HDR1	HDR1	DDR1
	5	Reserved		
Data set name	6-13	User name for data set	DATAƀƀƀƀ	DATA09ƀƀ through DATA26ƀƀ
	14-22	Reserved		

Figure 10-1 (Part 2 of 4). Diskette Basic Exchange Format

Position by Position Representation on Data on the Index Track in Any one Sector of Sectors 8-26			Field in Unused New Diskette Contains this Data	
Field Name	Position	Purpose	Sector 08	Sectors 09 through 26
Block/Record length[†]	23-27	Tells the system how much of each 128-position sector contains actual data.	ƀƀ080	ƀƀ080
	28	Reserved		
Beginning of extent (BOE)	29-33	Identifies the address of the first sector of the data set. Positions 29 and 30 contain the track number, position 31 must be 0, positions 32 and 33 contain the sector number.	01001	74001
	34	Reserved		
End of extent (EOE)	35-39	Identifies the address of the last sector reserved for this data set, using the same format as BOE.	73026	73026
	40	Reserved		
Bypass indicator	41	If set to ƀ, the data set is intended for processing. If set to B, the data set is not intended for processing even though it resides on the disk. That is, a 3741 or 3742 user could store 3741 or 3742 programs on a diskette (identified with B in the label) as well as data (identified with ƀ in the label), and neither a 3747 nor a 3540 would read the programs. Also, a data set identified with a B in this position would not be transmitted by a 3741 Model 2 operating in teleprocessing transmit mode.	ƀ	ƀ
Data set security	42	Blank indicates data set not secured (can be accessed). Nonblank character (which can be written only by 3540) means restricted access. When set to nonblank, the volume accessibility indicator must also be set to nonblank. The data cannot be read by 3741, 3742, 3747, but may be read by 3540 with operator qualification. The data set cannot be written upon, and the volume accessibility indicator cannot be changed from nonblank by the 3741, 3742, or 3747, or by 3540 programming support.	ƀ	ƀ
Write protect	43	If this field contains a P, the data set can be read only. This field must be a blank to allow both reading and writing.	ƀ	ƀ
Interchange type indicator	44	Must be blank. A blank indicates the data set can be used for data interchange.	ƀ	ƀ

Figure 10-1 (Part 3 of 4). Diskette Basic Exchange Format

Position by Position Representation of Data on the Index Track in Any one Sector of Sectors 8-26.			Field in Unused New Diskette Contains this Data	
Field Name	Position	Purpose	Sector 08	Sectors 09 through 26
Multivolume indicator	45	A blank in this field indicates a data set is wholly contained on this diskette; a C indicates a data set is continued on another diskette; an L indicates the last diskette on which a continued data set resides.*	ƀ	ƀ
Volume sequence number**	46-47	Volume sequence specifies the sequence of volumes in a multivolume data set. The sequence must be consecutive, beginning with 01 (to a maximum of 99). Blanks indicate that volume sequence checking is not to be performed.	ƀƀ	ƀƀ
Creation date**	48-53	May be used to record the date the data set was created. The format is digits representing YYMMDD, where YY is low-order 2 digits of year, MM is 2-digit representation of month, and DD is 2-digit representation of day of the month.	ƀƀƀƀƀƀ	ƀƀƀƀƀƀ
	54-66	Reserved		
Expiration date**	67-72	May be used to contain the date that the data set (and its label) may be purged. The format is as specified for creation date.	ƀƀƀƀƀƀ	ƀƀƀƀƀƀ
Verify mark	73	This field must contain a V or a blank. V indicates the data set has been verified.	ƀ	ƀ
	74	Reserved		
End of data (EOD)	75-79	Identifies the address of the next unused sector within the data set extent.	01001	74001
	80	Reserved		

†Each sector contains one record. Standard interchange does not support blocking.

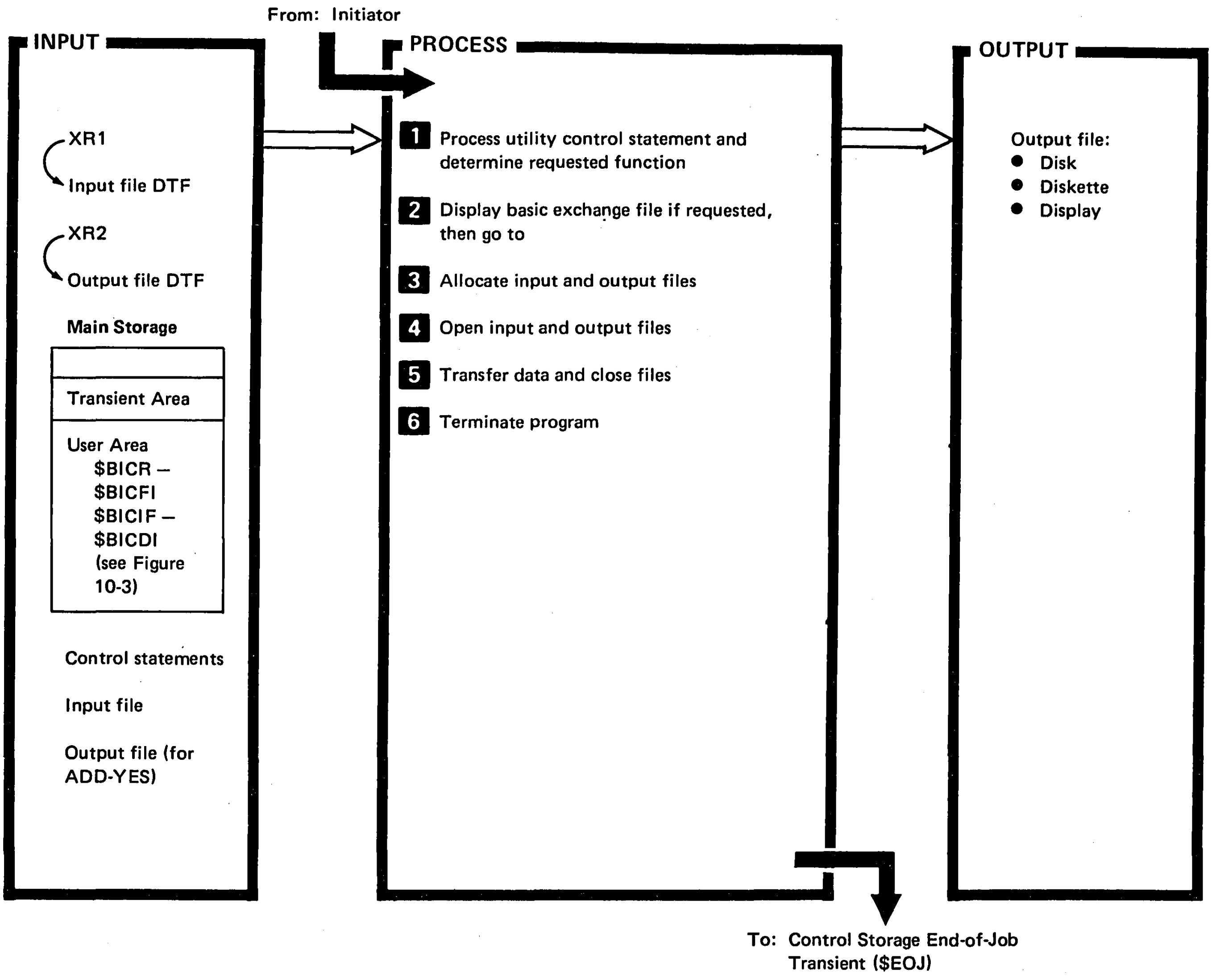
*When preparing data for conversion by the 3747, check the *IBM 3747 Data Converter Reference Manual*, GA21-9153, for the interpretation of the multivolume bytes.

**These fields are used only in conjunction with the 3540.

Figure 10-1 (Part 4 of 4). Diskette Basic Exchange Format

Method of Operation

Diagram 10.1 shows the function of the basic exchange utility.



From: Initiator

INPUT

XR1
Input file DTF

XR2
Output file DTF

Main Storage

Transient Area
User Area $BICR – $BICFI $BICIF – $BICDI (see Figure 10-3)

Control statements

Input file

Output file (for ADD-YES)

PROCESS

1 Process utility control statement and determine requested function

2 Display basic exchange file if requested, then go to

3 Allocate input and output files

4 Open input and output files

5 Transfer data and close files

6 Terminate program

OUTPUT

Output file:
- Disk
- Diskette
- Display

To: Control Storage End-of-Job
Transient ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Build parameter list to find syntax checker (#USYX).	$BICR
Locate #USYX.	#MASFN
Read and check syntax of control statements.	#USYX
If DISPLAY specified, set display indicator.	$BICR
A Find I/O device in device allocate table if diskette.	
If display indicator on: ● Load $BICDI into main storage following $BICR. ● Go to 2.	
Get format 1 for diskette COPYIN file.	#CSAF

Diagram 10.1 (Part 1 of 3). Perform Basic Exchange Utility Function ($BICR)

DESCRIPTION	MODULE/ ROUTINE
If format 1 found: ● Load $BICIF. ● Go to 3. If format 1 not found: ● Load $BICFI (copy diskette to disk). ● Go to 3.	$BICR
If diskette busy: ● Display message. ● If retry option (1) taken, go to 1 Ⓐ. ● If cancel option (3) taken, go to 6.	#CLXS
2 Get format 1 for diskette COPYIN file.	#CSAF
Return.	$BICDI
Allocate COPYIN file.	#CAML
Use main storage relocating loader (SVC 52) to load diskette data management (#DRDM). Also load display screen routine ($COCRT) and display service add ($FEKEY) or print interface ($COPRT) depending if SYSLIST goes to display screen or printer.	$BICDI
Open diskette DTF.	#DMOP
Return.	$BICDI
Read records from diskette file.	#DRDM
Return.	$BICDI
Output records to SYSLIST device (printer or display screen) as requested.	$COPRT or $COCRT
Continue processing until entire file displayed.	$BICDI
Issue error messages as necessary.	#CLSG
Close diskette file.	#DMCL
Go to 6 to terminate program.	$BICDI
3 If copy disk to diskette ($BICFI):	$BICFI
● Get format 1 for disk COPYIN file.	#CSAF
● Return.	$BICFI
● Allocate input file on disk. ● Allocate output file on diskette as a sequential basic exchange output file.	#CAML
If copy diskette to disk ($BICIF):	$BICIF
● Get format 1 for diskette COPYIN file.	#CSAF
● Return.	$BICIF
● Allocate input file on diskette.	#CAML
● Return.	$BICIF
● Allocate output file on disk as follows: – If ADD-YES, allocate output file as sequential input file to cause check of output file's existence. – If ADD-NO, allocate as output file: a. If key information specified, allocate output file as indexed file. b. If no key information specified, allocate output file as sequential file. c. If COPYO file statement specified, do normal allocate using COPYO file statement. d. If no COPYO file statement specified, do special allocate using input file's label and size.	#CAS1 or #CAML

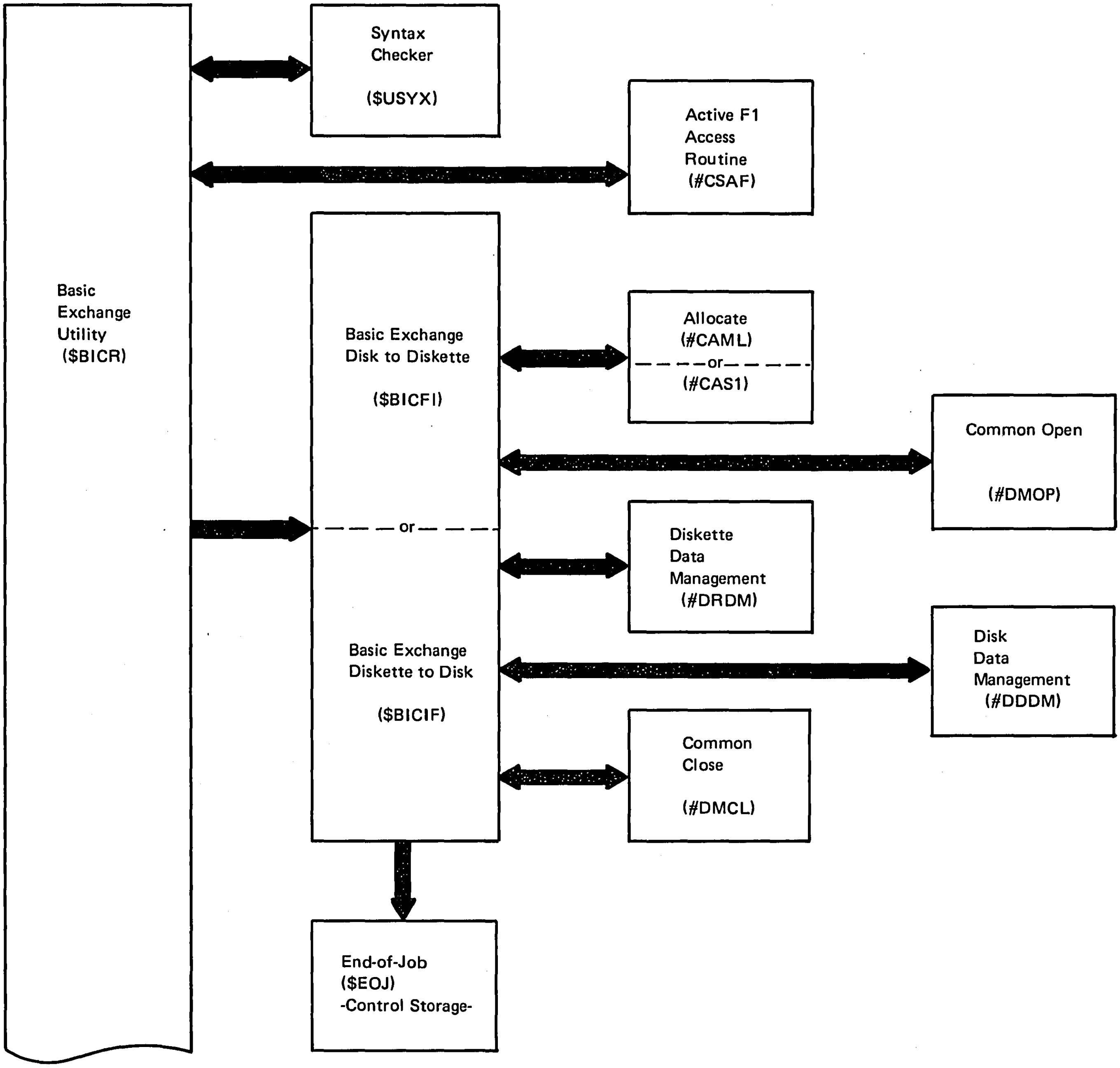
Diagram 10.1 (Part 2 of 3). Perform Basic Exchange Utility Function ($BICR)

DESCRIPTION	MODULE/ ROUTINE
4 If copy disk to diskette ($BICFI):	$BICFI
● Perform dummy open on COPYIN DTF.	#DMOP
● Reset UPSI in COPYIN and COPYO DTS's. ● Load diskette data management (#DRDM) into main storage. ● Set up input DTF as: – Indexed sequential input. – Sequential input. – Direct input. ● If one-sided single-density diskette: – Set output Record length to 128. ● If two-sided double-density diskette. – Set output Record length to 256. ● Set up I/O area.	$BICFI
● Open both DTFs.	#DMOP
If copy diskette to disk ($BICIF):	$BICIF
● If ADD-YES, perform dummy open on output DTF.	#DMOP
● Reset UPSI in COPYIN and COPYO DTFs. ● Load diskette data management (#DRDM) into main storage. ● If ADD-YES, set up output DTF to consecutive add. ● If key information given, set up output DTF for indexed output. ● Set all others to consecutive output. ● Set up I/O area.	$BICIF
● Open both DTFs.	#DMOP
5 If copy disk to diskette ($BICFI):	$BICFI
● Retrieve record from disk file (if record buffer exists, use move mode).	#DDDM
● Return.	$BICFI
● Write record to diskette output file.	#DRDM
If copy diskette to disk ($BICIF):	$BICIF
● Retrieve record from diskette file.	#DDDM
● Return.	$BICIF
● Write record to disk output file.	#DDDM
● Return.	$BICIF
Close input and output files.	#DMCL
6 Pass control to control storage end-of-job transient ($EOJ).	$BICR

Diagram 10.1 (Part 3 of 3). Perform Basic Exchange Utility Function ($BICR)

Program Organization

Figure 10-2 shows the control flow of the basic exchange utility. Figure 10-3 shows the main storage map for the utility.



Basic Exchange Utility ($BICR)
Syntax Checker ($USYX)
Active F1 Access Routine (#CSAF)
Basic Exchange Disk to Diskette ($BICFI)
or
Basic Exchange Diskette to Disk ($BICIF)
Allocate (#CAML)
or
(#CAS1)
Common Open (#DMOP)
Diskette Data Management (#DRDM)
Disk Data Management (#DDDM)
Common Close (#DMCL)
End-of-Job ($EOJ) -Control Storage-

Figure 10-2 (Part 1 of 2). Basic Exchange Utility Control Flow ($BICR)

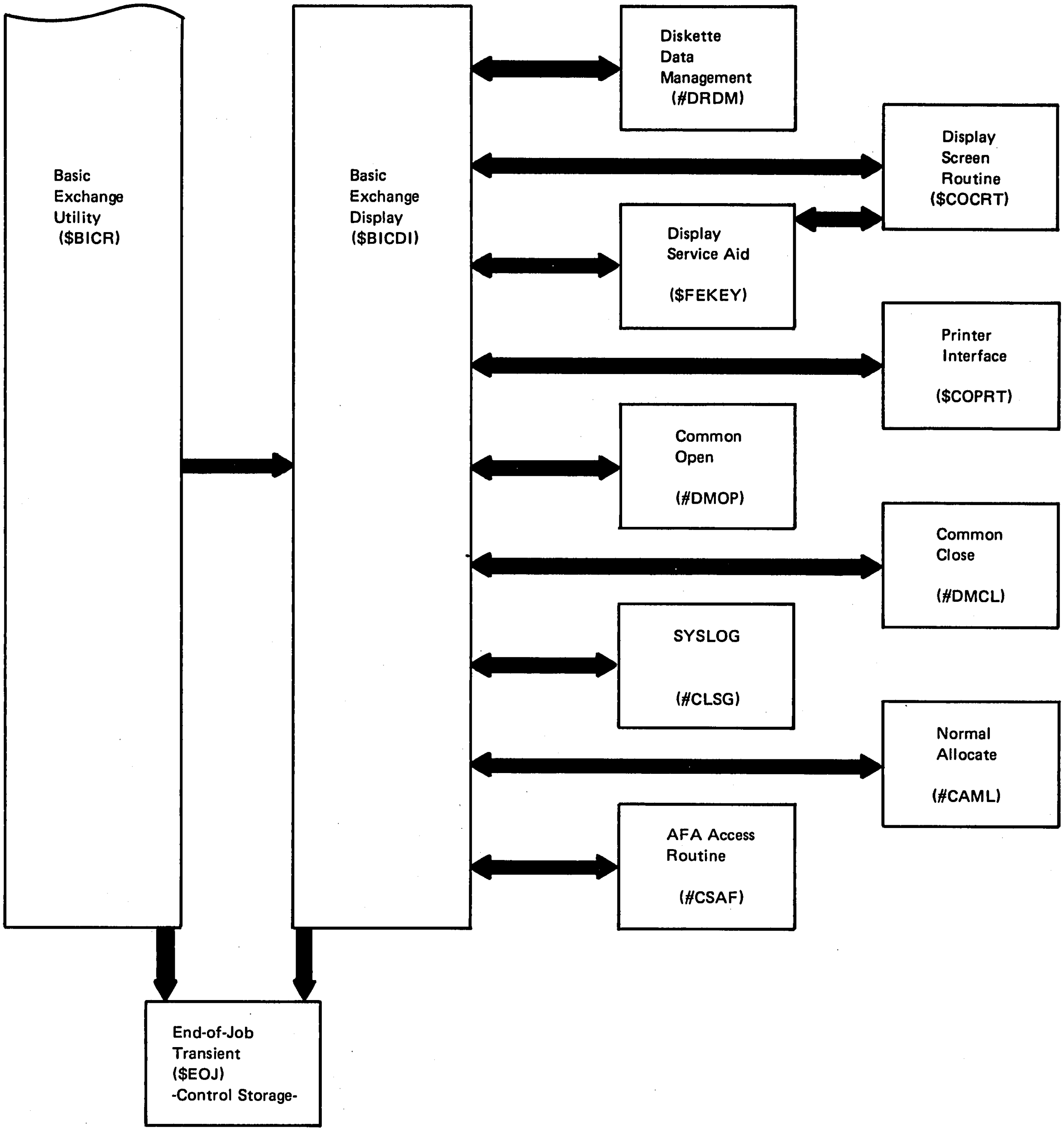

Basic Exchange Utility ($BICR)
Basic Exchange Display ($BICDI)
Diskette Data Management (#DRDM)
Display Screen Routine ($COCRT)
Display Service Aid ($FEKEY)
Printer Interface ($COPRT)
Common Open (#DMOP)
Common Close (#DMCL)
SYSLOG (#CLSG)
Normal Allocate (#CAML)
AFA Access Routine (#CSAF)
End-of-Job Transient ($EOJ) -Control Storage-

Figure 10-2 (Part 2 of 2). Basic Exchange Utility Control Flow ($BICR)

Data Areas

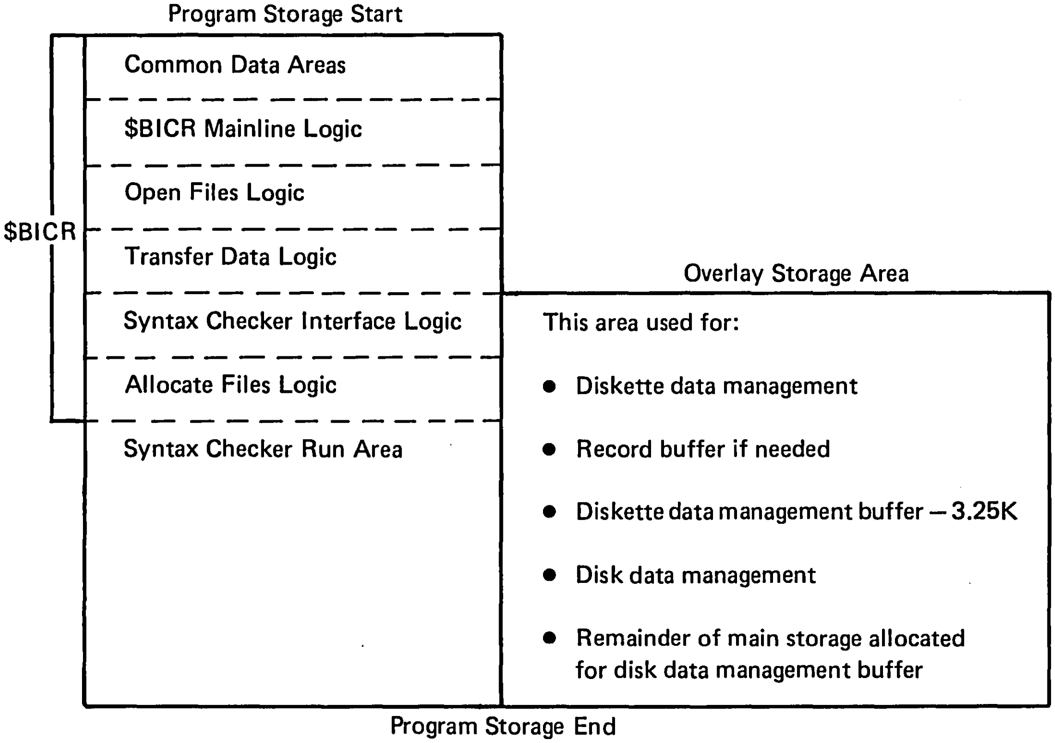


Figure 10-3. Main Storage Map for Basic Exchange Utility

BICCVLST VERB LIST

The BICCVLST verb list is a list of constants passed to the syntax checker (#USYX) each time #USYX is called. Figure 10-4 shows the format and contents of the verb list.

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes in Decimal	Description
0	BICCVLST	1	Transfer verb ID
1	–	1	End verb ID
2	–	1	Display
3	–	1	X'FF' end of verb list

Figure 10-4. BICCVLST Verb List

Chapter 11. Build Menu Utility ($BMENU)

Introduction

The build menu utility ($BMENU) provides a way to build a menu display. The menu display can be selected when signing on the system at a work station or by way of the MENU command. A menu gives the system user the convenience of submitting work by entering a menu item number as opposed to entering actual control commands, procedures, or OCL statements. The system user sees a description of the work to be performed without seeing the actual control command, procedure name, or OCL statement used as input to the system control program.

The purpose of the menu build utility is to build a screen format load member for the menu. When the system user accesses the menu at sign-on or by way of the MENU command, the menu contained in the screen format load member is displayed on the screen.

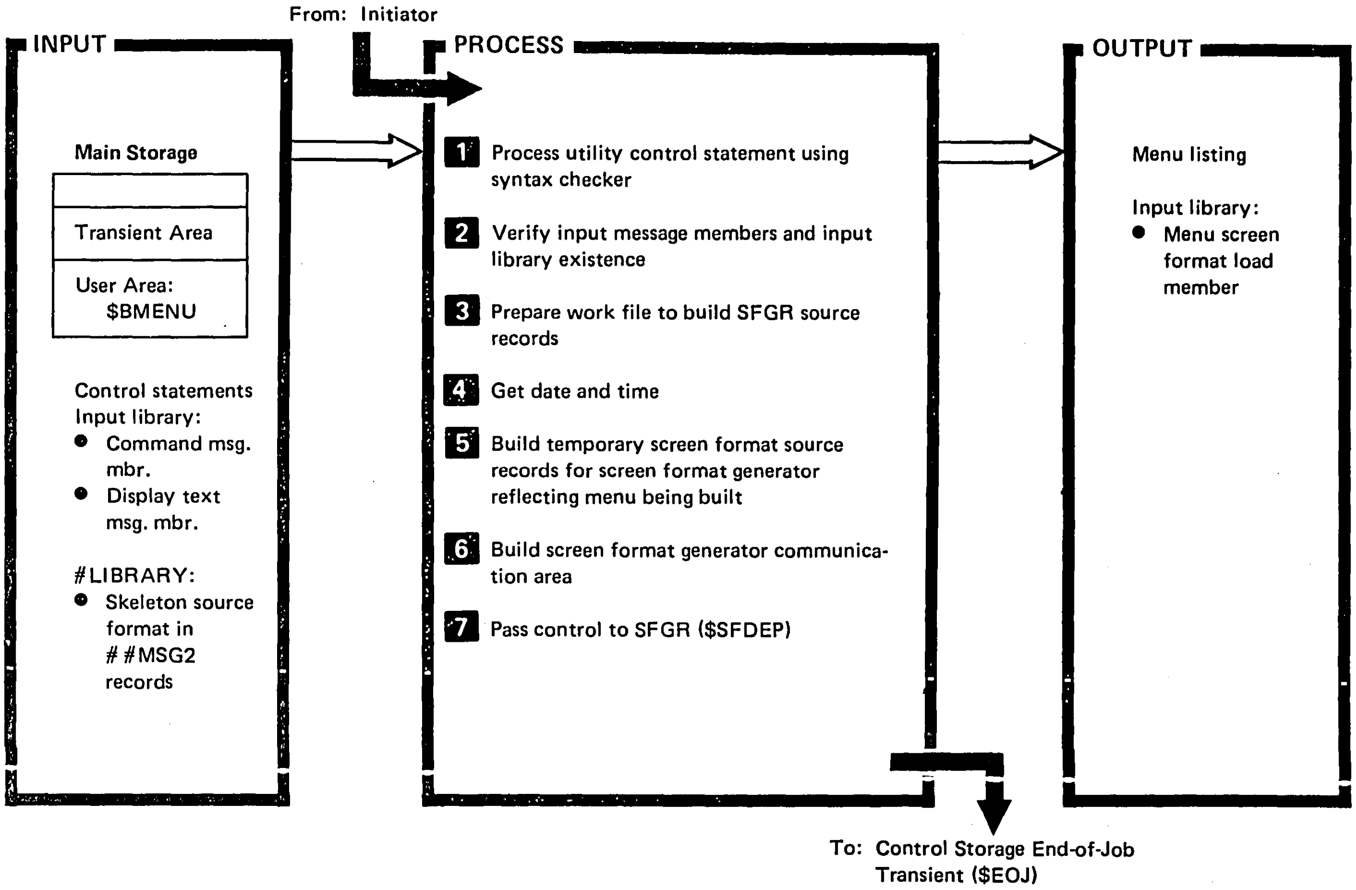
The $BMENU utility consists of the build menu phase ($BMENU), message retrieve (@MGRET) and addressing module (@BMAM) link-edited with $BMENU, and the build menu syntax checker specification module ($BMTB). The build menu program resides in the system library.

The build menu utility is called by the BLDMENU procedure or appropriate OCL statements. (See the *System Support Reference Manual* for more information about calling and executing $BMENU.)

The $BMENU utility requires 14K bytes of main storage for program execution.

Method of Operation

Diagram 11.1 shows the function of the build menu utility.



From: Initiator

INPUT

Main Storage

Transient Area

User Area: $BMENU

Control statements

Input library:

- Command msg. mbr.
- Display text msg. mbr.

#LIBRARY:

- Skeleton source format in ##MSG2 records

PROCESS

1 Process utility control statement using syntax checker

2 Verify input message members and input library existence

3 Prepare work file to build SFGR source records

4 Get date and time

5 Build temporary screen format source records for screen format generator reflecting menu being built

6 Build screen format generator communication area

7 Pass control to SFGR ($SFDEP)

OUTPUT

Menu listing

Input library:

- Menu screen format load member

To: Control Storage End-of-Job Transient ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Read and syntax check MENU control statement:	#USYX
Check INPMSG value validity:	$BMENU
● Issue message SYS-5756 if INPMSG less than 2 characters. ● Issue message SYS-5751 if INPMSG does not end with ##.	#CLSG
2 Find library specified by INLIB parameter.	#MAFLB
Return.	$BMENU
Find INPMSG member in input library.	#MASFN
Save library format 1 address. Issue message SYS-5752 if INLIB value not found or message SYS-5753 if INPMSG member not found.	$BMENU
Look for old menu member in input library.	#MASFN
Return.	$BMENU
Issue message SYS-5004 if menu exists but is not SFGR member. Issue message SYS-5005 if REPLACE-NO specified (or defaulted) and menu exists and is an SFGR member.	#CLXS

Diagram 11.1 (Part 1 of 3). Perform Build Menu Function ($BMENU)

DESCRIPTION	MODULE/ ROUTINE
Find MENMSG member in input library if MENMSG specified:	#MASFN
● Save library format 1 address.	$BMENU
● Issue message SYS-5754 if not found. ● Issue message SYS-5762 if MENMSG name is same as menu being built.	#CLXS
Read END control statement.	#USYX
3 Allocate work file for menu source build and issue message SYS-5755 if work file not allocated.	#CAS1
Return.	$BMENU
Open work file just allocated.	#DMOP
Build program1 and program2 message member pointers in job control block (JCB). 4 Get date and time by reading MIC 5400 and filling in date and time fields in # #MSG2.	$BMENU
5 Get skeleton source statements from # #MSG2 (in #LIBRARY) to build format and place statements in work file:	@MGRET
● Return.	$BMENU
▶ ● MIC 5755 – SFGR S record and source record for line 1. ● MIC 5756 – source records for line 1. ● MIC 5757 – source records for line 2 ● MIC 5758 – skeleton source records for lines 03-14.	#DDDM ◀
Build source records for item number lines (lines 03-14 on menu format): ● Build source records for item numbers 1 to 24 for specified line: – Find item number MIC in INPMSG member. – Find item number MIC in MENMSG member. – Build 30 byte item text description as follows: a. If item number MIC found in both INPMSG and MENMSG members, use first 30 bytes of MENMSG member MIC. b. If item MIC found in INPMSG member only, use first 30 bytes of INPMSG member MIC. c. If item MIC found in MENMSG member only, set item text to blanks. d. If item MIC missing in both message members, set item text to blanks. e. If item MIC found in INPMSG member is blank record, set item text to blanks. f. If item MIC found in INPMSG member is nonblank, and corresponding MENMSG memeber MIC is a blank record, use the first 30 bytes of INPMSG MIC.	$BMENU
– Write SFGR source records for this item number.	#DDDM
– Return	$BMENU
– Print messages if any.	#CLST
Build lines 21, 23, and 24 in menu format using skeleton source statements in # #MSG2: ● MIC 5759 – line 21 source records. ● MIC 5760 – line 21 source records. ● MIC 5761 – line 23 source records. ● MIC 5762 – line 24 source records.	$BMENU
Print last heading line for build menu listing if menu not built OK. Issue # #MSG2 MIC 5770; all INPMSG item MICs not found or contained blank records.	#CLST
Return.	$BMENU
6 Close work file.	#DMCL
Check if any item MICs in INPMSG message member:	$BMENU
● If no, issue message SYS-5757 with cancel option only.	#CLXS
● If yes, and all INPMSG MICs contain blank records, issue MIC 5750 with cancel-only option. ● If yes, and valid INPMSG MICs found, continue to build SFGR communication area.	$BMENU

Diagram 11.1 (Part 2 of 3). Perform Build Menu Function ($BMENU)

DESCRIPTION	MODULE/ ROUTINE
Move work file data sector count to SFGR communication area.	$BMENU
Move 256 bytes of zero to location X'C800' – X'C8FF'.	
Move SFGR communication area to X'C800'.	
Move X'FF' to X'C900' through X'CAFF'.	
7 Use main storage relocating loader to load SFGR Diagnose Errors and Print phase ($$FDFP).	
Note: The screen format generator uses the temporary source screen format member just built as input to build the menu screen format load member.	

Diagram 11.1 (Part 3 of 3). Perform Build Menu Function ($BMENU)

Program Organization

Figure 11-1 shows the control flow for the build menu utility.

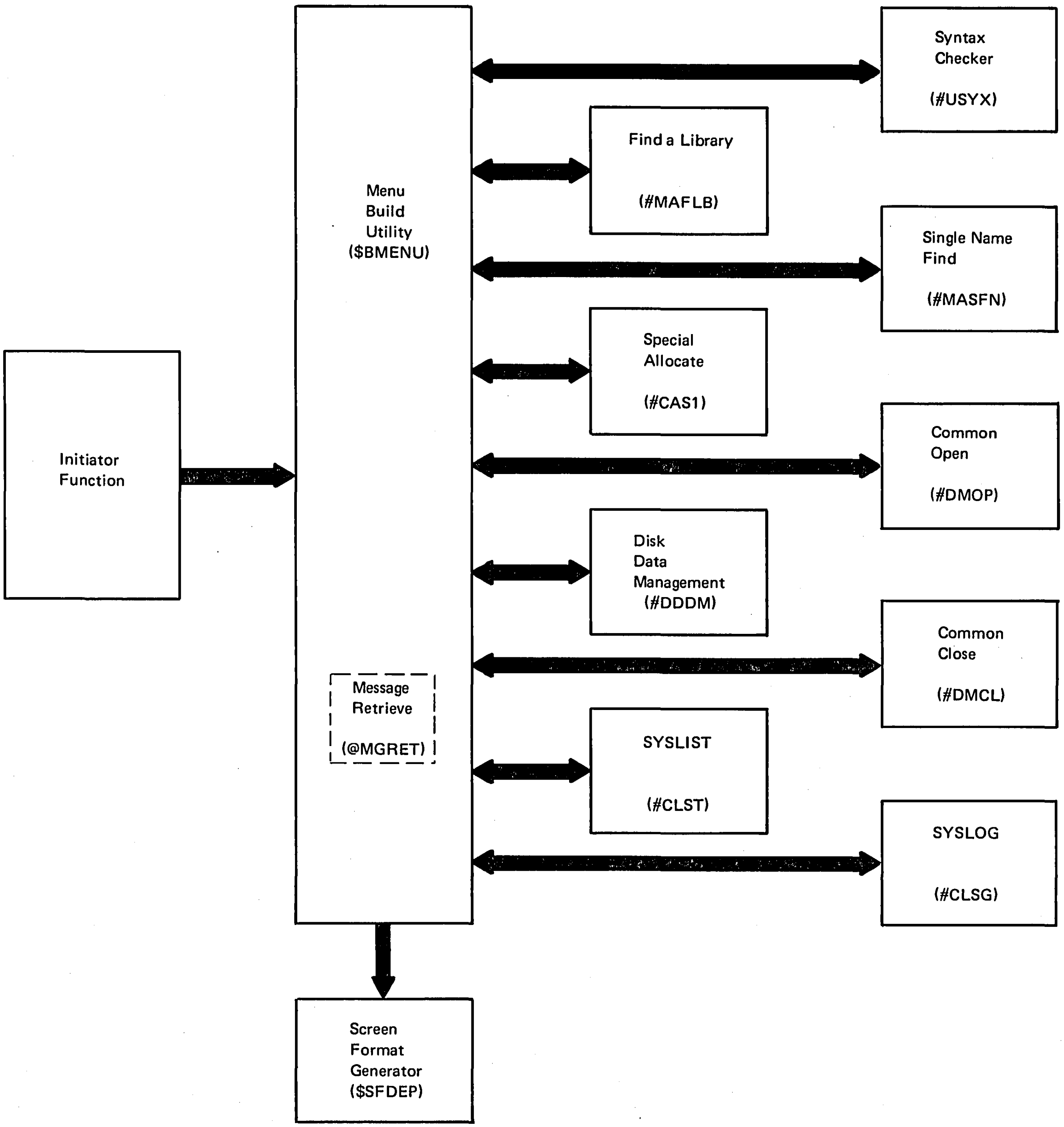


Figure 11-1. Build Menu Utility Control Flow ($BMENU)

Chapter 12. Alternate Sector Rebuild Utility ($BUILD)

Introduction

The alternate sector rebuild utility ($BUILD) allows the System/34 user to display and correct data on disk after a disk read error has occurred. When a disk read/write error occurs, the data is written to an alternate sector. Disk alternate sectors are sectors reserved for use in place of defective disk sectors.

$BUILD searches alternate disk sectors for data that is unreadable because of a read/write error. Each sector containing bad data is printed, along with the sector preceding and the sector following the bad sector.

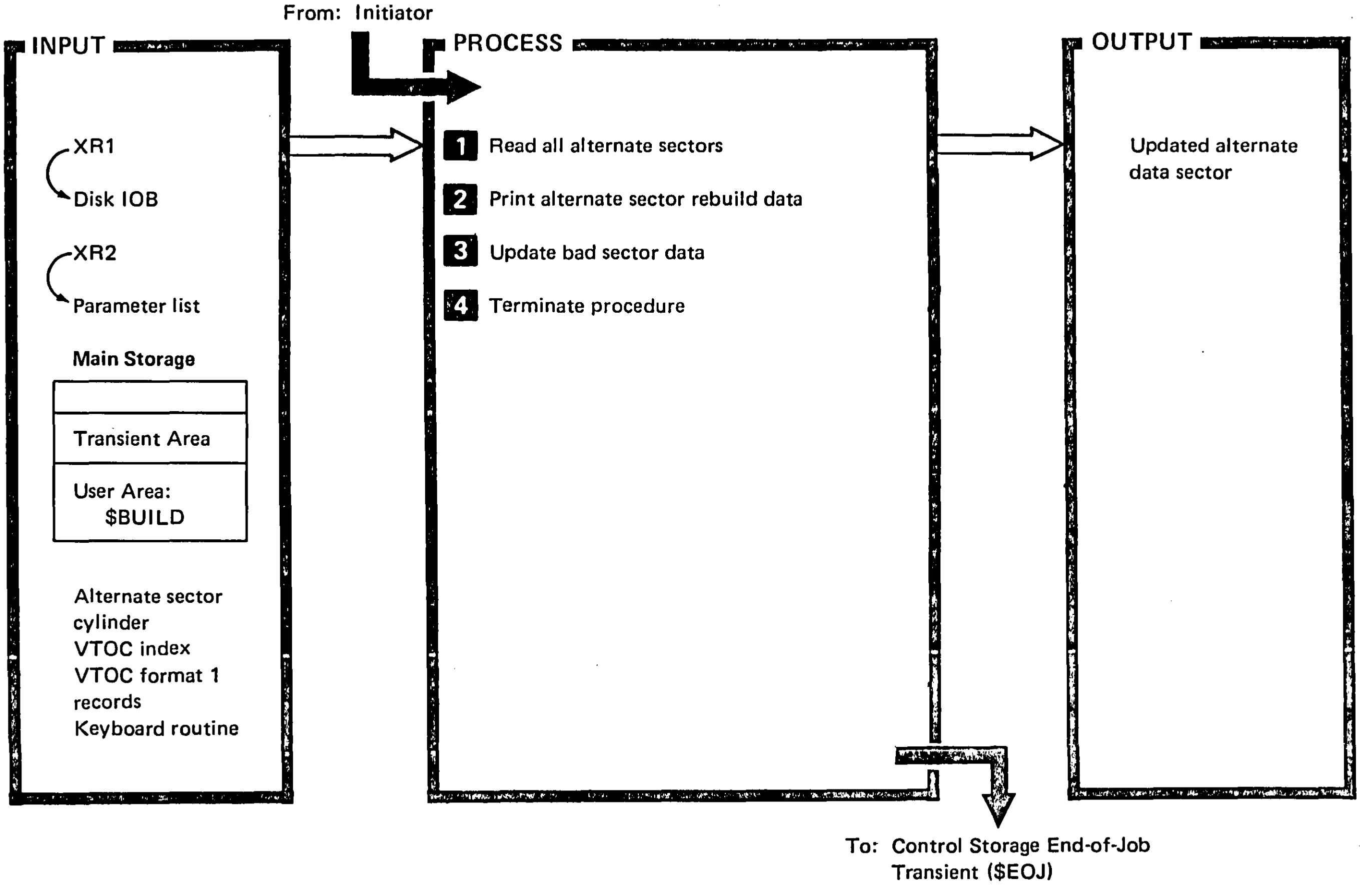
The sector containing bad data is then presented on the display screen where it can be corrected. (See the *System Support Reference Manual* for information about the data format and how the bad data can be corrected.)

The alternate sector rebuild utility is called by the BUILD procedure or appropriate OCL statements. $BUILD has no utility control statements.

$BUILD requires 14K bytes of main storage and must run dedicated until it finishes reading and writing alternate tracks.

Method of Operation

Diagram 12.1 shows the function of the alternate sector rebuild utility.



DESCRIPTION	MODULE/ ROUTINE
1 Initialize disk IOB to read alternate sector IDs on alternate sector tracks 4 and 5, cylinder 2/spindle 1 (also spindle 2 if available). Build bad data table:	$BUILD
● Read alternate sector IDs.	Disk IOS
● Test bad data flags looking for address of bad data. ● If bad data found, convert address to SS format and place address in 6 byte/entry bad data table (each entry contains SSS of bad primary and SSS of alternate). If no entries in bad data table, go to **4**.	$BUILD
Display operator paper form change warning message (MIC 4352).	#CLXS
Find CRT window display routine ($FEKEY).	#MASFN
Use main storage relocating loader (SVC-52) to load $FEKEY.	$BUILD
Load heading messages (MIC 1550-1558) into main storage.	#MGRET
Return.	$BUILD
A Read active format 1s to locate file containing bad sector.	#CSVF
If bad sector not in file, set name to NOFILE for print.	$BUILD

Diagram 12.1 (Part 1 of 2). Perform Alternate Sector Rebuild Function ($BUILD)

DESCRIPTION	MODULE/ ROUTINE
Obtain sector address.	$BUILD
If bad sector is first sector in file, set on no-sector-before indicator.	
If bad sector is last sector in file, set on no-sector-after indicator.	
If sector-before to process (BUSWITCH), read sector-before data.	Disk IOS
Read bad-data sector (set on no-ERP before read operation).	
If no sector-after to process, go to 2.	$BUILD
Read sector-after data.	Disk IOS
2 Convert sector-before, bad sector, and sector-after sector addresses to printable characters.	$BUILD
Print $BUILD heading line.	#CLST
If sector-before to be printed (BUSWITCH): ● Print filename and sector-before address. ● Print sector-before message (MIC 1552). ● Print character position line. ● Print actual data characters. ● Print zone and numeric portion of each character.	
Print bad sector data: ● Print filename and bad sector address. ● Print bad data message (MIC 1553). ● Print character position line. ● Print actual data characters. ● Print zone and numeric portion of each character.	
If sector-after to be printed (BUSWITCH): ● Print filename and sector-after address. ● Print sector-after message (MIC 1554). ● Print character position line. ● Print actual data. ● Print zone and numeric portion of each character. ● Print information message.	
3 Initialize $FEKEY parameter list.	$BUILD
Display bad data sector.	$FEKEY
Update bad data if desired.	
If Enter/Rec Adv key returned and not at end of bad-data table, return to 1 A.	$BUILD
If Roll key returned, return to 3.	
Write updated sector back to disk.	Disk IOS
Return.	$BUILD
4 If errors detected display appropriate error message with 3 option halt.	#CLXS
Pass control to control storage end-of-job transient ($EOJ).	$BUILD

Diagram 12.1 (Part 2 of 2). Perform Alternate Sector Rebuild Function ($BUILD)

Program Organization

Figure 12-1 shows the control flow for the alternate sector rebuild utility.

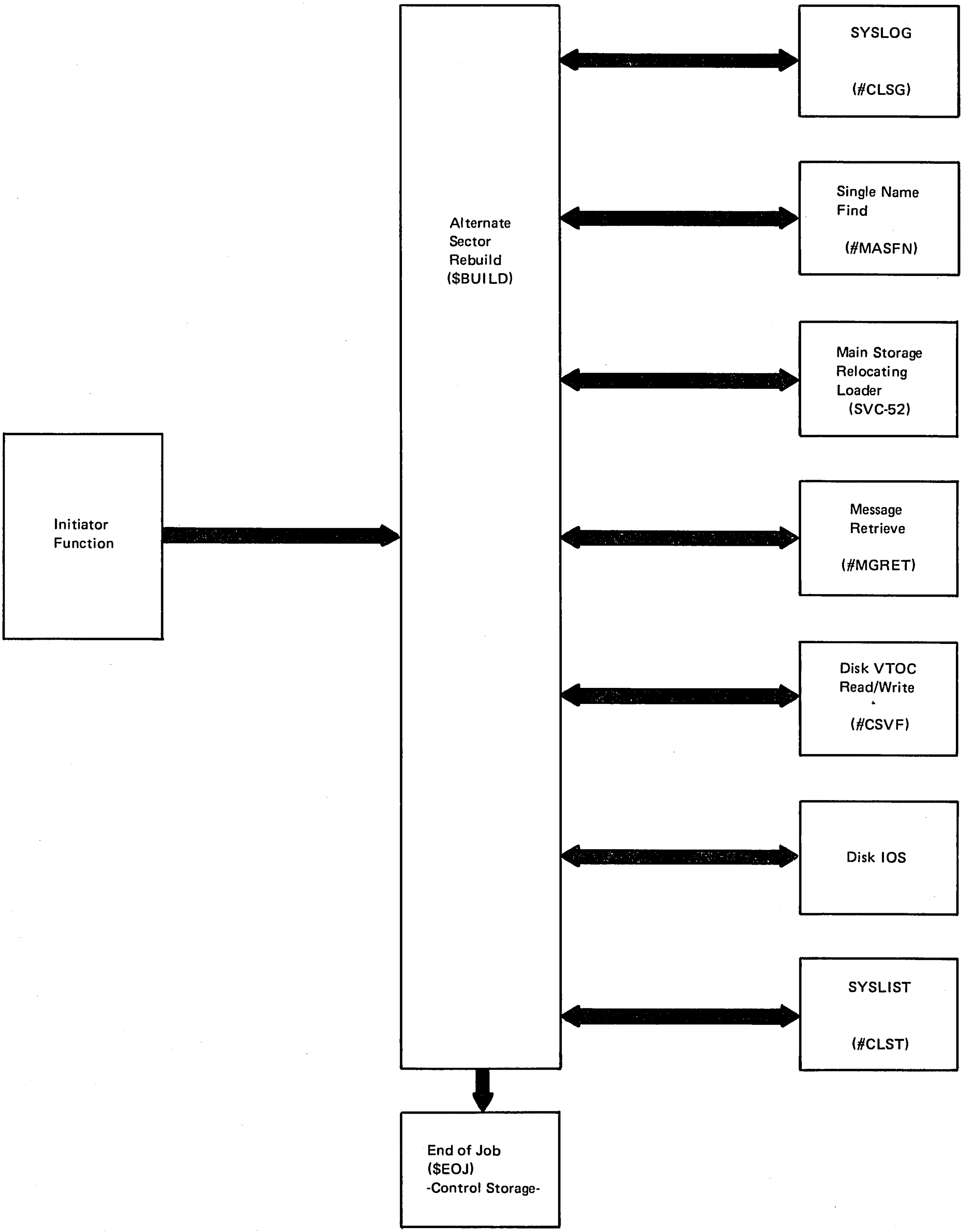


Figure 12-1. Alternate Sector Rebuild Utility Control Flow ($BUILD)

Chapter 13. Disk Copy/Display Utility ($COPY)

Introduction

The disk copy/display utility performs the following functions:

- Copies an entire file from the disk to diskette(s), from diskette(s) to the disk, or from disk to another location on the disk to:
  - Provide a duplicate of a file
  - Move a file to a larger disk area
- Deletes records from a file (selected records are omitted from the copy; the original remains unchanged).
- Copies a portion of a file. Selected records can be deleted from the copy.
- Copies all user disk data files (except user libraries), all disk data files with a specified file group identifier, or all disk data files with no group identifier to diskette(s) to create a backup copy of the files or to obtain more space on the disk. When all files are to be copied, $COPY must be requested from the system console, and no other jobs can be running.
- Restores previously copied files from diskette(s) to the disk.
- Copies an indexed file, and puts the records in key order (reorganize the file) to improve the performance, in some cases, of programs that use the file. Selected records can be deleted from the copy.
- Adds a disk file to an existing diskette file.
- Displays all or part of a file (either on the display screen or printer, depending on the current system list device assignment for the requesting work station).

The disk copy/display utility consists of the following modules that reside in the system library:

- Disk copy display mainline phase ($COPY)
- Copy all files primary phase ($COALL)
- Add to diskette file phase ($COADD)
- Copy initialization phase – for copying all files ($COANT)
- Copy initialization phase – for copying one file ($COINT)
- Input/output interface – for record mode ($COGET)
- Sectorized file copy phase ($COZIP)
- Record exclusion routine ($COSEL)
- SYSLIST interface ($COPRT)
- Window (CRT) display interface routine ($COCRT)
- Syntax specification module ($COTAB)

The disk copy/display utility is called by the appropriate procedure or OCL statements. (See the *System Support Reference Manual* for more information about calling and executing $COPY.)

Input to the utility consists of the control statements, input file(s) on disk or diskette, and format 1's. $COPY calls the syntax checker (#USYX) to read and syntax check the control statements.

The $COPY module uses control statement information to build the copy communication area ($CCCA) and then loads $COALL, $COADD, $COANT or $COINT, depending upon the function requested.

Depending on the control statement, output can consist of one of the following:

- COPYADD control statement: disk file added to existing diskette file
- COPYALL control statement:
  - All or selected disk data files saved on diskette
  - Previously saved data files restored to disk
- COPYFILE control statement:
  - The entire file copied from disk to diskette, diskette to disk, or disk to another location on disk, using sector mode processing
  - Selected records copied to disk, diskette, printer, or display screen, using record mode processing
  - Indexed file records reorganized in key order
- KEY control statement with COPYFILE control statement: A direct, indexed, or sequential file, with specified records deleted, copied from disk to diskette, diskette to disk, or disk to another location on disk creating an indexed file.

The disk copy/display utility requires 14K bytes of main storage for program execution.

Method of Operation

Diagrams 13.0 through 13.4 show the functions of the disk copy/display utility.

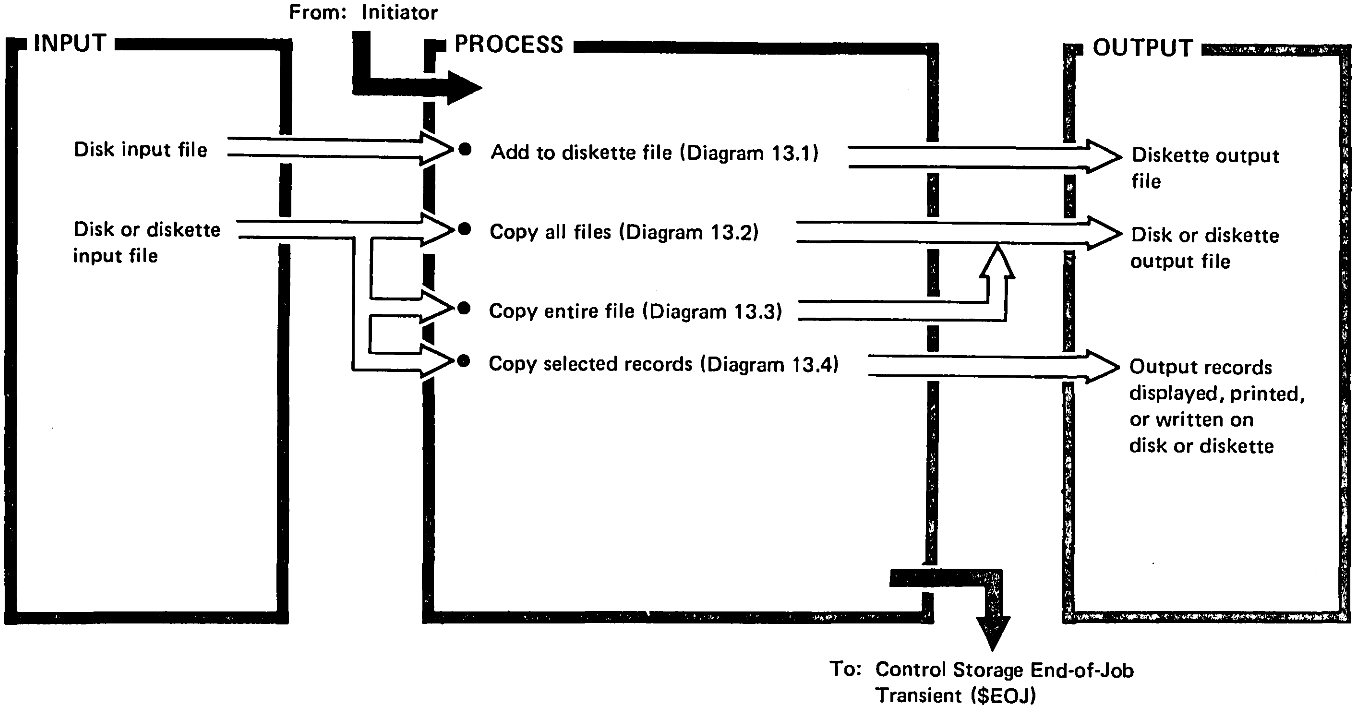
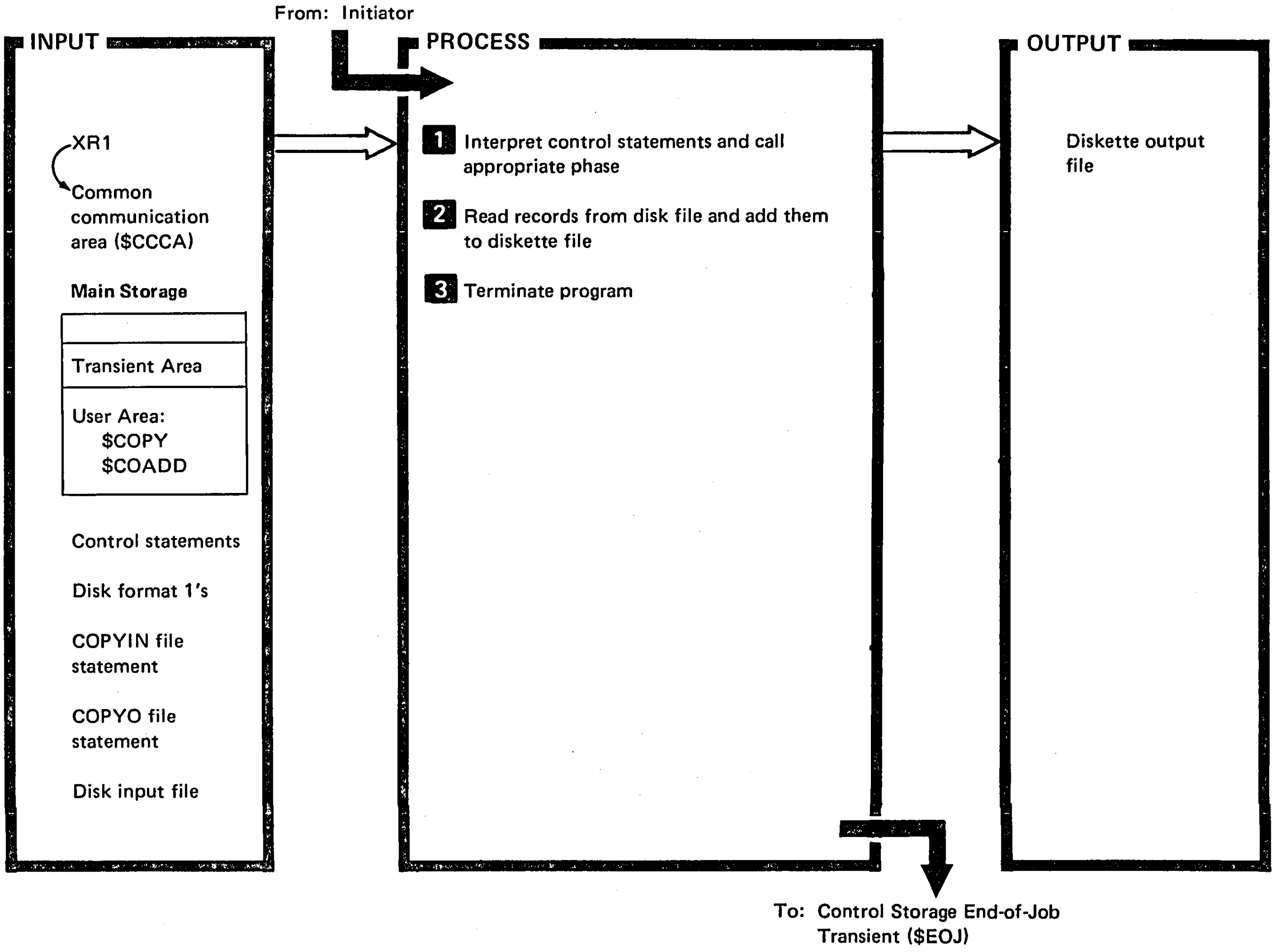

From: Initiator
INPUT
PROCESS
OUTPUT
Disk input file
Disk or diskette input file
● Add to diskette file (Diagram 13.1)
● Copy all files (Diagram 13.2)
● Copy entire file (Diagram 13.3)
● Copy selected records (Diagram 13.4)
Diskette output file
Disk or diskette output file
Output records displayed, printed, or written on disk or diskette
To: Control Storage End-of-Job Transient ($EOJ)

Diagram 13.0. Overview of Disk Copy/Display Utility



From: Initiator

INPUT

XR1

Common communication area ($CCCA)

Main Storage

Transient Area

User Area:
$COPY
$COADD

Control statements

Disk format 1's

COPYIN file statement

COPYO file statement

Disk input file

PROCESS

1 Interpret control statements and call appropriate phase

2 Read records from disk file and add them to diskette file

3 Terminate program

OUTPUT

Diskette output file

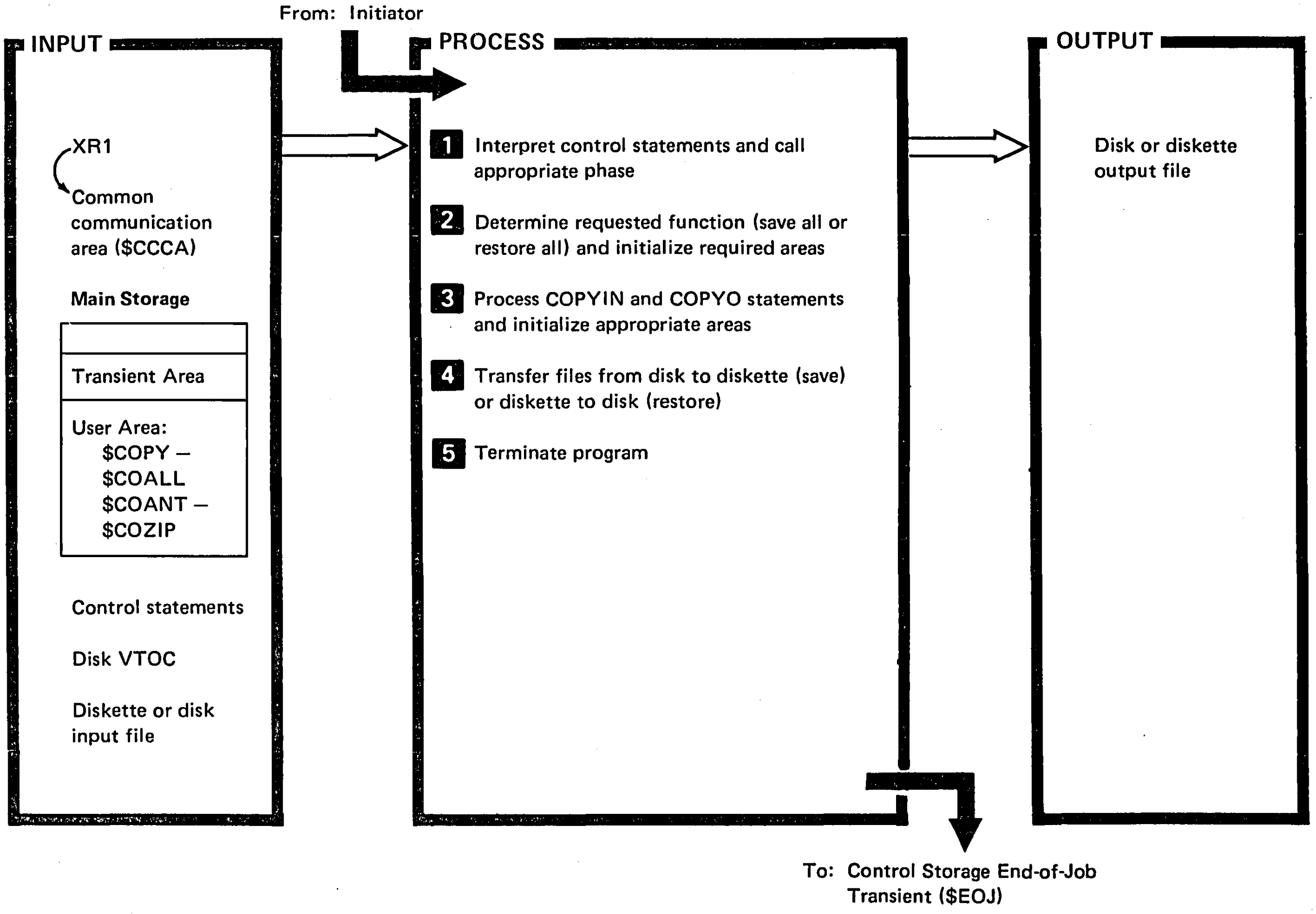
To: Control Storage End-of-Job Transient ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Initialize common communication area ($CCCA).	$COPY
Read and syntax check COPYADD utility control statement.	#USYX
Read and syntax check END utility control statement.	
Load and pass control to add to diskette file phase ($COADD).	$COPY
2 Load sectorized diskette data management (#DRSM) and sectorized disk data management (#DDSM) into main storage.	$COADD
Use active file area (AFA) access routine to find COPYO and COPYIN format 1's.	
Allocate diskette and disk files.	#CAML
Return.	$COADD

Diagram 13.1 (Part 1 of 2). Perform Add to Diskette File – COPYADD – Function ($COPY)

DESCRIPTION	MODULE/ ROUTINE
Open diskette and disk files.	#DMOP
Return.	$COADD
▶ Read records from disk and place in buffer.	#DDSM ◀
Return.	$COADD
Write records from buffer to diskette.	#DRSM
Continue processing until end of file reached on input.	$COADD
Display error messages as necessary.	#CLXS
Close disk and diskette files.	#DMCL
3 Transfer control to end-of-job transient ($EOJ) to terminate job.	$COADD

Diagram 13.1 (Part 2 of 2). Perform Add to Diskette File – COPYADD – Function ($COPY)



From: Initiator

INPUT

XR1

Common communication area ($CCCA)

Main Storage

Transient Area

User Area:
$COPY –
$COALL
$COANT –
$COZIP

Control statements

Disk VTOC

Diskette or disk input file

PROCESS

1 Interpret control statements and call appropriate phase

2 Determine requested function (save all or restore all) and initialize required areas

3 Process COPYIN and COPYO statements and initialize appropriate areas

4 Transfer files from disk to diskette (save) or diskette to disk (restore)

5 Terminate program

OUTPUT

Disk or diskette output file

To: Control Storage End-of-Job Transient ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Initialize common communication area ($CCCA).	$COPY
Read and syntax check COPYALL utility control statement. Read and syntax check END utility control statement.	#USYX
Determine TO and FROM units (disk and diskette). Use active file area (AFA) access routine to read COPYIN and COPYO statements from disk. If input from disk:	$COPY
● Get first (or next) VTOC entry from disk for the specified set of files (a file on disk, a file with a specified group identifier, or a file without any file group identifier).	#CSVF
Load sectorized disk data management (#DDSM) and sectorized diskette data management (#DRSM) into main storage. Load and pass control to copy all files (COPYALL) phase ($COALL).	$COPY

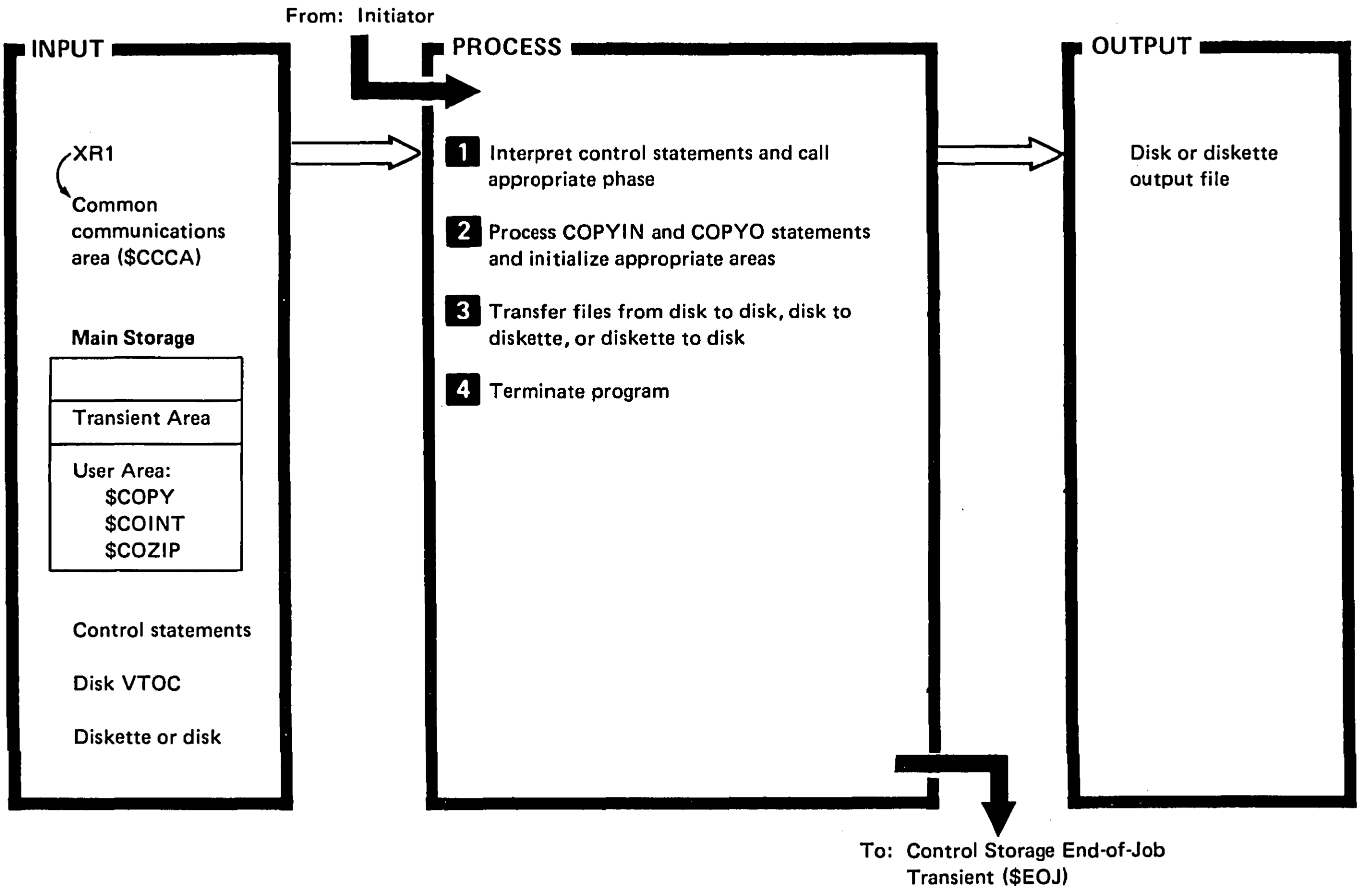
Diagram 13.2 (Part 1 of 3). Perform Copy All Files (COPYALL) Function ($COPY)

DESCRIPTION	MODULE/ ROUTINE
2 Determine whether save all or restore all function requested.	$COALL
If save all requested:	
• Set up format 1 area and generic name.	
• If recursion to $COALL:	
– Get next VTOC format 1 entry for the specified set of files (a file on disk, a file with a specified group identifier, or a file without any file group identifier).	#CSVF
– Issue message to ask for next diskette to be mounted (if necessary).	#CLXS
– Update AFA format 1 for COPYO to diskette.	$COALL
If restore all request:	
• Set up format 1 area and generic name.	
• If recursion to $COALL:	
– Get diskette VTOC format 1.	#CSVI
– Update COPYIN format 1 and place in AFA.	$COALL
3 Load and pass control to copy initialization phase ($COANT).	
If COPYIN for disk (save all request):	
• Ensure validity of utility statement parameters and file type.	$COANT
• Prepare input DTF for disk.	
• Prepare output DTF for diskette.	
• Allocate output file.	#CAML
• Return.	$COANT
• Open output file.	#DMOP
• Load sectorized file copy phase ($COZIP) into main storage and go to **4**.	$COANT
If COPYIN for diskette (restore all request):	
• Allocate input diskette file.	$CAML
• Prepare input DTF for diskette.	$COANT
• Open input file.	#DMOP
• Return.	$COANT
▷ • Read first sector of diskette file for embedded format 1.	#DRSM ◁
• Convert embedded format 1 to AFA format 1.	$COANT
• Prepare output file for disk.	
• Allocate output file.	#CAS1
• Load sectorized file copy phase ($COZIP) into main storage and go to **4**.	$COANT
4 Initialize work area, instructions, and disk (if necessary).	$COZIP
If SAVE function (disk to diskette copy):	
• Open diskette DTF.	#DMOP
• Convert AFA format 1 structure into embedded format 1 structure and place in putput buffer.	$COZIP
• Allocate disk input file.	$CAS1
• Return.	$COZIP
• Open disk DTF.	#DMOP
• Return.	$COZIP
• Read data from disk.	#DDSM
• Return.	$COZIP
• Write data to diskette.	#DRSM
• Continue processing until entire file transferred to diskette.	$COZIP
• Close output (diskette) file and input (disk) file.	#DMCL
• Deallocate input (disk) file.	
• If last file not yet saved, indicate recursion, reload $COALL and go to **2**.	$COZIP
• Go to **5** to terminate program.	

Diagram 13.2 (Part 2 of 3). Perform Copy All Files (COPYALL) Function ($COPY)

DESCRIPTION	MODULE/ ROUTINE
If RESTORE function (diskette to disk copy):	$COZIP
● Open disk output file.	#DMOP
● If indexed file, prepare index buffer.	$COZIP
● Read data from diskette.	#DRSM
● Return.	$COZIP
● Write data to disk.	#DDSM
● If indexed file:	#DRSM
– Copy keys to buffer.	$COZIP
– Write keys to disk.	FDIOS
● Continue processing until entire file copied to disk.	$COZIP
● Update output format 1's in AFA.	
● Close output (disk) and input (diskette) files.	#DMCL
● Deallocate output (disk) file.	
● If last file not yet restored, indicate recursion, reload $COALL, and go to **2**.	$COZIP
● Go to **5** to terminate program.	
5 Transfer control to control storage end-of-job transient ($EOJ) to terminate program.	

Diagram 13.2 (Part 3 of 3). Perform Copy All Files (COPYALL) Function ($COPY)



From: Initiator

INPUT

XR1

Common communications area ($CCCA)

Main Storage

Transient Area

User Area:
$COPY
$COINT
$COZIP

Control statements

Disk VTOC

Diskette or disk

PROCESS

1 Interpret control statements and call appropriate phase

2 Process COPYIN and COPYO statements and initialize appropriate areas

3 Transfer files from disk to disk, disk to diskette, or diskette to disk

4 Terminate program

OUTPUT

Disk or diskette output file

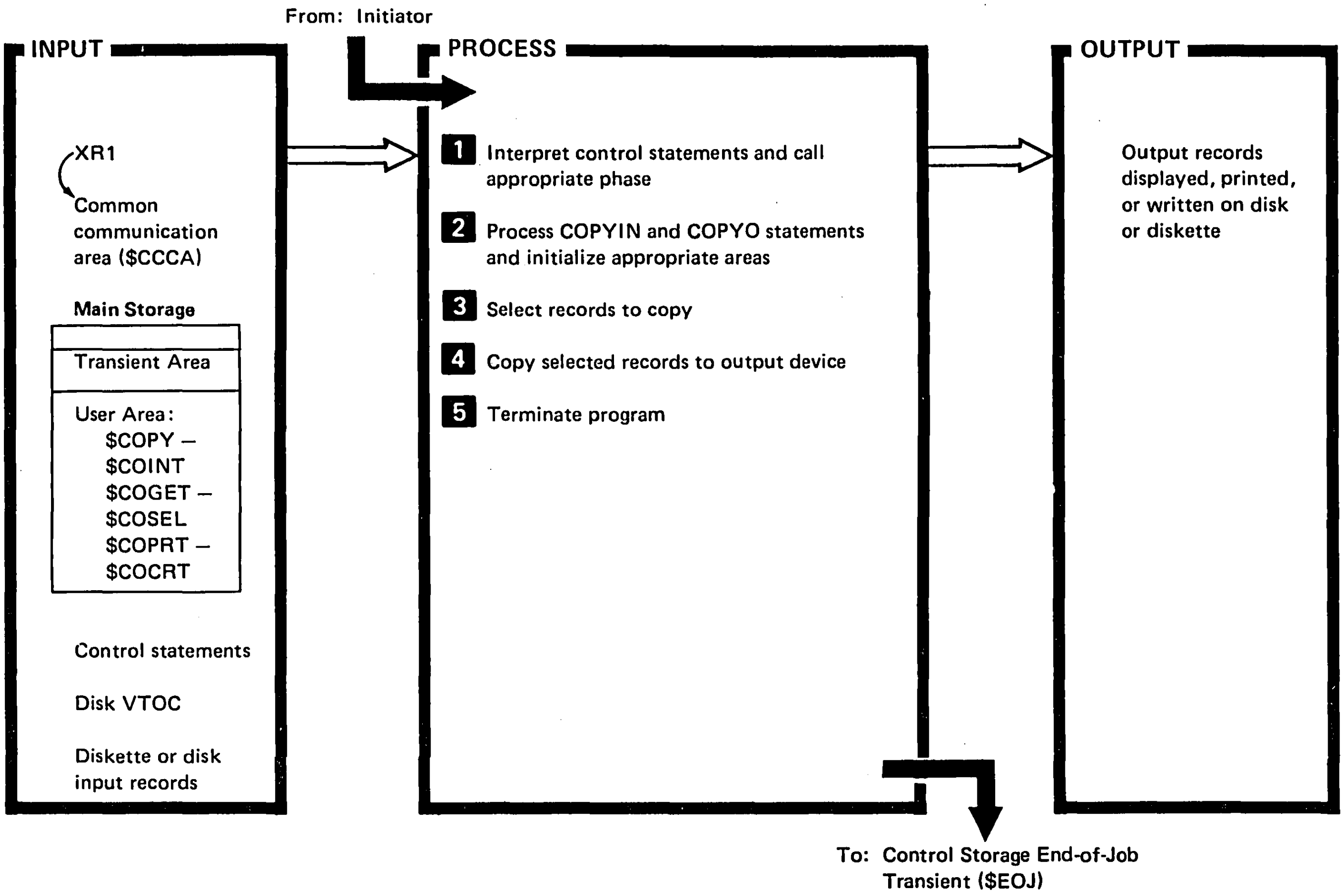
To: Control Storage End-of-Job Transient ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Initialize common communication area ($CCCA).	$COPY
Read and syntax check COPYFILE utility control statement.	#USYX
Ensure control statement is // COPYFILE OUTPUT-DISK,DELETE-NO,REORG-NO (sectorized data management can be used — if not, go to Diagram 13.4).	$COPY
Read and syntax check END utility control statement.	#USYX
Use active file area (AFA) access routine to read COPYIN and COPYO file statements (determines if input from disk or diskette); save attributes in $CCCA.	$COPY
Load disk and diskette sectorized data management (#DDSM and #DRSM) into main storage.	$COPY
Load and pass control to copy initialization phase (COINT).	
2 If COPYIN for disk (input device is disk):	$COINT
▶ ● Allocate input file.	#CAML ◀
● Ensure validity of utility statement parameters and file type.	$COINT
● Prepare input and output device DTFs.	
● Allocate output file.	#CAS1
● Load sectorized file copy phase ($COZIP) into main storage and go to **3**.	$COINT

Diagram 13.3 (Part 1 of 2). Copy Entire File (COPYFILE) Using Sector Mode Processing ($COPY)

DESCRIPTION	MODULE/ ROUTINE
If COPYIN for diskette (input device is diskette):	$COINT
● Allocate input file.	#CAML
▶ ● Prepare input DTF for diskette.	$COINT ◀
● Open input file.	#DMOP
● Return.	$COINT
● Read first sector of diskette file to get embedded format 1.	#DRSM
● Convert embedded format 1 to AFA format 1.	$COINT
● Prepare output file for disk.	
● Update COPYO statement in AFA with file input label.	
● Update output DTF.	
● Allocate output file.	#CAS1
● Load sectorized file copy phase ($COZIP) into main storage and go to **3**.	$COINT
3 Initialize work areas, instructions, and disk if necessary.	$COZIP
If disk to disk copy:	
● Open input and output files.	#DMOP
● Return.	$COZIP
● If file is indexed, read index from disk into buffer and write index to new disk location.	Disk IOS
● Read sector of input data and write to new area on disk (continue processing until entire file transferred).	#DDSM
If disk to diskette copy:	$COZIP
● Open diskette output file and disk input file.	#DMOP
● Return.	$COZIP
● Read data from disk.	#DDSM
● Return.	$COZIP
● Write data to diskette.	#DRSM
● Continue processing until entire file transferred.	$COZIP
● Close output (diskette) file.	#DMCL
If diskette to disk copy:	$COZIP
● Open disk output file.	#DMOP
● Return.	$COZIP
● Read data from diskette.	#DRSM
● Return.	$COZIP
▶ ● Write data to disk.	#DDSM ◀
● If indexed disk file:	$COZIP
— Copy keys to buffer.	#DRSM
— Write sectors of keys to disk.	FDIOS
● Continue processing until entire file copied to disk.	$COZIP
● Update output format 1's in AFA.	
Close input and output files.	#DMCL
Issue error messages as necessary.	#CLXS
4 Transfer control to control storage end-of-job transient (EOJ) to terminate job.	$COZIP

Diagram 13.3 (Part 2 of 2). Copy Entire File (COPYFILE) Using Sector Mode Processing ($COPY)



From: Initiator

INPUT

XR1

Common communication area ($CCCA)

Main Storage

Transient Area
User Area: $COPY – $COINT $COGET – $COSEL $COPRT – $COCRT

Control statements

Disk VTOC

Diskette or disk input records

PROCESS

1. Interpret control statements and call appropriate phase
2. Process COPYIN and COPYO statements and initialize appropriate areas
3. Select records to copy
4. Copy selected records to output device
5. Terminate program

OUTPUT

Output records displayed, printed, or written on disk or diskette

To: Control Storage End-of-Job Transient ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Initialize common communication area ($CCCA).	$COPY
Read and syntax check COPYFILE, SELECT, KEY, and END control statement parameters.	#USYX
If DELETE parameter specified: ● Ensure values are valid. ● Save values specified. Process REORG keywork parameter (if any). If SELECT control statement given: ● Process FROM and TO values for SELECT KEY (if given). ● Process FROM and TO values for SELECT PKY (if given). ● Process FROM and TO values for SELECT RECORD (if given). ● Save values specified. Process KEY parameters (if any) and save values specified.	$COPY
Use AFA access routine to find COPYO statement if PRINT not specified; save attributes in $CCCA. Use active file area (AFA) access routine to find COPYIN statement (determines if input from disk or diskette); save attributes in $CCCA.	$COPY

Diagram 13.4 (Part 1 of 3). Copy Selected Records – COPYFILE – Using Record Mode Processing ($COPY)

DESCRIPTION	MODULE/ ROUTINE
If print not specified, use AFA access routine to find COPYO statement. Set up control for printing if requested. Load and pass control to copy initialization phase ($COINT).	$COPY
2 If COPYIN for disk (input device is disk):	$COINT
● Allocate input file.	#CAML
● Ensure validity of utility statement parameters and file type. ● Use AFA access routine to find COPYO statement if PRINT not specified. ● Prepare input and output device DTFs.	$COINT
● If PRINT not specified, allocate output file.	#CAS1
● Load record mode diskette data management (#DRDM). ● If output is PRINT: – Load SYSLIST interface ($COPRT) into main storage if SYSLIST device is printer. – Load window display routine ($FEKEY) and $COPY interface routine ($COCRT) into main storage if SYSLIST device is display screen. ● Load and pass control to input/output interface module ($COGET) and go to **3**. If COPYIN for diskette (input device is diskette): ·	$COINT
● Allocate input file.	#CAML
● Prepare input DTF for diskette.	$COINT
● Open input file.	#DMOP
● Return.	$COINT
● Read first sector of diskette file to verify data address.	#DRSM
● Convert embedded format 1 to AFA format 1. ● If output not to print: – Prepare output file for diskette. – Update COPYO statement in AFA with file input label. – Update output DTF.	$COINT
– Allocate output file.	#CAS1
– Load record mode disk data management (#DRDM).	$COINT
– Open output diskette file.	#DMOP
● If output is PRINT: – Load SYSLIST interface ($COPRT) into main storage if SYSLIST device is printer. – Load window display routine ($FEKEY) and $COPY interface routine ($COCRT) into main storage if SYSLIST device is display screen. ● Load and pass control to input/output interface module ($COGET) and go to **3**.	$COINT
3 Assign input and output buffer space (output buffer not assigned if output to print).	$COGET
Open output DTF if output to file. Open input DTF if input from disk file..	#DMOP
If SELECT or DELETE requested, load record exclusion routine ($COSEL) into main storage, overlaying beginning of $COGET.	$COGET
Read records from input device.	#DRDM
If SELECT statement, indicate records to exclude that fall outside range sepcified by FROM and/or TO parameters on SELECT statement (range type may be record number, key, or packed key). If DELETE parameter, delete record from output file or SYSLIST when specified character appears at specified position stated by DELETE parameter on COPYFILE statement (enter $COSEL at $COMIT).	$COSEL

Diagram 13.4 (Part 2 of 3). Copy Selected Records – COPYFILE – Using Record Mode Processing ($COPY)

DESCRIPTION	MODULE/ ROUTINE
4 If not output to PRINT request, write selected records to output device.	#DRDM
If output to PRINT request and SYSLIST device is printer:	$COGET
▶ ● If first time call:	$COPRT ◀
– Retrieve page and line headings.	#MGRET
– Initialize headings.	$COPRT
– Get print image from configuration record to determine valid characters.	Disk IOS
– Print heading line.	#CLST
● Put record number or key (if indexed file) into print buffer.	$COPRT
● Print line from buffer.	#CLST
● Convert unprintable characters in record into over/under hexadecimal characters, or if OUTPTX specified, convert all characters in record to over/under format.	$COPRT
● Print records.	#CLST
● Print last line including number of output records.	
Continue processing until all selected records printed.	$COPRT
If output to PRINT request and SYSLIST device is display screen:	$COGET
● If first time call:	$COCRT
– Retrieve heading legends.	#MGRET
– Initialize heading legend values.	$COCRT
– Set window display parameters.	
● Output records to display screen.	$FEKEY
● If input file indexed, display key length and position.	
● Display last line including number of output records.	
Continue processing until all selected records displayed.	$COCRT
Display error messages as necessary.	#CLXS
5 Close input and output files (if applicable) when end-of-file reached on input file.	$COGET
Transfer control to control storage end-of-job transient ($EOJ).	

Diagram 13.4 (Part 3 of 3). Copy Selected Records – COPYFILE – Using Record Mode Processing ($COPY)

Program Organization

Figures 13-1 through 13-4 show the control flow for the disk copy/display utility.

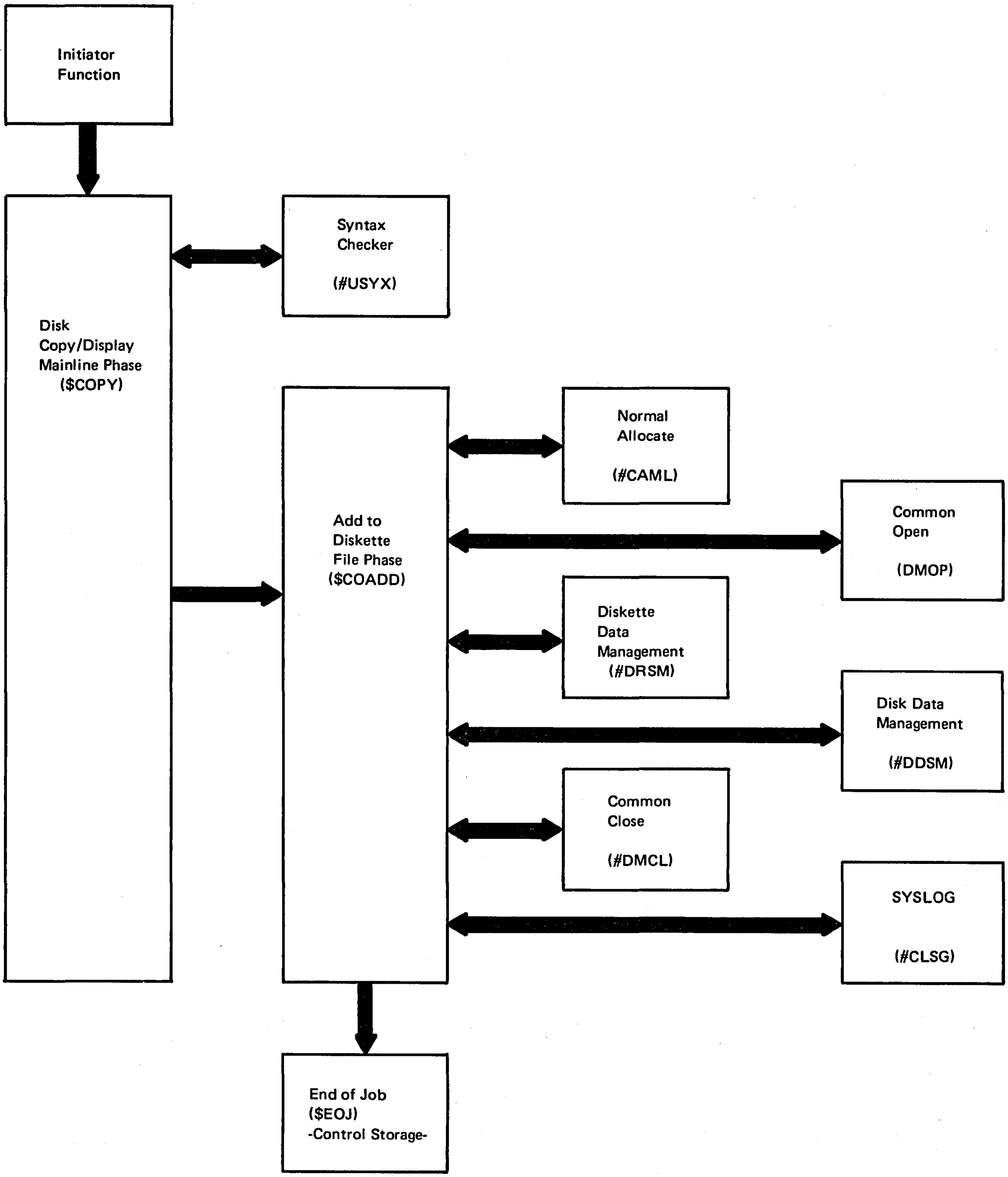

Initiator Function
Disk Copy/Display Mainline Phase ($COPY)
Syntax Checker (#USYX)
Add to Diskette File Phase ($COADD)
Normal Allocate (#CAML)
Common Open (DMOP)
Diskette Data Management (#DRSM)
Disk Data Management (#DDSM)
Common Close (#DMCL)
SYSLOG (#CLSG)
End of Job ($EOJ) -Control Storage-

Figure 13-1. Add To Diskette File (COPYADD) Control Flow ($COPY)

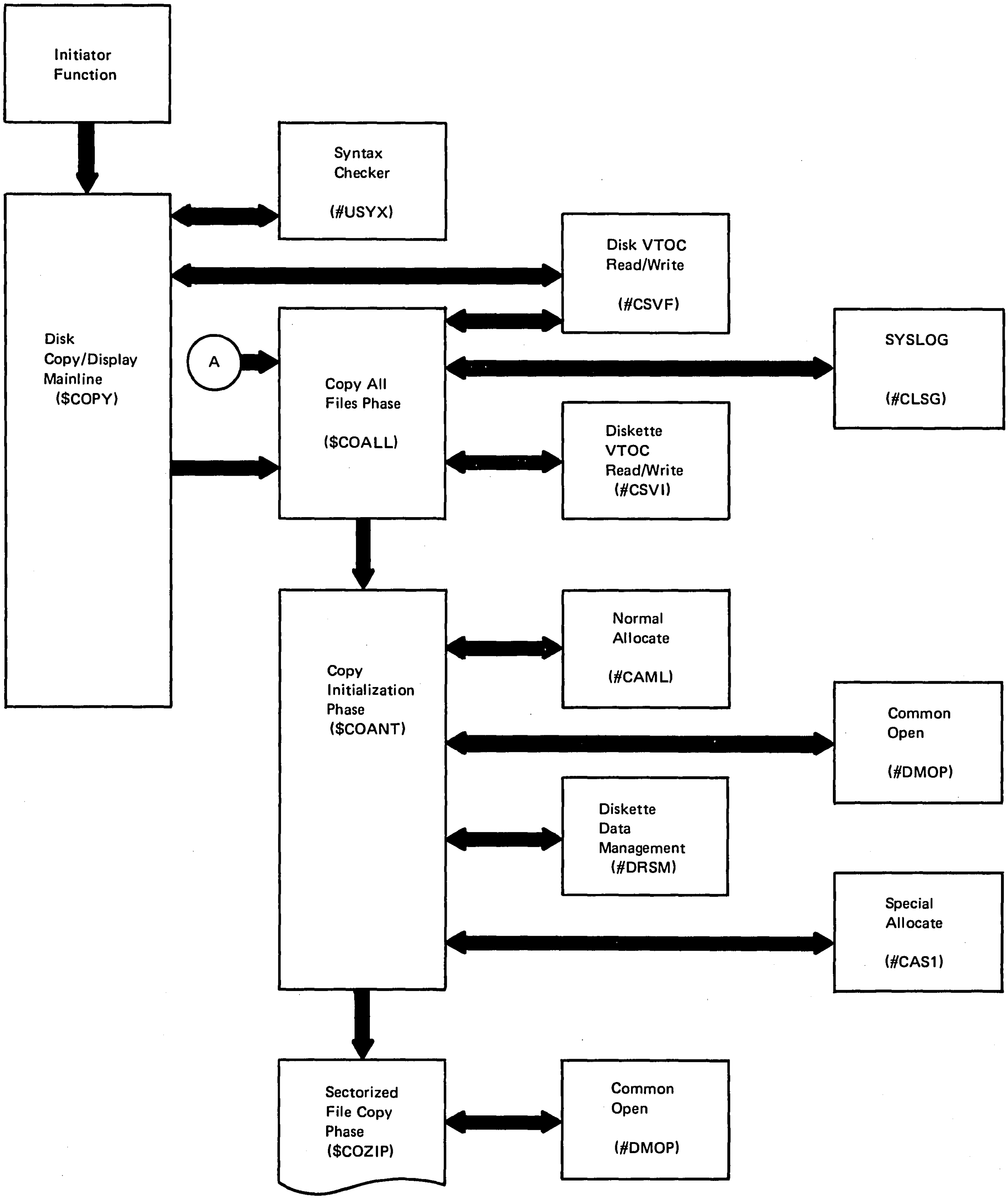

Initiator Function
Syntax Checker (#USYX)
Disk VTOC Read/Write (#CSVF)
Disk Copy/Display Mainline ($COPY)
A
Copy All Files Phase ($COALL)
SYSLOG (#CLSG)
Diskette VTOC Read/Write (#CSVI)
Normal Allocate (#CAML)
Copy Initialization Phase ($COANT)
Common Open (#DMOP)
Diskette Data Management (#DRSM)
Special Allocate (#CAS1)
Sectorized File Copy Phase ($COZIP)
Common Open (#DMOP)

Figure 13-2 (Part 1 of 2). Copy All Files (COPYALL) Control Flow ($COPY)

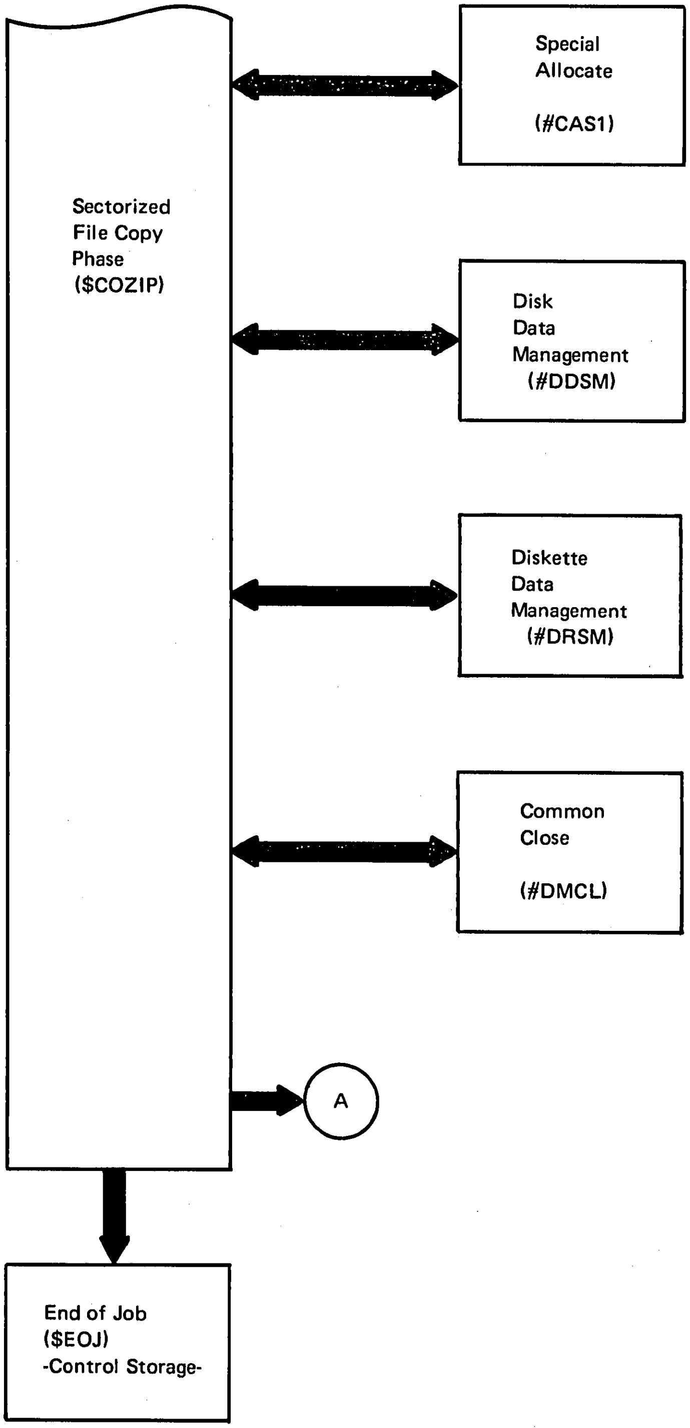


Figure 13-2 (Part 2 of 2). Copy All Files (COPYALL) Control Flow ($COPY)

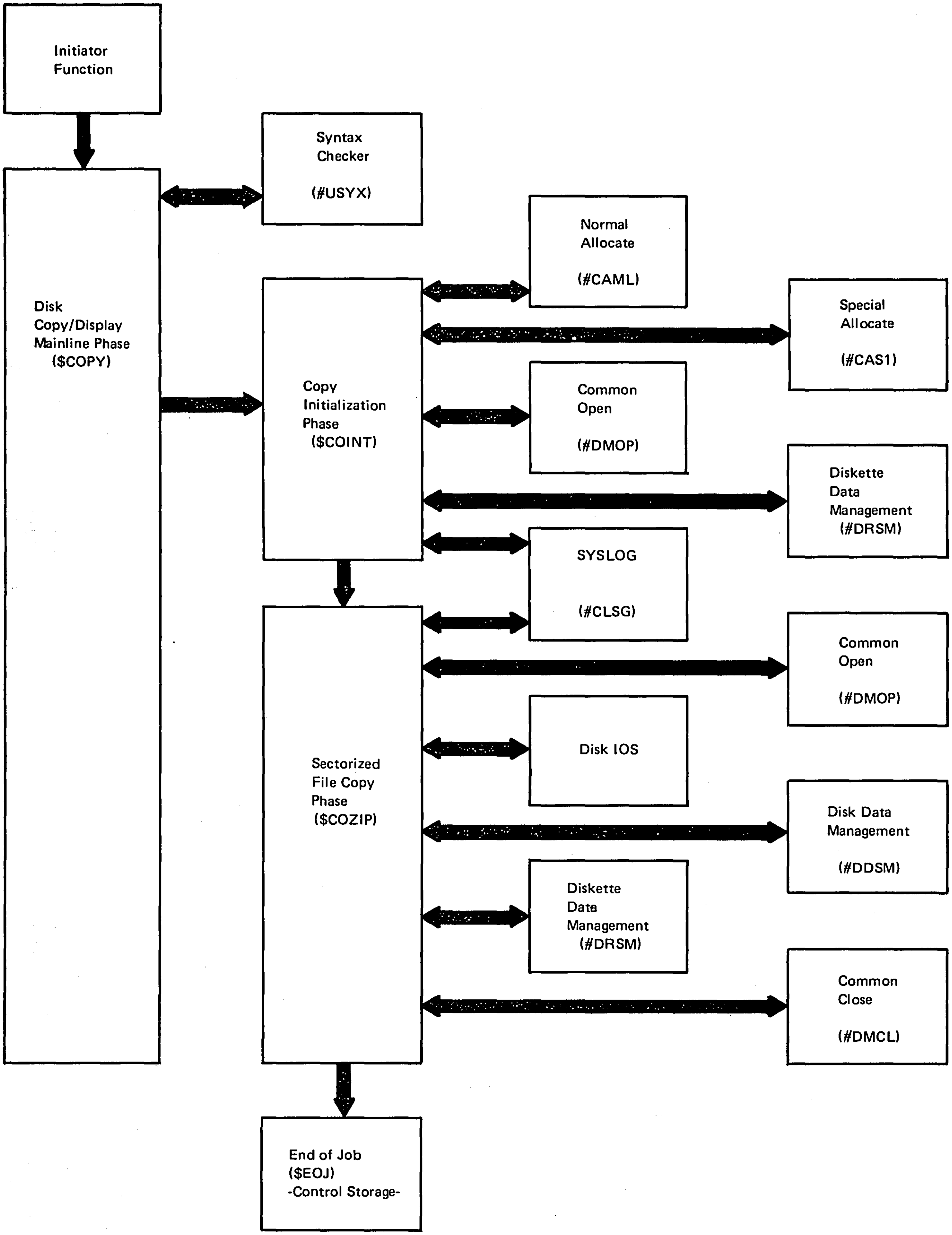

Initiator Function
Syntax Checker (#USYX)
Disk Copy/Display Mainline Phase ($COPY)
Normal Allocate (#CAML)
Special Allocate (#CAS1)
Copy Initialization Phase ($COINT)
Common Open (#DMOP)
Diskette Data Management (#DRSM)
SYSLOG (#CLSG)
Common Open (#DMOP)
Sectorized File Copy Phase ($COZIP)
Disk IOS
Disk Data Management (#DDSM)
Diskette Data Management (#DRSM)
Common Close (#DMCL)
End of Job ($EOJ) -Control Storage-

Figure 13-3. Copy Entire File (COPYFILE) Control Flow ($COPY)

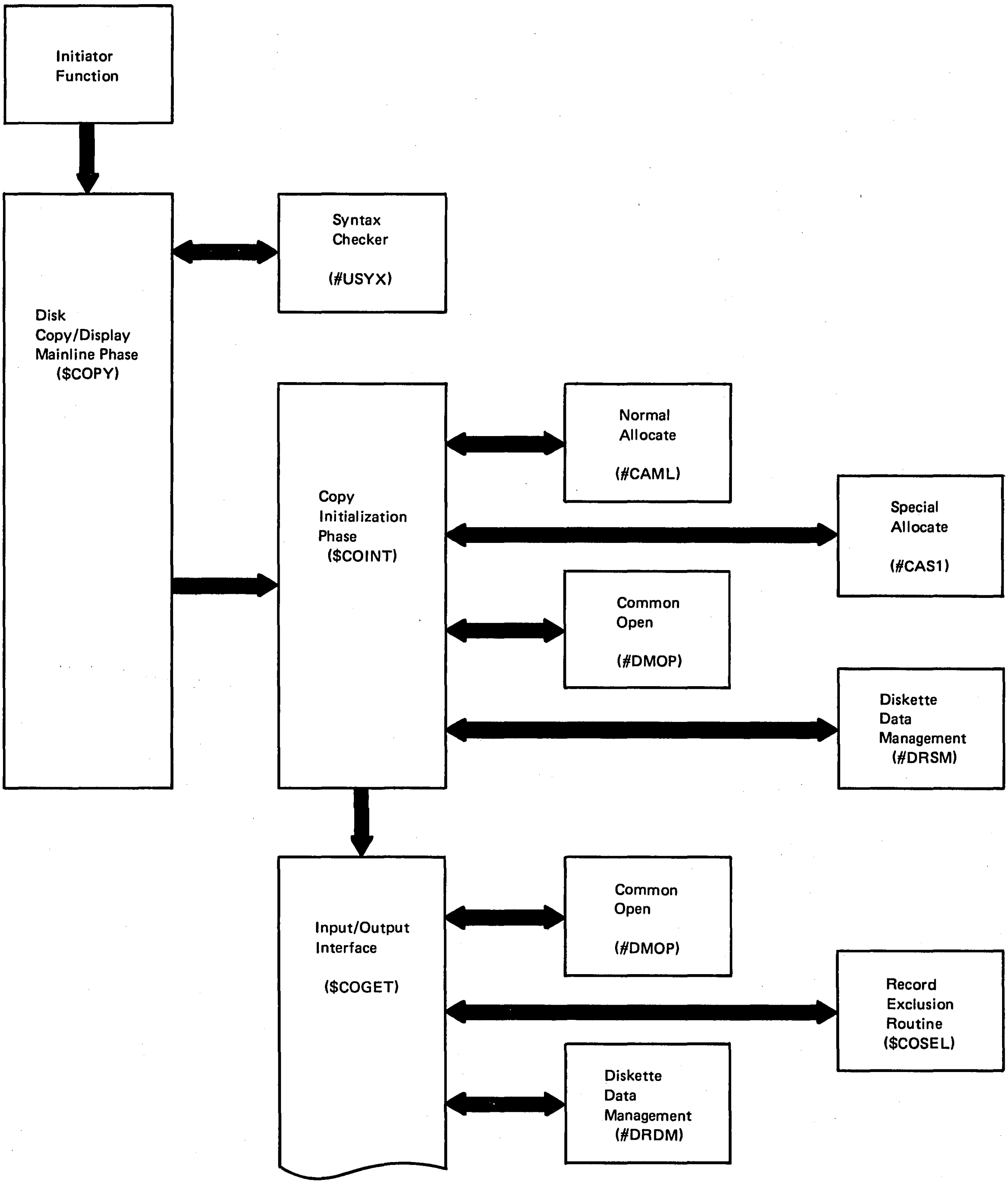

Initiator Function
Disk Copy/Display Mainline Phase ($COPY)
Syntax Checker (#USYX)
Copy Initialization Phase ($COINT)
Normal Allocate (#CAML)
Special Allocate (#CAS1)
Common Open (#DMOP)
Diskette Data Management (#DRSM)
Input/Output Interface ($COGET)
Common Open (#DMOP)
Record Exclusion Routine ($COSEL)
Diskette Data Management (#DRDM)

Figure 13-4 (Part 1 of 2). Copy Selected Records (COPYFILE) Control Flow ($COPY)

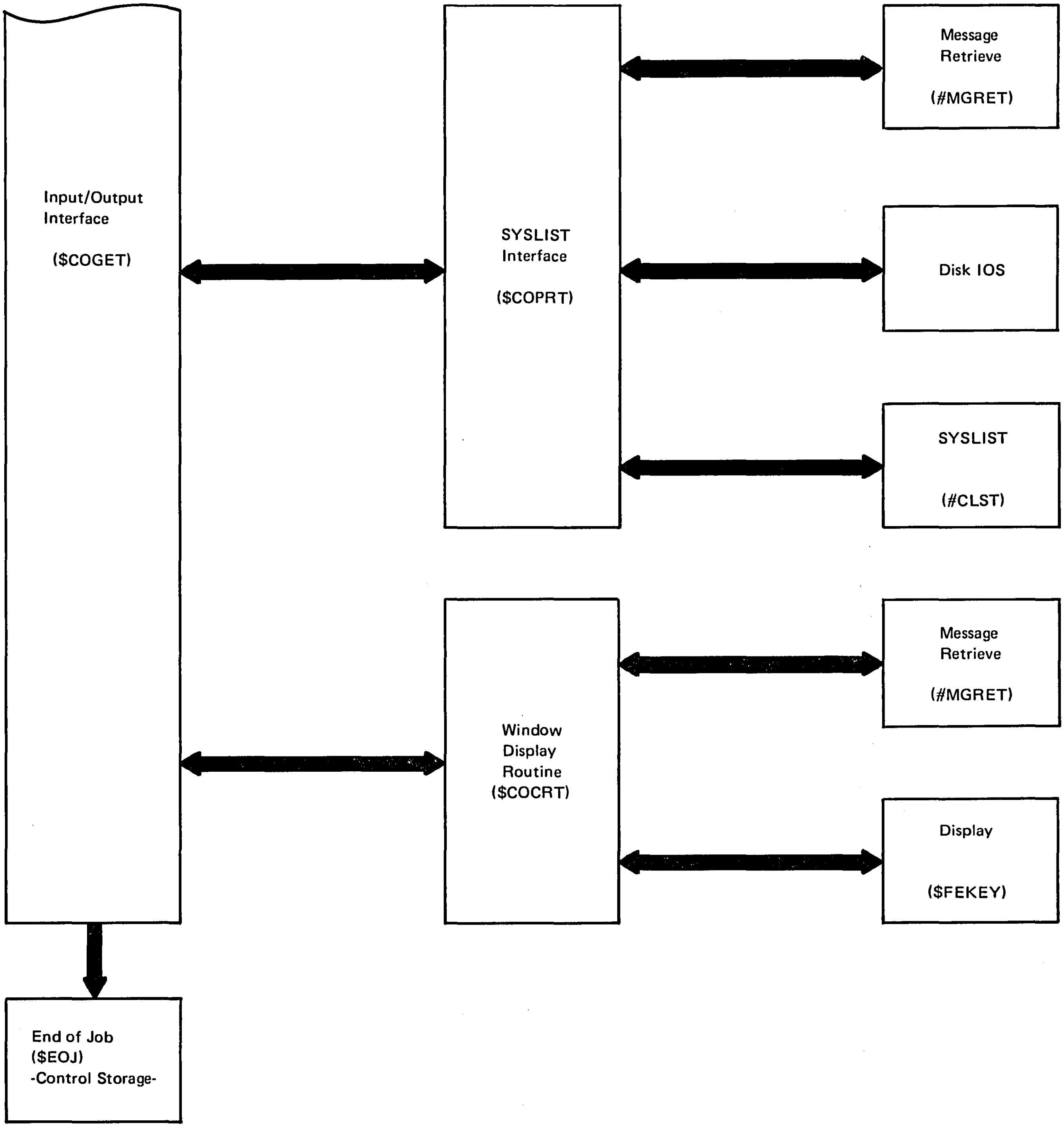


Figure 13-4 (Part 2 of 2). Copy Selected Records (COPYFILE) Control Flow ($COPY)

VERB LISTS – PYZVL1, PYZVL2, PYZVL3

There are three verb lists for the disk copy/display utility; PYZVL1, PYZVL2, and PYZVL3.

Each time the syntax checker (#USYX) is called, one of the lists must be passed. PYZVL1 is passed on the first call. On each subsequent call, the control statement read determines the verb list that is passed on the next call.

The third and fourth bytes of the syntax checker parameter list contain the address of the verb list. Figure 13-5 shows the format and contents of each verb list.

$COPY COMMON COMMUNICATION AREA – $CCCA

The $COPY common communication area, $CCCA, is an area that is initialized by the copy/display mainline ($COPY) and used to pass information among the phases of the utility. The area is located at the start of the user area in main storage. Figure 13-6 shows the format and contents of the communication area.

Displacement of Leftmost Byte	Label	Length in Bytes	Description
0	PYZVL1	1	COPYFILE verb ID
1		1	COPYADD verb ID
2		1	COPYALL verb ID
3		1	END verb ID
4		1	X'FF' end of verb list
0	PYZVL2	1	END verb ID
1		1	X'FF' end of verb list
0	PYZVL3	1	KEY verb ID
1		1	SELECT verb ID
2		1	END verb ID
3		1	X'FF' end of verb list

Figure 13-5. PYZVL1, PYZVL2, and PYZVL3 Verb Lists

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes in Decimal	Description	Routines that Change Data
00	STTYPE	1	Statement type:	$COPY
			COPYALL X'03' = COPYALL statement read	
			COPYFILE X'04' = COPYFILE statement read	
			SELECT X'11' = SELECT statement read	
			KEY X'0A' = KEY statement read	
			COPYADD X'19' = COPYADD statement read	
01	SELCT	1	Select statement type:	$COPY
			BSKEY X'10' = SELECT KEY statement read	
			BSPKY X'08' = SELECT PKY statement read	
			BSRCD X'04' = SELECT RECORD statement read	
02	SW0	1	COPYFILE statement output type:	$COPY
			BPRTX X'C0' = OUTPTX-PRINT specified	
			BPRT X'40' = PRINT specified	
			BFILE X'20' = OUTPUT-FILE specified	
03	SW1	1	COPYFILE statement reorganization parameter	$COPY
			BRYES X'01' = REORG-YES specified	
			BRNO X'00' = REORG-NO specified	

Figure 13-6 (Part 1 of 6). Copy Communication Area

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes in Decimal	Description	Routines that Change Data
04	SW2	1	COPYFILE statement delete parameter	$COPY
			BDLTE X'40' = Deletion specified X'00' = No deletion specified	
05	SW3	1	File group qualifier indicator	—
06	OMPOS	2	Position of omit character	$COPY
08	OMCHAR	1	Omit character	$COPY
09	KEYLN	2	Key length (from //KEY statement or format 1)	$COPY, $COINT
0B	KEYLOC	2	Key location (from //KEY statement or format 1)	$COPY, $COINT
0D	SFRM#	3	From parameter of select record statement (initialized to hex 000000)	$COPY
10	STO#	3	To parameter of select record statement (initialized hex FFFFFF)	$COPY
13	INQ	1	Input device Q-code	$COINT, $COALL, $COANT, $COGET, $COZIP
14	OUTQ	1	Output device Q-code	$COINT, $COALL, $COANT, $COZIP, $COGET
15	CPYIND	1	Indicator	$COGET, $COALL
			IGNORE X'02' = Record to be ignored or error recursion through $COALL module	
			RECALL X'08' = Indicates normal recursion through $COALL module	
16	LOLIM	2	Address of low key limit	$COINT
18	HILIM	2	Address of high key limit	$COINT
1A	FMKEY	29	From parameter of select KEY/PKY statement (initialized to 29XL1'00')	$COPY, $COINT, $COGET

Figure 13-6 (Part 2 of 6). Copy Communication Area

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes in Decimal	Description	Routines that Change Data
37	TOKEY	29	To parameter of select KEY/PKY statement (initialized to 29XL1'FE')	$COPY, $COINT, $COGET
54	PRTOUT	8	SYSLIST device hex 00 = Printer hex 20 = Display screen	$COINT
5C	RCDL	2	Hold area for record length	$COINT, $COANT, $COGET, $COZIP
5E	FMTAFA	2	AFA format 1 address	$COINT, $COANT, $COGET, $COZIP, $COALL
60	INDTF	2	Address of the input file DTF (initialized to X'FFFF')	$COINT, $COANT, $COGET, $COZIP
62	OUTDTF	2	Address of the output file DTF (initialized to X'FFFF')	$COINT, $COANT, $COGET, $COZIP
64	INADDR	2	Address of input data management (#DRDM, #DRSM)	$COINT, $COGET, $COANT, $COZIP
66	OUTADR	2	Address of output data management (#DDSM, #DRSM, $COPRT, $COCRT)	$COINT, $COZIP, $COGET, $COANT
68	WINDO@	2	Address of $FEKEY (if loaded)	$COINT, $COGET
6A	LWRLIM	2	Current lower limit of user program area	$COPY, $COALL, $COINT, $COANT, $COGET, $COZIP
6C	UPRLIM	2	Current upper limit of user program area	$COPY, $COALL, $COINT, $COANT, $COGET, $COZIP
6E	LWRSAV	2	Initial lower limit of user program area	$COPY, $COALL, $COINT, $COANT
70	UPRSAV	2	Initial upper limit of user program area	$COPY, $COALL, $COINT, $COANT
72	COIOA@	2	Address of I/O area for sectorized data management	$COPY, $COINT, $COANT, $COALL, $COGET
74	COEOF@	2	Sector after last data sector on diskette	$COINT, $COZIP

Figure 13-6 (Part 3 of 6). Copy Communication Area

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes in Decimal	Description	Routines that Change Data
76	SVEBKL	2	Diskette block length	$COINT, $COZIP, $COANT
78	COINDS	1	Value of 'DISP' parameter on COPYIN file statement	$COPY
79	COINRT	1	Value of 'RETAIN' parameter on COPYIN file statement	$COPY
7A	CFGAFA	2	Address of AFA format 1 for next disk file to be saved for save all	$COPY
7C	INATTR	2	Input file attribute	$COINT, $COGET, $COZIP
7E	OUTATR	2	Output file attribute	$COINT, $COGET, $COZIP
80	SVEWKB	2	Address of I/O work buffer	$COINT, $COGET, $COZIP, $COANT
82	COFINB	2	Input buffer length – disk	$COALL
84	COFOTB	2	Output buffer length – disk	$COALL
86	RDFLCT	1	Count of files on diskette	$COGET, $COZIP, $COALL
87	RDFLMX	1	Maximum files	$COALL, $COINT, $COZIP
88	CXDOTA	8	File group identifier	$COPY
90	CXDOTL	1	File group identifier length	$COPY
Information Related to Files Copied via COPYALL (displacements hex 91 through hex A7):				
91	ALNDXL	2	Length of index on diskette	$COALL, $COANT
93	ALNDXF	1	Index indicator: F1XMNXSY X'01' = Index requires sort	
94	ALCTYP	1	Control indicator: F1XMCSNG X'00' = Single file copy F1XMCALL X'03' = Copy all files	—

Figure 13-6 (Part 4 of 6). Copy Communication Area

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes in Decimal	Description	Routines that Change Data
95	LABALL	8	COPYALL name	–
9D	ALSQNO	1	File sequence number	–
9E	ALSQND	1	File set indicator:	–
			F1XMOFSC C'C' = Not last file	
			F1XMOFSL C'L' = Last file	
9F	ALNXLB	8	Next file in set	–
A7	ALDLIB	1	Librarian file mode	
			X'81' = Sector mode	
			X'01' = Record mode	
A8	COMXBF	2	Track size (in bytes) of diskette	$COINT, $COANT
AA	RDSCTL	2	Sector size (in bytes) of diskette	$COINT, $COANT
AC	RDZIP@	2	Address where sectorized data management is loaded; X'FFFF' implies not loaded	$COINT, $COANT
AE	RDPREP	1	Indicator for diskette prepare	$COINT
AF	CZIPSW	1	Indicator for sector data management	$COPY, $COINT
B0	RDLAST	1	Physical format of last diskette processed	$COANT
DTFs				
B1	FDDTF1	138	Input DTF #1	$COINT, $COGET, $COALL, $COZIP
13B	RDDTF1	138	Output DTF #1	$COINT, $COGET, $COALL, $COZIP
Format 1 Save Area				
1C5	COFMT1	4	CL4'FMT1'	–
1C9	SVF1IN	87	Input format 1 plus extension	$COINT, $COGET, $COALL, $COANT
220	SVF1CO	87	Output format 1	$COINT, $COANT, $COALL

Figure 13-6 (Part 5 of 6). Copy Communication Area

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes in Decimal	Description	Routines that Change Data
277	SVFSKL	64	Format 1 from AFA	$COALL, $COANT
2B7	COF1SZ	1	Records/blocks request from COPYO statement	$COINT, $COANT
2B8	COF1RT	1	Retention request from COPYO statement	$COINT, $COANT
2B9	COF1A1	1	Attribute byte for spindle preference from COPYO statement	$COINT, $COANT
2BA	COF1LC	1	Disposition request from COPYO statement	$COINT, $COANT
Parameter Lists				
2BB	PYZLOG	24	SYSLOG	$COPY, $COALL, $COINT, $COGET, $COANT, $COZIP
2D3	COAVTI	19	Space for diskette VTOC read/write	$COINT, $COALL
2E6	COAVTF	17	Space for disk VTOC read/write	$COINT, $COALL
2F7	CFGAFA	2	AFA format 1 for next disk file to be saved for save all	—

Figure 13-6 (Part 6 of 6). Copy Communication Area

EMBEDDED FORMAT 1

Each file copied from disk to diskette by the disk copy/display utility has 87 bytes of control information in the first diskette sector immediately preceding the initial sector containing file data on diskette. This control information is called the embedded format 1. It is an extension and modification of the 64-byte AFA format 1 for the file, and it is used to ensure that the file reappears on disk with its original attributes.

The *Data Areas Handbook* shows the format and contents of the embedded format 1.

Chapter 14. File Delete Utility ($DELET)

Introduction

The file delete utility ($DELET) provides a way to free space on disk or diskette for use by new files.

The $DELET utility is called by the DELETE procedure or appropriate OCL statements. (See *System Support Reference Manual* for more information about calling and executing $DELET.)

The file delete utility consists of a mainline phase ($DELET), a disk file delete phase ($DELF1), a diskette file delete phase ($DELI1), and a syntax specification module ($DETAB). The program resides in the system library.

Space is freed in the following way:

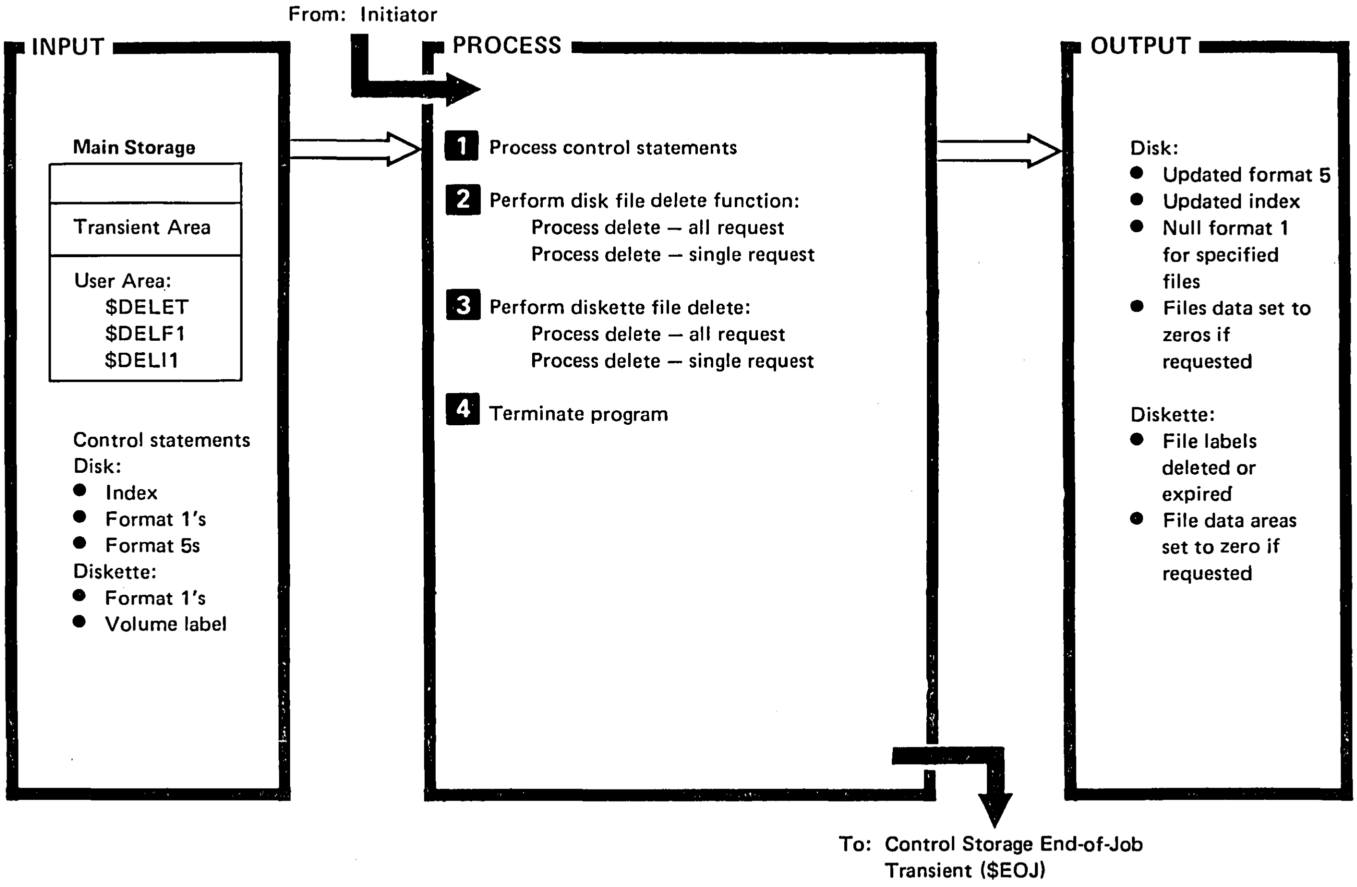
- For a SCRATCH control statement, the diskette file(s) expiration date is changed to the current job date. For disk files, SCRATCH removes the VTOC entry.
- For a REMOVE control statement, the VTOC entry is removed. The contents of the named file(s) on disk or diskette is optionally erased by writing over it with binary zeros.

Control is returned to the system by way of the control storage end-of-job transient ($EOJ).

The file delete utility requires 14K bytes of main storage for program execution.

Method of Operation

Diagram 14.1 shows the function of the file delete utility.



From: Initiator

INPUT

Main Storage

Transient Area

User Area:
$DELET
$DELF1
$DELI1

Control statements
Disk:
- Index
- Format 1's
- Format 5s

Diskette:
- Format 1's
- Volume label

PROCESS

1 Process control statements

2 Perform disk file delete function:
Process delete — all request
Process delete — single request

3 Perform diskette file delete:
Process delete — all request
Process delete — single request

4 Terminate program

OUTPUT

Disk:
- Updated format 5
- Updated index
- Null format 1 for specified files
- Files data set to zeros if requested

Diskette:
- File labels deleted or expired
- File data areas set to zero if requested

To: Control Storage End-of-Job
Transient ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Initialize date to blanks.	$DELET
Find syntax checker (#USYX).	#MASFN
Use main storage relocating loader (SVC 52) to load #USYX into main storage.	
Read and syntax check control statements and perform requested action until //END statement is read.	#USYX
If disk file delete request, go to 2.	$DELET
If diskette file delete request, go to 3.	
If //END statement, go to 4.	
2 Determine delete function requested.	$DELF1
A Delete-all request specified.	
Examine task control block (TCB) chain to ensure no other task running.	
Examine terminal unit block (TUB) to ensure request from master console.	
Set indicator to skip system files (#LIBRARY).	

Diagram 14.1 (Part 1 of 3). Perform File Delete Utility Function ($DELET)

DESCRIPTION	MODULE/ ROUTINE
Read format 1's for files to be processed.	#CSVF
Delete files (also user library if requested):	$DELF1
● Write zeros over file data extent if erase with DATA-YES specified.	Disk IOS
● Return.	$DELF1
● Allocate files to be deleted (must allocate files before deallocate can be performed).	#CAS1
▶ ● Return.	$DELF1 ◀
● Nullify file's format 5, format 1, and index.	#CAD1
● Return.	$DELF1
● Write updated VTOC back to disk.	#CSVF
B Delete single file specified.	$DELF1
Delete file specified:	
● Get file label (and date if given).	
● Read format 1 for file(s) to be processed.	#CSVF
● Return.	$DELF1
● Test for multiple files and issue multifile message (MIC 1621) if necessary.	#CLXS
● If 3 option taken to multifile message, go to 4.	
● Write zeros over file data extent(s) if erase with DATA-YES specified.	Disk IOS
● Return.	$DELF1
● Allocate files to be deleted.	#CAS1
● Return.	$DELF1
● Null file's format 5, format 1, and index.	#CAD1
● Return.	$DELF1
● Write updated VTOC back to disk.	#CSVF
Issue error messages if necessary.	#CLXS
3 Get system date.	$DELI1
Find diskette device in device allocate table.	
Allocate diskette device.	
Determine delete function requested.	
A Delete-all request specified.	
Prepare new diskette.	#CSVI
Verify diskette's ID.	$DELI1
Read format 1 for file(s) to be processed.	#CSVI
If REMOVE request:	$DELI1
● Write zeros over file data extent if erase and DATA-YES specified.	Diskette IOS
▶ ● Display file's data erased message (MIC 1627) if applicable.	#CLXS ◀
● Zero out format 1.	$DELI1
● Remove file label from VTOC.	#CSVI
● Write updated VTOC back to diskette.	
If SCRATCH request:	$DELI1
● Set file expiration date to date in program communication area.	
● Write file's label with new expiration date out to diskette.	#CSVI
Repeat until all files on diskette are deleted.	$DELI1

Diagram 14.1 (Part 2 of 3). Perform File Delete Utility Function ($DELET)

DESCRIPTION	MODULE/ ROUTINE
Display message asking for next volume (MIC 1628):	#CLXS
● If continue option (0) taken, return to 3 A.	$DELI1
● If terminate option (2) taken, go to 4 .	
B Delete single file specified.	
Delete file specified:	
● Find specified file.	
● If PACK specified:	
– Prepare new diskette.	#CSVI
– Verify volume ID correct for mounted diskette.	$DELI1
▶ ● Read format 1 label for specified file (use date field if date specified).	#CSVI ◀
● If multivolume file:	$DELI1
– Check volume sequence.	
– Issue prompts for proper diskettes.	#CLXS
● If REMOVE request:	$DELI1
– Write zeros over file data extents if erase and DATA-YES specified.	Diskette IOS
– Display file's data erased message (MIC 1627) if applicable.	#CLXS
– Zero out format 1.	$DELI1
– Remove file label from VTOC.	#CSVI
– Write updated VTOC back to diskette.	
● If SCRATCH request:	$DELI1
– Set file expiration date to date in program communication area.	
– Write file's label with new expiration date out to diskette.	#CSVI
● If multivolume file, issue message for next diskette (MIC 1485) and repeat delete until entire file deleted.	#CLXS
Issue error messages if necessary.	
4 Pass control to control storage end-of-job transient ($EOJ).	$DELET

Diagram 14.1 (Part 3 of 3). Perform File Delete Utility Function ($DELET)

Program Organization

Figure 14-1 shows the control flow for the file delete utility.

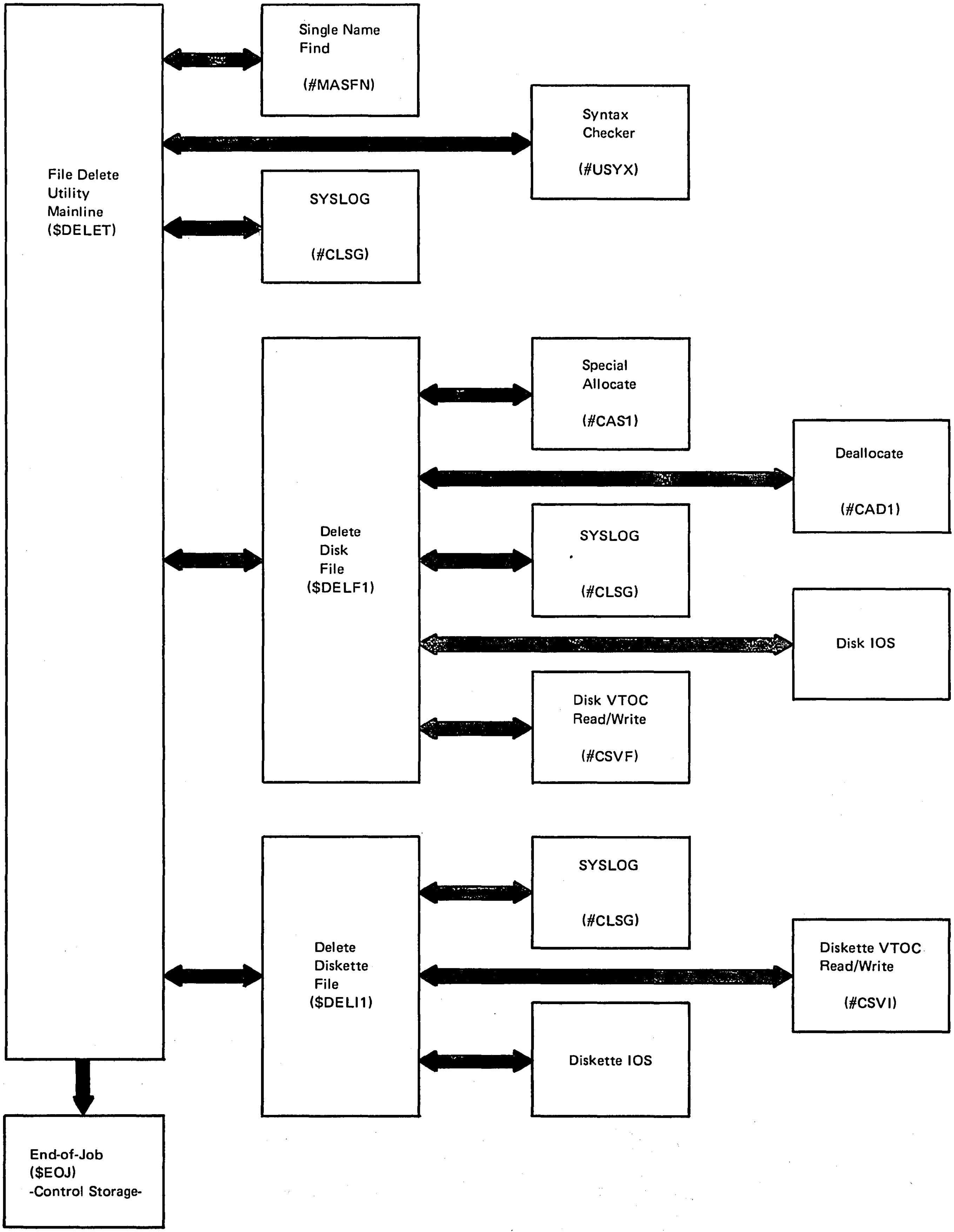

Single Name Find (#MASFN)
Syntax Checker (#USYX)
File Delete Utility Mainline ($DELET)
SYSLOG (#CLSG)
Special Allocate (#CAS1)
Deallocate (#CAD1)
Delete Disk File ($DELF1)
SYSLOG (#CLSG)
Disk IOS
Disk VTOC Read/Write (#CSVF)
SYSLOG (#CLSG)
Delete Diskette File ($DELI1)
Diskette VTOC Read/Write (#CSVI)
Diskette IOS
End-of-Job ($EOJ) -Control Storage-

Figure 14-1. File Delete Utility Control Flow ($DELET)

Chapter 15. Diskette Copy Utility ($DUPRD)

Introduction

The diskette copy utility ($DUPRD) copies one or all data files from one diskette to another diskette. The copied files are placed on the output diskette such that all unused space follows the data files, except multivolume files remain at the same physical position.

When copying all files, the system user has the option to delete expired files. The space previously used by the expired files is removed, causing the active files to be contiguous on the output diskette.

The diskette copy utility consists of the $DUPRD phase and $DUTAB syntax specification module. The program resides in the system library.

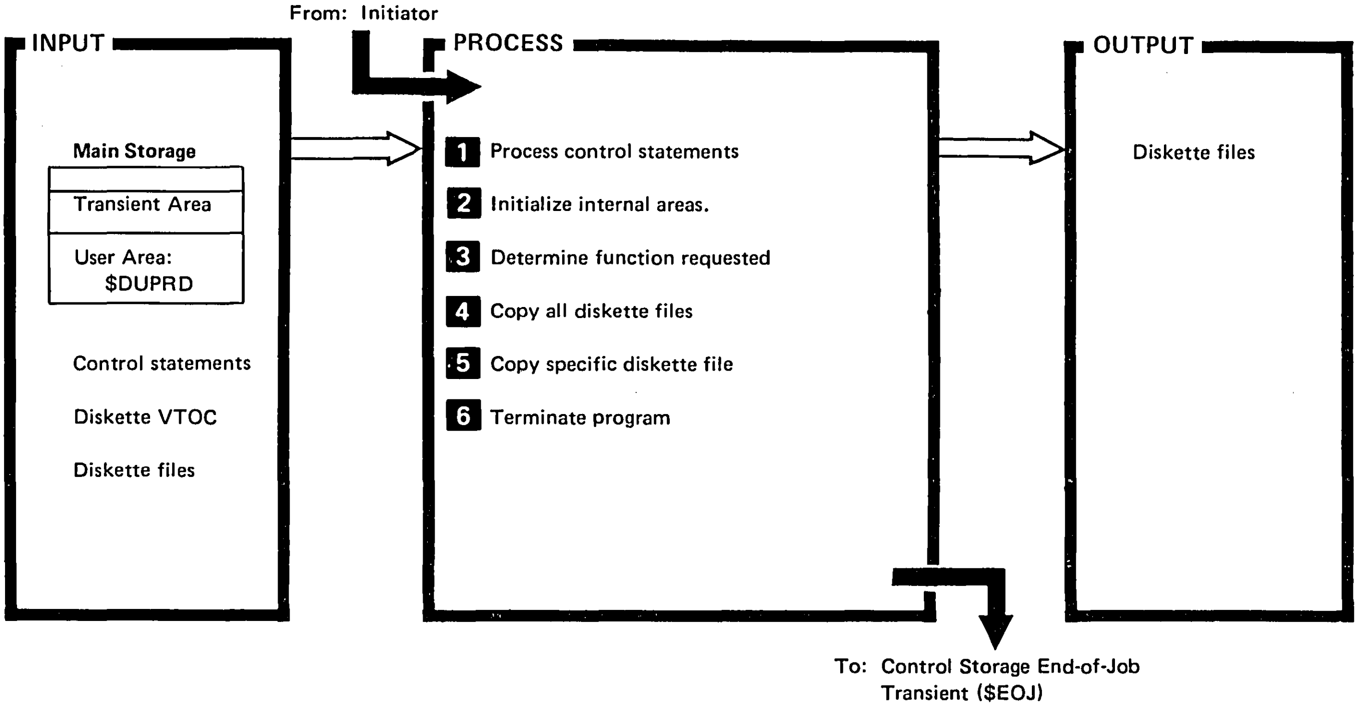
$DUPRD is called by the COPYI1 procedure or appropriate OCL statements. (See the *System Support Reference Manual* for more information about calling and executing $DUPRD.)

The diskette copy utility does not support copying between diskettes of different formats; that is, a standard format diskette cannot be copied to an extended format diskette or vice versa. The bytes per sector must also be the same.

The utility requires 14K bytes of main storage for program execution and enough disk space must be available to contain the data and control records for the diskette files being copied. The system operator may, however, take advantage of more main storage by increasing the region size beyond 14K bytes.

Method of Operation

Diagram 15.1 shows the function of the diskette copy utility.



From: Initiator

INPUT

Main Storage

Transient Area

User Area: $DUPRD

Control statements

Diskette VTOC

Diskette files

PROCESS

1 Process control statements

2 Initialize internal areas.

3 Determine function requested

4 Copy all diskette files

5 Copy specific diskette file

6 Terminate program

OUTPUT

Diskette files

To: Control Storage End-of-Job Transient ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Find syntax checker (#USYX).	#MASFN
Read and syntax check utility control statements.	#USYX
If END card read, go to 6.	$DUPRD
If COPYI1 card read, go to 2.	
Issue error message (MIC 1651) if no valid control statement found.	#CLXS
2 Allocate device for diskette.	#CAML
Return.	$DUPRD
Prepare diskette.	#CSVI
Return.	$DUPRD
Find disk data management (#DDSM) and diskette data management (#DRSM).	#MASFN
Set I/O buffer size based on unused main storage region size.	$DUPRD
3 Determine if entire diskette to be duplicated or just one file.	
If COPY-ALL specified, go to 4.	
If COPY-ONE specified, go to 5.	

Diagram 15.1 (Part 1 of 3). Perform Diskette Copy Utility Function ($DUPRD)

DESCRIPTION	MODULE/ ROUTINE
4 Read all format 1's in diskette VTOC and place in table of modified format 1's (DUPF1TAB) (entire format 1 not saved).	#CSVI
If DELETE-Y specified: ● Check DUPF1TAB for expired files. ● Flag expired files with X'F2'.	$DUPRD
Allocate sufficient disk space to hold all diskette data files specified in DUPF1TAB.	#CAS1
Return.	$DUPRD
A Open disk file just allocated.	#DMOP
Return.	$DUPRD
Place format 1 of file(s) to be copied into intermediate disk file.	Disk IOS
Set file type based on interchange level in format 1.	$DUPRD
Open diskette file:	Disk IOS
▶ ● Write active format 1 to AFA.	#CSAF ◀
● Return.	$DUPRD
● Allocate diskette file.	#CAML
● Return.	$DUPRD
● Open diskette file.	#DMOP
● Return.	$DUPRD
Copy data file(s) from input diskette to intermediate disk based on DUPF1TAB (expired files are not copied).	#DRSM
Return.	$DUPRD
Reopen disk file (for disk to diskette copy).	#DMOP
Return.	$DUPRD
Issue halt to mount new diskette (MIC 1647).	#CLXS
Ensure output diskette same track format as input diskette (bytes per sector):	$DUPRD
● Allocate device for diskette.	#CAML
● Return.	$DUPRD
● Prepare diskette.	#CSVI
● Return.	$DUPRD
● Issue error message if wrong format (MIC 1653).	#CLXS
Open output diskette file.	#DMOP
Return.	$DUPRD
Read record(s) from disk.	#DDSM
Return.	$DUPRD
Write record(s) to diskette.	#DRSM
Return.	$DUPRD
Close current diskette file and disk file.	#DMCL
Return to **1** to read next control statement.	$DUPRD

Diagram 15.1 (Part 2 of 3). Perform Diskette Copy Utility Function ($DUPRD)

DESCRIPTION	MODULE/ ROUTINE
5 Find requested format 1 in diskette VTOC and place in DUPF1TAB.	#CSVI
Return.	$DUPRD
If file name not on diskette, issue halt (MIC 1650): ● If option 0 taken return to **1** to read another control card. ● If option 1 taken, retry under same control card.	#CLXS
Allocate sufficient disk space to hold file specified in DUPF1TAB.	#CAS1
Go to **4** Ⓐ to perform copy function.	$DUPRD
6 Pass control to end-of-job transient ($EOJ).	

Diagram 15.1 (Part 3 of 3). Perform Diskette Copy Utility Function ($DUPRD)

Program Organization

Figure 15-1 shows the control flow for the diskette copy utility.

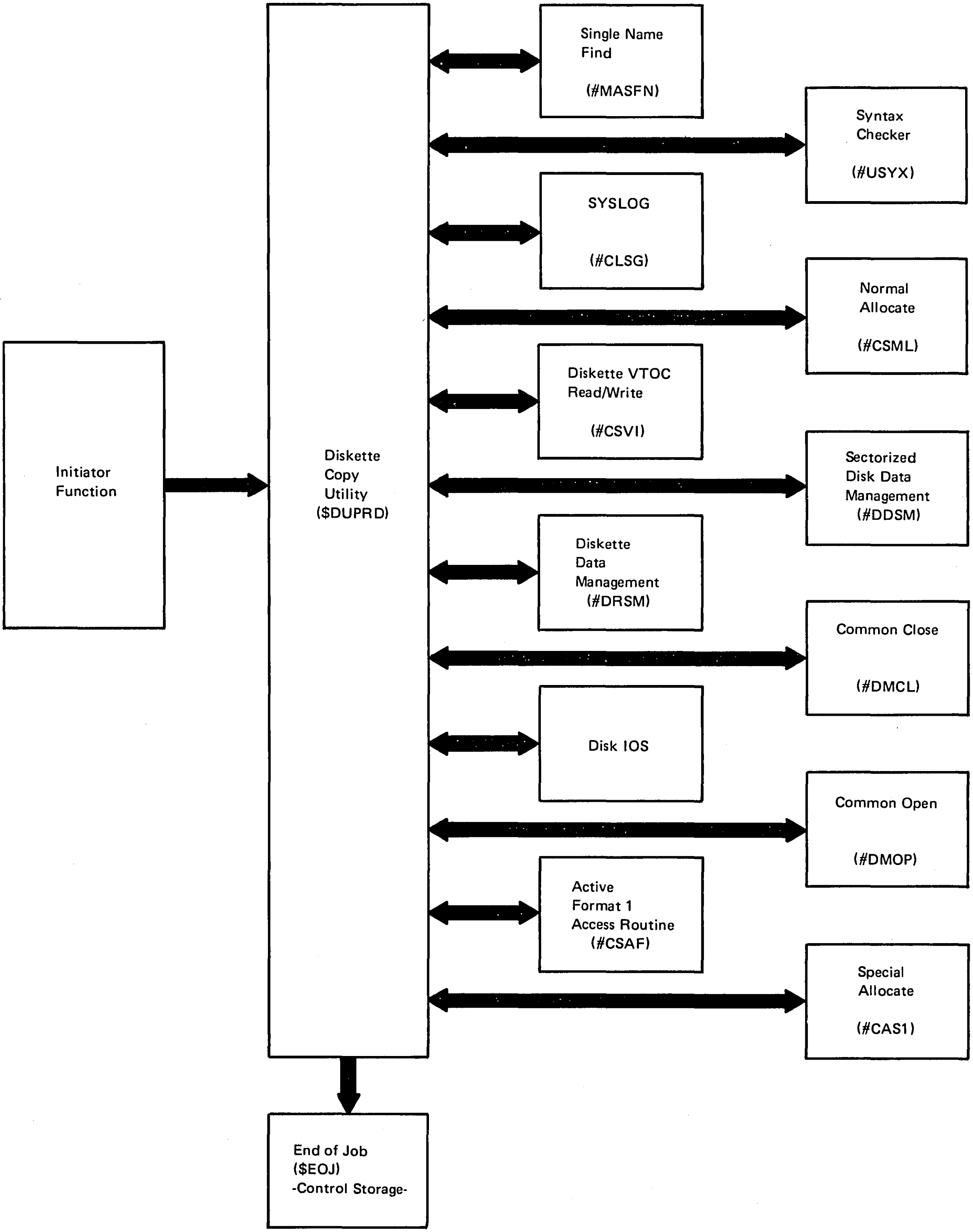

Initiator Function
Diskette Copy Utility ($DUPRD)
Single Name Find (#MASFN)
Syntax Checker (#USYX)
SYSLOG (#CLSG)
Normal Allocate (#CSML)
Diskette VTOC Read/Write (#CSVI)
Sectorized Disk Data Management (#DDSM)
Diskette Data Management (#DRSM)
Common Close (#DMCL)
Disk IOS
Common Open (#DMOP)
Active Format 1 Access Routine (#CSAF)
Special Allocate (#CAS1)
End of Job ($EOJ) -Control Storage-

Figure 15-1. Diskette Copy Utility Control Flow ($DUPRD)

Chapter 16. File Build Utility ($FBLD)

Introduction

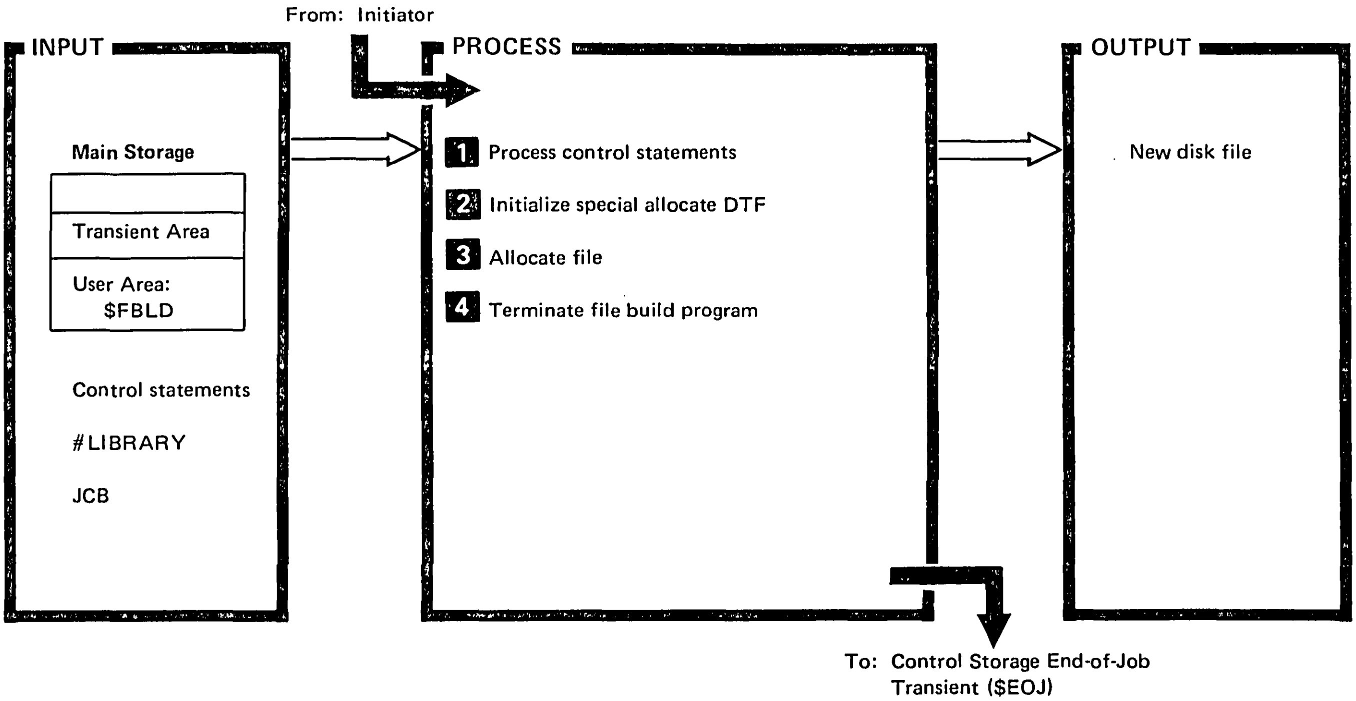
The file build utility ($FBLD) provides a way to allocate disk files without supplying data records. $FBLD is primarily used by application writers to create files that can be referenced as existing files in following steps.

$FBLD creates the file with the precise characteristics specified in the control statements. The utility is called by the BLDFILE procedure or appropriate OCL statements. (See the *System Support Reference Manual* for more information about calling and executing $FBLD.)

The file build utility consists of a mainline module ($FBLD) and a syntax specification module ($FBTAB). The program resides in the system library.

Method of Operation

Diagram 16.1 shows the function of the file build utility.



DESCRIPTION	MODULE/ ROUTINE
1 Clear out work areas and restore DTF.	$FBLD
Find syntax checker (#USYX).	#MASFN
Use main storage relocating loader (SVC 52) to load #USYX.	$FBLD
Read and syntax check control statements. If //END, go to 4.	#USYX
2 If spindle specified, move information into DTF. If location given: ● Scan location value to find length. ● Convert location to hexadecimal. ● Multiply value by 10 and place in DTF. Move records or blocks information into DTF.	$FBLD
Issue error message (MIC 2145) if no records or blocks specified.	#CLXS
If indexed file, set up key position and length. 3 Load and pass control to special allocate (#CAS1).	$FBLD
Allocate new file.	#CAS1
Return to 1 to read next control statement. 4 Pass control to control storage end-of-job transient ($EOJ).	$FBLD

Diagram 16.1. Perform File Build Function ($FBLD)

Program Organization

Figure 16-1 shows the control flow for the file build utility.

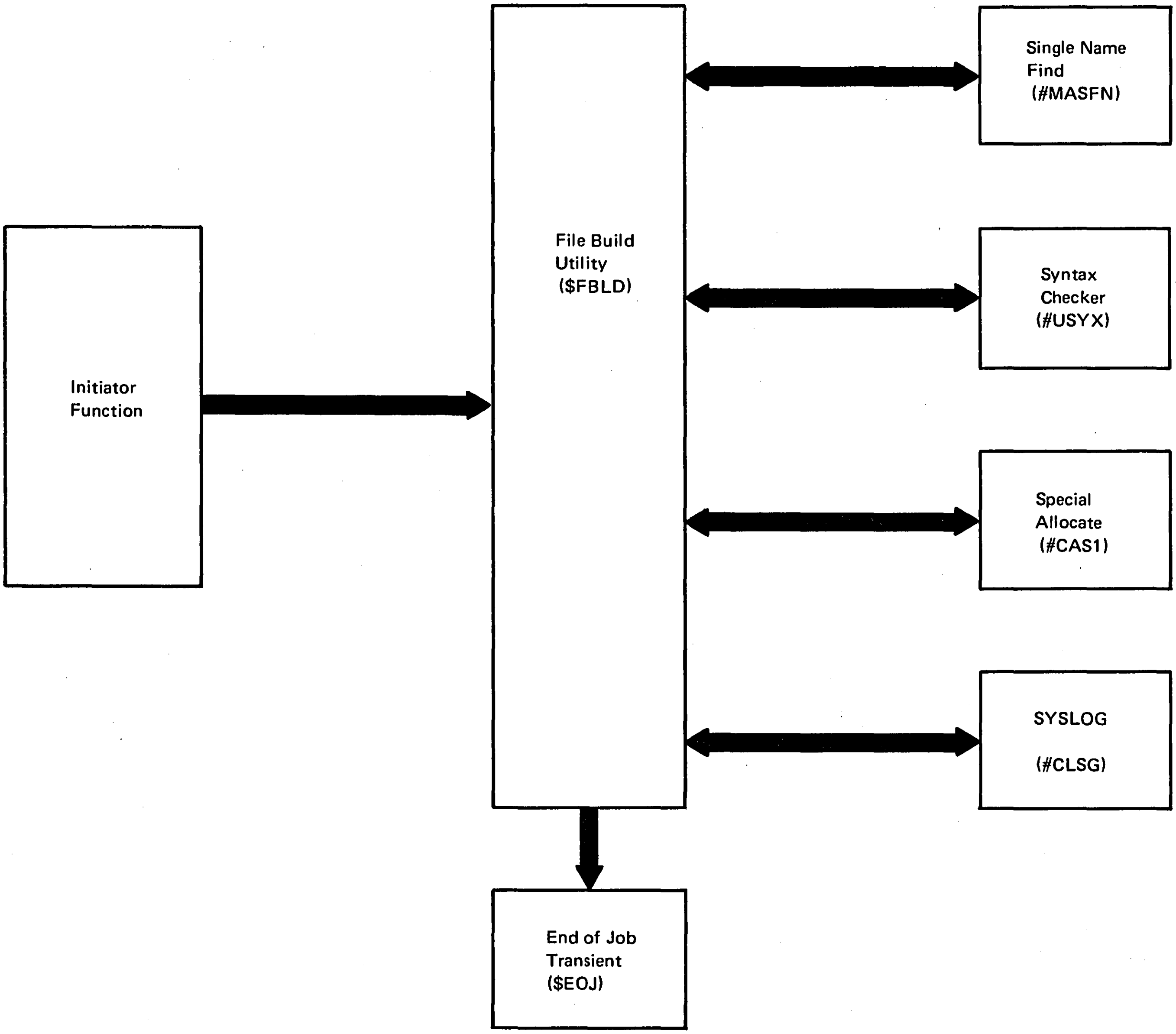

Single Name Find (#MASFN)
File Build Utility ($FBLD)
Syntax Checker (#USYX)
Initiator Function
Special Allocate (#CAS1)
SYSLOG (#CLSG)
End of Job Transient ($EOJ)

Figure 16-1. File Build Utility Control Flow ($FBLD)

Introduction

The history file display utility ($HIST) is the mechanism whereby entries in the history file are printed or displayed to an operator.

The history file is not a data file but is an area in the system area on disk where pertinent operator actions, operator communications, and job queue logging are recorded. Each such display unit, OCL statement, or operator response consists of a line of text associated with an indication whether the operator saw the entry before it was put in the history file or not and whether the text was a broadcast message or not.

The text is prefixed by the user identifier (JCBDUSER) obtained from the job control block (JCB), the terminal identifier (TUBWSID) from the terminal unit block (TUB), an 8 byte job name field and a 6 byte time stamp. If, however, the entry is from the job queue, no valid user identifier or terminal identifier can be supplied. In this case, the job name field contains asterisks.

The indicator fields record whether the text was broadcast or not or seen by the operator or not, and are used to record whether the terminal entries came from a work station or the system console, and mark entries as required for $HIST. Two indicators bytes are at the beginning of each entry. The first byte contains the total length of the entry and the second byte contains the length of the input text.

$HIST performs the following functions:

- Prints or displays all entries
- Prints or displays only entries seen by the operator
- Prints or displays entries that were not printed or displayed before
- Reinitializes the history file — making all entries unavailable

The history file can be accessed in two modes. The modes are defined as a work station request and a system console request. A work station request is from any terminal. A system console request is from the master console with the SYSTEM parameter specified.

If the request for any of the above functions is from a user at a work station, the entries displayed, printed, or made unavailable are restricted to those having both the terminal identifier and user identifier matching the work station identifier and user identifier of the requester.

If the request for any of the above functions is from the system operator at the master console with the SYSTEM parameter specified, access to the history file entries is unlimited.

The history file display utility is called by the HISTORY procedure or appropriate OCL statements. (See the *System Support Reference Manual* for information about calling $HIST.)

The $HIST parameters determine how much history file information is displayed or printed and whether it may be accessed in the future, as follows:

- ALL implies all entries
- If ALL is omitted, only those entries seen by the operator are accessed
- RESET implies that the entries are marked as unavailable for any future display or printing
- If RESET is omitted, the entries are available for further display or viewing
- CURRENT causes entries to be flagged so when CURRENT is again specified, only entries not displayed before are accessed
- TEXTONLY implies the user identifier, terminal identifier, job name, and time stamp are not displayed

$HIST calls the syntax checker to read the DISPLAY and/or END control statement(s) and save the parameters for later interrogation.

The sector address of the sector containing the current history file entry is extracted from the system communication area (SCA).

The entries from the sector containing the current entry to the end of the history file are read into the history file buffer.

The oldest entry in the history file is always the entry logically following the current entry. Beginning with the oldest entry, each entry is examined to determine if the requester is permitted to display, print, or make it unavailable.

If the entry is to be printed or displayed, a heading line containing the user identifier, terminal identifier, job name, and time stamp is printed or displayed unless TEXTONLY was specified. The text in the entry is printed or displayed on a separate line.

If RESET is specified, each entry that the requester is permitted to access is rewritten into the history file with the HFTMRSET bit set. The bit is set in the first byte (HFTDCTLS) of the indicator field if the system console request is specified. The bit is set in the second byte (HFTDCTLW) of the indicator field if the work station request is specified.

If CURRENT is specified, then as each entry is printed or displayed the HFTMCURR is set in the appropriate indicator byte (HFTDCTLS or HFTDCTLW) as indicated above.

Access to entries in the history file is controlled by the mode (work station or system console) of the requester and is further inhibited by the RESET (HFTMRSET) and CURRENT (HFTMCURR) bits.

If a system console request is specified and the RESET (HFTMRSET) bit is set in the console (HFTDCTLS) indicator byte, the entry is unavailable. If the RESET (HFTMRSET) bit is set in the work station (HFTDCTLW) indicator byte, the entry is accessible.

If a system console request is specified, the CURRENT parameter is specified, and the CURRENT (HFTMCURR) bit is set in the console (HFTDCTLS) byte, the entry is not printed nor displayed.

If the requester is from a work station and the RESET (HFTMRSET) bit is set in either of the two indicator bytes, the entry is unavailable.

If the request is from a work station, the CURRENT parameter specified, and the CURRENT (HFTMCURR) bit is set in the work station (HFTDCTLW) indicator byte, the entry is not printed or displayed.

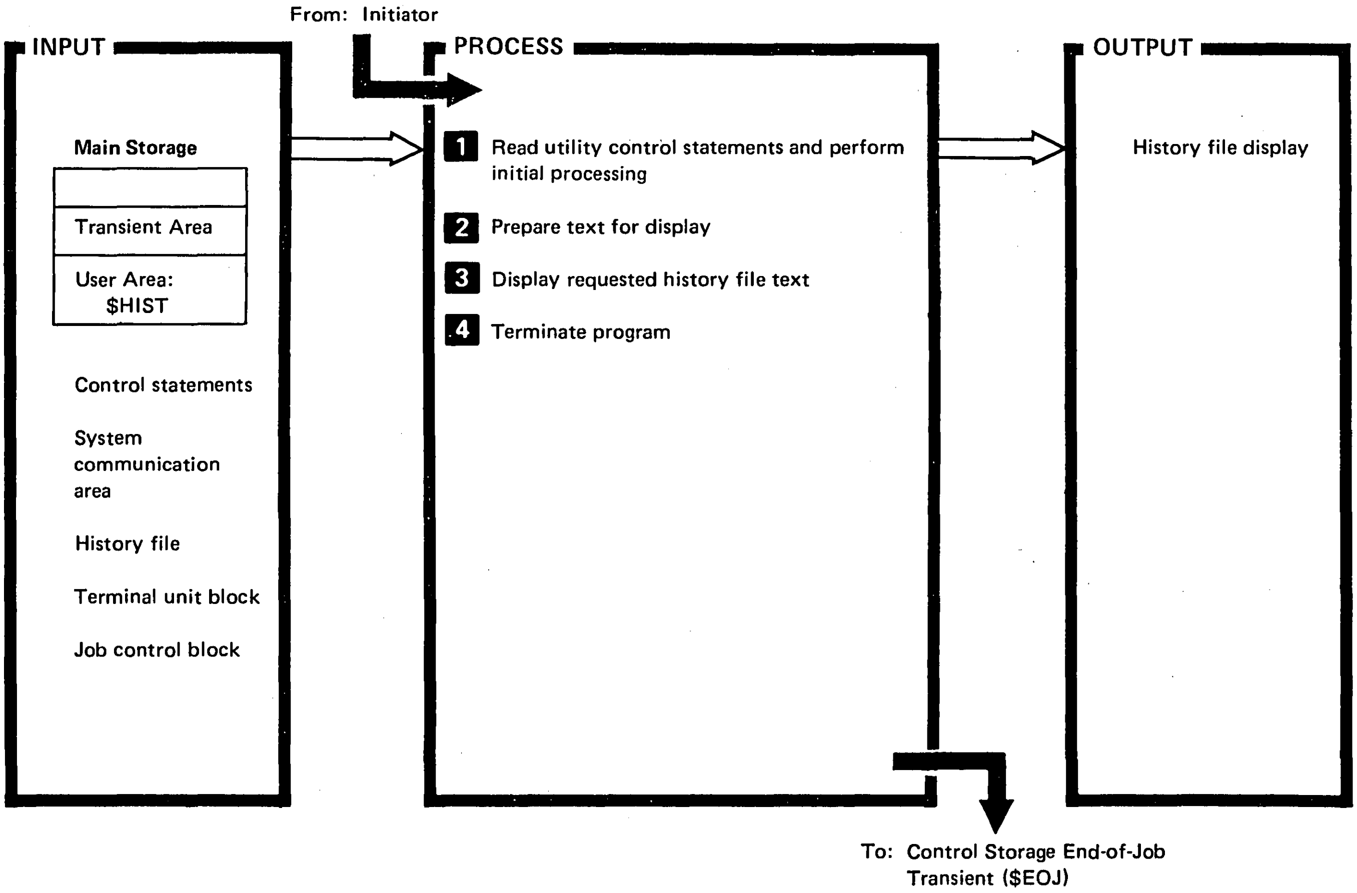
The history file display utility will issue errors by way of system logging if any of the following conditions occur:

- The SYSTEM parameter is specified from a work station that is not the master console
- The length of any entry is outside the permissible range
- A permanent I/O error occurs in reading or writing the disk
- The sector address of the sector containing the current entry is not within the history file area

$HIST requires 14K bytes of main storage for program execution.

Method of Operation

Diagram 17.1 shows the function of the history file display utility.


From: Initiator
INPUT
Main Storage
Transient Area
User Area: $HIST
Control statements
System communication area
History file
Terminal unit block
Job control block
PROCESS
1 Read utility control statements and perform initial processing
2 Prepare text for display
3 Display requested history file text
4 Terminate program
OUTPUT
History file display
To: Control Storage End-of-Job Transient ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Read and syntax check utility control statements.	#USYX
Retrieve inserts from MIC 1667, page headings from MIC 1665, and control legend from MIC 1665.	#MGRET
Use time-of-day function to retrieve time.	$HIST
Determine current history file sector and set up pointer.	
If output to printer, print page heading.	#CLST
Return to function mainline.	$HIST
2 Read proper sector of history file into input buffer.	Disk IOS
Adjust pointer for current entry.	$HIST
If work station request specified, ensure both user and terminal identifiers from requester match entry identifiers (all entries accessible if system console request).	
Determine entries to display based on user request (ALL or CURRENT).	
Prepare control field data for display.	

Diagram 17.1 (Part 1 of 2). Perform History File Display Utility Function ($HIST)

DESCRIPTION	MODULE/ ROUTINE
3 Prepare time stamp for display.	$HIST
Display control information and time stamp if TEXTONLY not specified.	#CLST
Display history file text.	
Update pointers to indicate current and next entries.	$HIST
Write sector just processed back to history file if rewrite required and sector not changed by another task.	Disk IOS
Continue processing until all requested history file entries displayed.	$HIST
Retrieve last line legend and place in SYSLIST buffer.	#MGRET
Place number of output entries in buffer.	$HIST
Output SYSLIST buffer to printer.	#CLST
Issue error messages as necessary.	#CLXS
4 Pass control to control storage end-of-job transient ($EOJ) to terminate program.	$HIST

Diagram 17.1 (Part 2 of 2). Perform History File Display Utility Function ($HIST)

Program Organization

Figure 17-1 shows the control flow for the history file display utility.

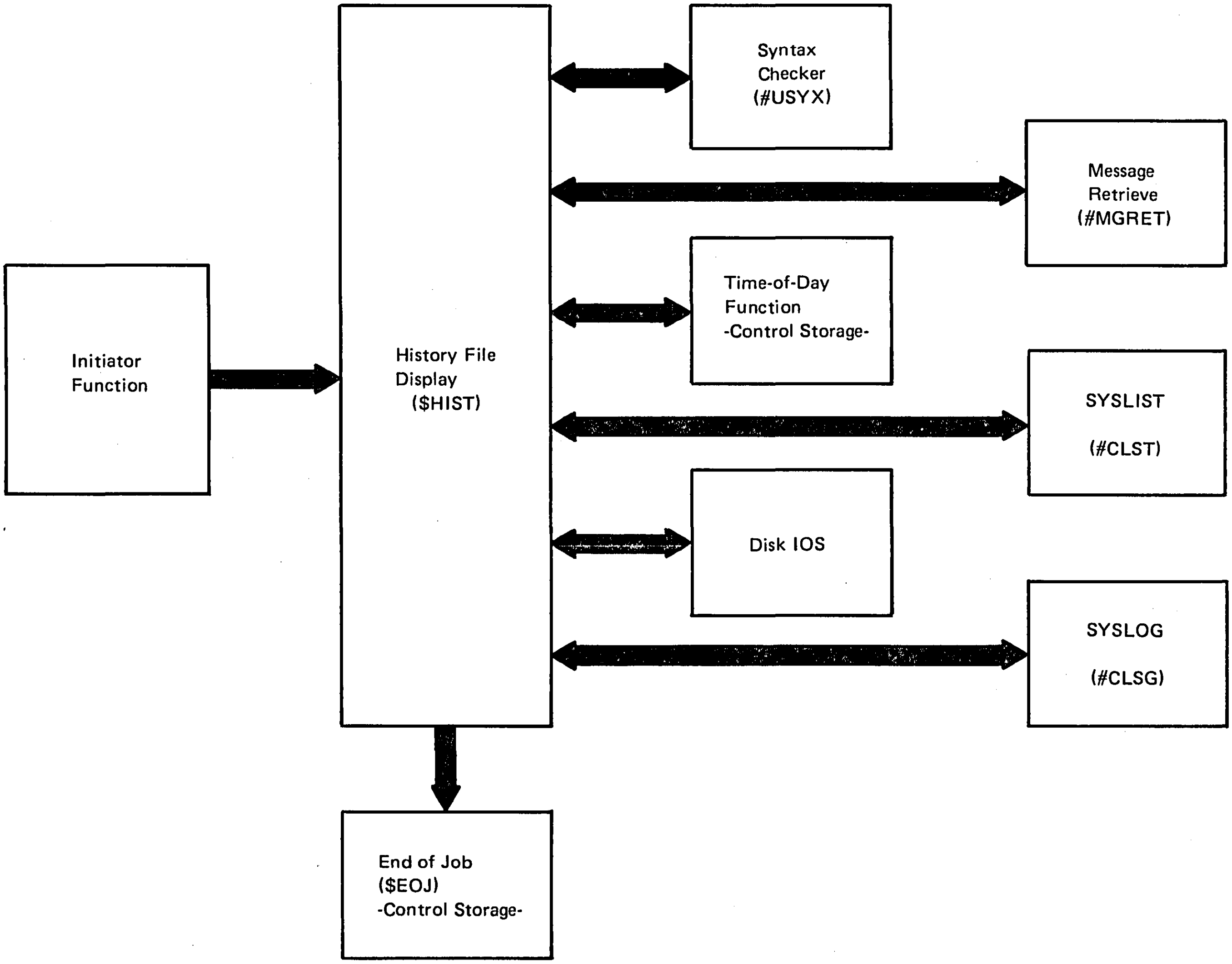

Syntax Checker (#USYX)
Message Retrieve (#MGRET)
Time-of-Day Function -Control Storage-
Initiator Function
History File Display ($HIST)
SYSLIST (#CLST)
Disk IOS
SYSLOG (#CLSG)
End of Job ($EOJ) -Control Storage-

Figure 17-1. History File Display Utility Control Flow ($HIST)

Introduction

The diskette labeling and initialization utility performs one of three functions:

- When the FORMAT or FORMAT2 option is specified, formats the diskette by (1) setting the volume ID and owner ID, (2) initializing all data to blanks, (3) setting the IDs of usable tracks to consecutive numbers, (4) setting the IDs of unusable tracks to X'FFFFFFFF', (5) assigning sector IDs to usable tracks, and (6) writing track zero.

  Note: If FORMAT is specified, single sided diskettes are formatted to contain twenty-six 128-byte sectors. Double sided diskettes format cylinder 0, head 0 to twenty-six 128-byte sectors; cylinder 0, head 1 to twenty-six 256-byte sectors; and cylinder 1 through cylinder 74 to twenty-six 256-byte sectors. If FORMAT2 is specified, single sided diskettes format cylinder 0 to twenty-six 128-byte sectors and cylinder 1 through cylinder 74 to eight 512-byte sectors. Double sided diskettes format cylinder 0, head 0 to twenty-six 128-byte sectors, cylinder 0, head 1 to twenty-six 256-byte sectors, and cylinder 1 through cylinder 74 to eight 1024-byte sectors.

- When the DELETE option is specified, rewrites the HDR labels on cylinder zero, eliminating all references to the files on the diskette.

- When the RENAME option is specified, sets the volume ID and the owner ID to the values specified by the VOL control statement or to default values if no VOL control statement is specified.

This utility consists of the $INIT phase and $INTAB syntax specification module that reside in the system library.

The diskette labeling and initialization utility is called by the INIT procedure or appropriate OCL statements. (See the *System Support Reference Manual* for more information about calling and executing the $INIT utility.)

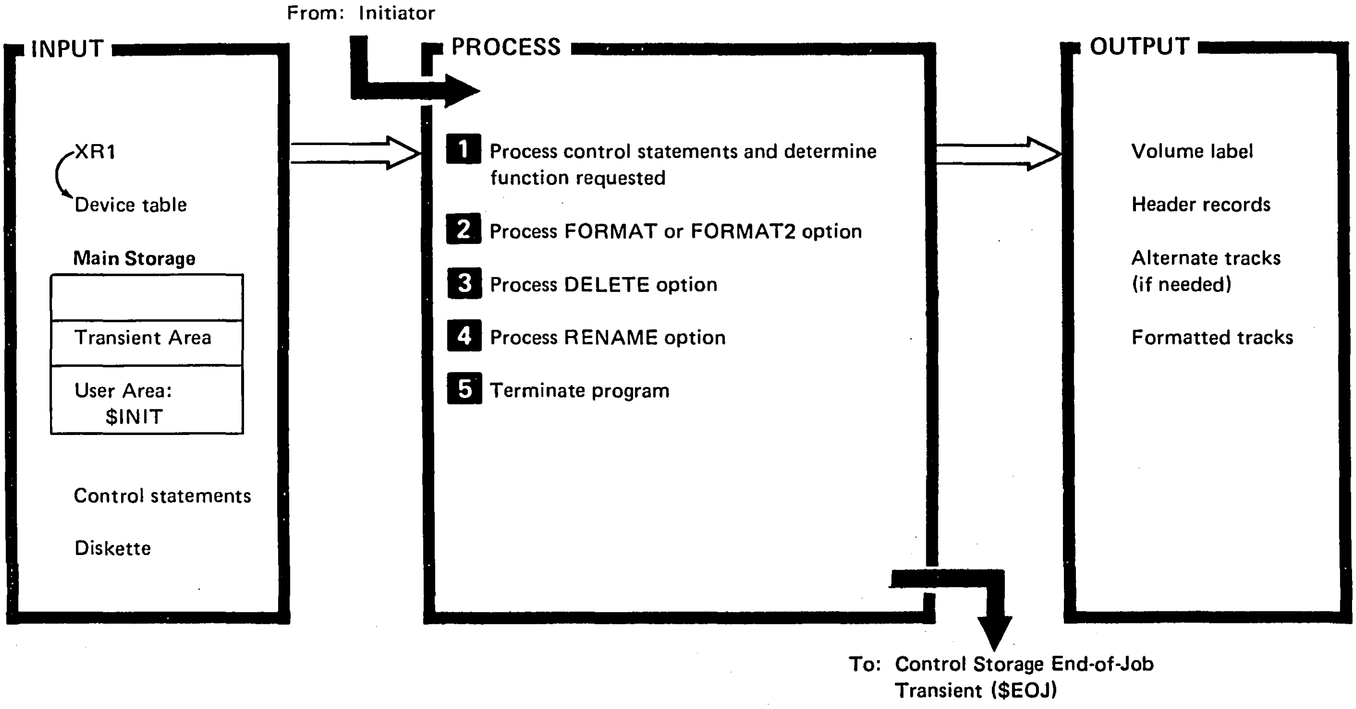
The control statements and the diskette to be initialized are the input to the utility. The syntax checker (#USYX) is used to read the control statements and check them for errors.

Depending upon the function specified by the control statements, the output is (1) an initialized diskette with an updated VTOC and volume label, (2) an updated VTOC in which entries for all files on the diskette have been deleted, or (3) a renamed diskette in which the volume ID and owner ID in the label have been changed.

$INIT requires 14K bytes of main storage and exclusive control of the diskette resources for program execution.

Method of Operation

Diagram 18.1 shows the function of the diskette labeling and initialization utility.



From: Initiator

INPUT

XR1

Device table

Main Storage

Transient Area

User Area: $INIT

Control statements

Diskette

PROCESS

1 Process control statements and determine function requested

2 Process FORMAT or FORMAT2 option

3 Process DELETE option

4 Process RENAME option

5 Terminate program

OUTPUT

Volume label

Header records

Alternate tracks (if needed)

Formatted tracks

To: Control Storage End-of-Job Transient ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Allocate diskette device in device allocate table.	$INIT
Find syntax checker (#USYX).	#MASFN
Read and syntax check control statements.	#USYX
Ensure valid alphameric characters in PACK parameter.	$INIT
Determine option requested: ● If FORMAT or FORMAT2, go to **2**. ● If DELETE, go to **3**. ● If RENAME, go to **4**.	
2 Set format type (one sided or two sided).	
A Check for new diskette (not formatted):	
● Try to read volume label.	Diskette IOS
● Set indicator if new diskette.	$INIT
● Issue MIC 1676 if protected diskette.	#CLXS
B Check for active files (if not new diskette).	$INIT
● Get system date from job control block.	
● Read diskette VTOC.	Diskette IOS
● Check header record for unexpired date.	$INIT
● If files active, issue MIC 1675.	#CLXS
● If permanent diskette error, issue MIC 1673.	

Diagram 18.1 (Part 1 of 2). Perform Diskette Labeling and Initialization Utility Function ($INIT)

DESCRIPTION	MODULE/ ROUTINE
Test surface for defects:	$INIT
● Write IDs with data sectors of all X'E5'. ● Write data sectors of all blanks to all tracks.	Diskette IOS
● Flag entire cylinder as defective if error found. Write labels on track zero:	$INIT
● Build header records for appropriate diskette type (one sided or two sided). ● Write records to diskette.	Diskette IOS
Go to 5 to rename volume ID and/or owner ID on diskette. *Note:* The FORMAT option formats the diskette to: ● Single sided diskette — cylinder 0 through cylinder 74 formatted to twenty-six 128-byte sectors. ● Double sided diskettes: – Cylinder 0, head 0 formatted to twenty-six 128-byte sectors. – Cylinder 0, head 1 formatted to twenty-six 256-byte sectors. – Cylinder 1 through cylinder 74 formatted to twenty-six 256-byte sectors. The FORMAT2 option formats the diskette to: ● Single sided diskette: – Cylinder 0 formatted to twenty-six 128-byte sectors. – Cylinder 1 through cylinder 74 formatted to eight 512-byte sectors. ● Double sided diskette: – Cylinder 0, head 0 formatted to twenty-six 128-byte sectors. – Cylinder 0, head 1 formatted to twenty-six 256-byte sectors. – Cylinder 1 through cylinder 74 formatted to eight 1024-byte sectors.	$INIT
3 Check volume ID: ▶ ● Read diskette volume label.	Diskette IOS ◀
● Issue MIC 1672 if volume ID not same as PACK parameter. ● Issue MIC 1674 if diskette not formatted.	#CLXS
Check VTOC for active files (same as 2 Ⓑ).	$INIT
Write track 0 VTOC (record 8 is inactive file header record). Delete header records 9 through 26. If double sided diskette, write header records on cylinder 0, head 1.	Diskette IOS
Go to 5 to terminate program. 4 Check for new diskette (not formatted) (same as 2 Ⓐ).	$INIT
Issue MIC 1674 if track 0 not formatted.	#CLXS
Update volume ID and/or owner-ID fields of volume label: ● If user supplied valid volume-ID, move volume-ID to new volume LABEL. ● If user supplied valid owner-ID, move owner-ID to new volume LABEL. ● If owner-ID still blank, move words OWNER-ID to volume LABEL. ● Set system identification code in volume LABEL.	$INIT
● Write updated volume LABEL to diskette volume.	Diskette IOS
Go to 5 to terminate program. 5 Deallocate diskette device in device table. Post device waiters (TCBDKTWT). Pass control to control storage end-of-job transient ($EOJ).	$INIT

Diagram 18.1 (Part 2 of 2). Perform Diskette Labeling and Initialization Utility Function ($INIT)

Program Organization

Figure 18-1 shows the control flow for the diskette labeling and initialization utility.

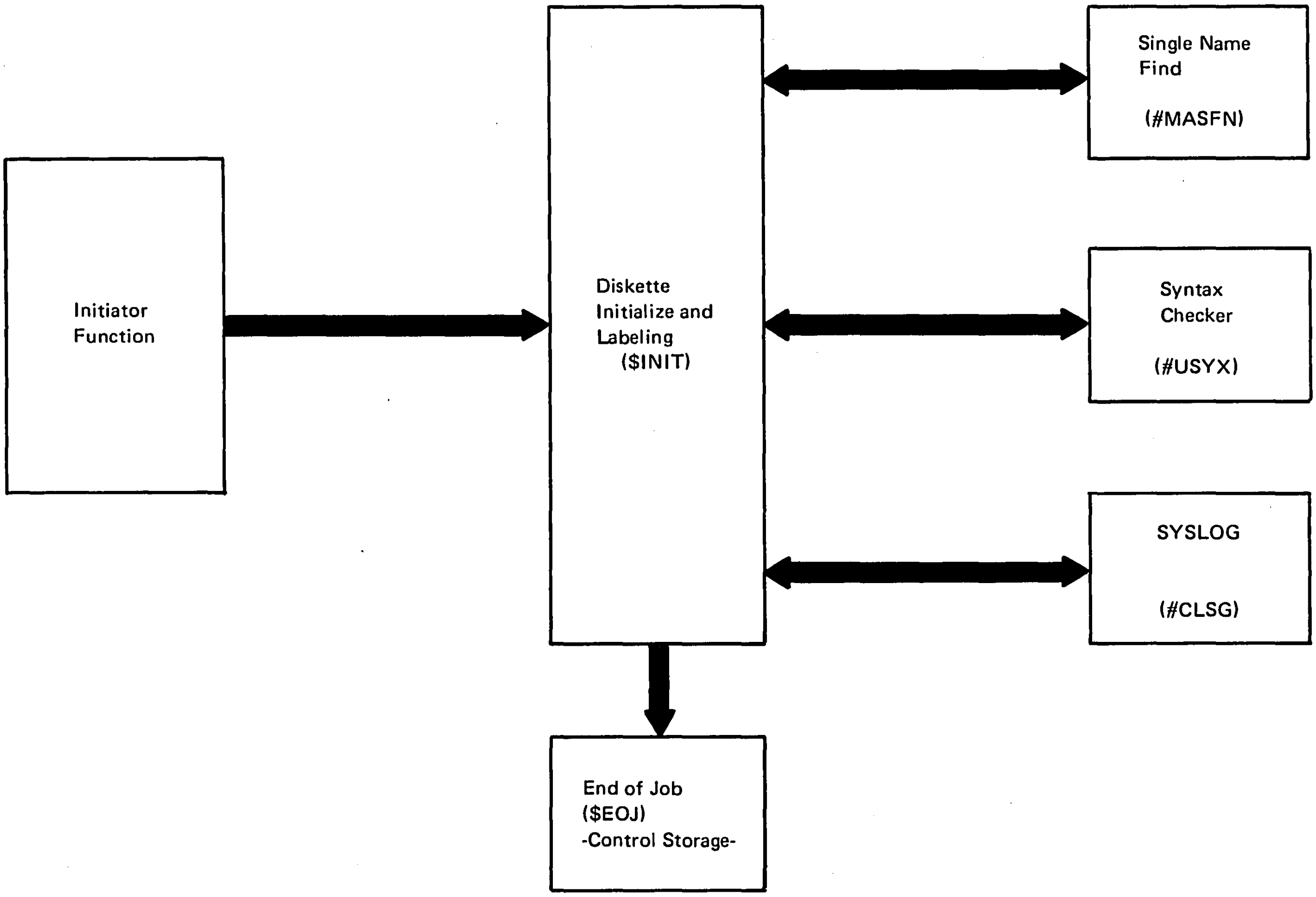

Initiator Function
Diskette Initialize and Labeling ($INIT)
Single Name Find (#MASFN)
Syntax Checker (#USYX)
SYSLOG (#CLSG)
End of Job ($EOJ) -Control Storage-

Figure 18-1. Diskette Labeling and Initialization Utility Control Flow ($INIT)

Data Areas

VERBSUVE, VERBSVE, AND VERBSE VERB LISTS

There are three verb lists for the diskette labeling and initialization utility: VERBSUVE, VERBSVE, and VERBSE. Each time the syntax checker (#USYX) is called, one of the lists must be passed. VERBSUVE is passed on the first call. On each subsequent call, the control statement read determines the verb list that is passed on the next call. The third and fourth bytes of the syntax checker parameter list contain the address of the verb list. Figure 18-2 shows the format and contents of each verb list.

Displacement of Byte	Label	Description
0	VERBSUVE	UIN verb ID
1		VOL verb ID
2		END verb ID
3		X'FF' end of verb list
0	VERBSVE	VOL verb ID
1		END verb ID
2		X'FF' end of verb list
0	VERBSE	END verb ID
1		X'FF' end of verb list

Figure 18-2. VERBSUVE, VERBSVE, and VERBSE Verb Lists

Chapter 19. VTOC Display Utility ($LABEL)

Introduction

The VTOC display utility displays or prints VTOC information from the disk or a diskette.

The disk information displayed or printed is:

- Volume ID and owner ID
- Disk capacity
- One or all of the VTOC entries
- Available disk space

The diskette information displayed or printed is:

- Volume ID and owner ID
- Available diskette space
- One or all of the VTOC entries

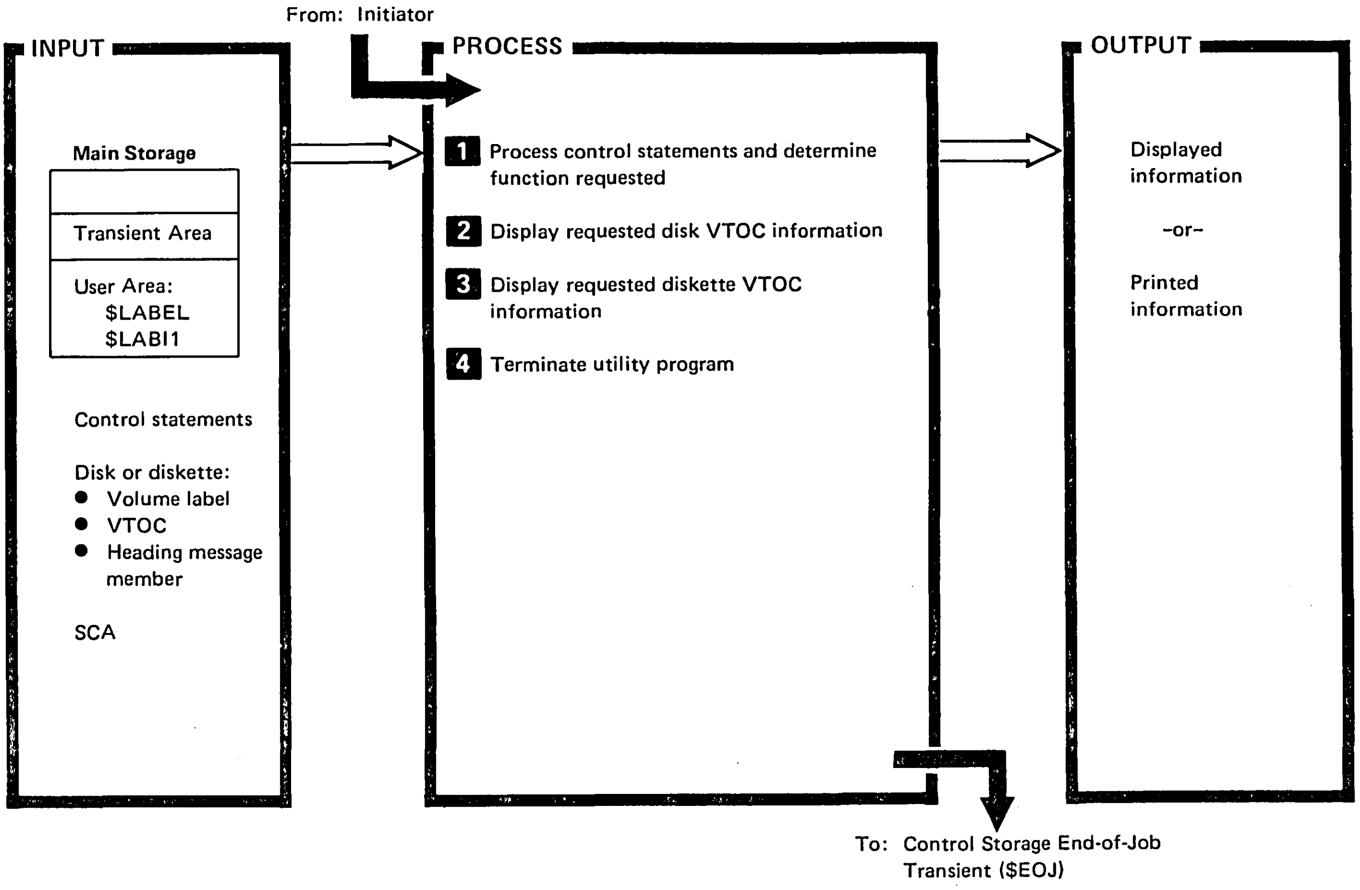
This utility program consists of the disk VTOC display routine ($LABEL), the diskette VTOC display routine ($LABI1), and the syntax specification module ($LABLS). The program resides in the system library.

The VTOC display utility is called by the CATALOG procedure or appropriate OCL statements. (See the *System Support Reference Manual* for more information about calling and executing $LABEL.)

The VTOC display utility requires 14K bytes of main storage for program execution.

Method of Operation

Diagram 19.1 shows the function of the VTOC display utility.


From: Initiator
INPUT
Main Storage
Transient Area
User Area:
$LABEL
$LABI1
Control statements
Disk or diskette:
● Volume label
● VTOC
● Heading message member
SCA
PROCESS
1 Process control statements and determine function requested
2 Display requested disk VTOC information
3 Display requested diskette VTOC information
4 Terminate utility program
OUTPUT
Displayed information
-or-
Printed information
To: Control Storage End-of-Job Transient ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Find syntax checker (#USYX).	#MASFN
Use main storage relocating loader to load #USYX.	$LABEL
Read and syntax check control statement. Determine function requested: ● If display disk VTOC requested, go to 2. ● If display diskette VTOC requested, go to 3.	#USYX
2 Get disk VTOC SSS address from system communication area (SCA) and place in IOB.	$LABEL
Read in system and user format 1s to build table of file start and end SSS addresses (F1TABLE).	Disk IOS
Sort VTOC address table just built (sort by start location).	$LABEL
Set up header information: ● Retrieve heading message.	#MGRET
● Retrieve pack and owner ID.	Disk IOS
● Unpack date. ● Determine disk capacity from SCA.	$LABEL
Display or print heading.	#CLST

Diagram 19.1 (Part 1 of 3). Perform VTOC Dispaly Utility Function ($LABEL)

DESCRIPTION	MODULE/ ROUTINE
If request to display a specific file (LABEL-file label):	$LABEL
● Read specified file entry from F1TABLE.	#CSVF
● Extract appropriate data from format 1.	$LABEL
● Display requested VTOC information on SYSLIST device (display or print).	#CLST
● Check for another file with same name and display if found.	$LABEL
● Calculate available space on disk.	
● Display or print available space amount and location.	#CLST
If request to display all VTOC entries (LABEL-ALL):	$LABEL
▶ ● Read format 1s from F1TABLE.	#CSVF ◀
● Extract appropriate data from format 1.	$LABEL
● Display VTOC information on SYSLIST device (display or print).	#CLST
● Continue processing each VTOC entry until all entries displayed.	$LABEL
● Calculate available space on disk.	
● Display or print available space amount and location.	#CLST
Display error messages as needed.	#CLXS
When last VTOC entry processed, go to 4 to terminate program.	$LABEL
3 Find diskette VTOC display routine ($LABI1).	#MASFN
Use main storage relocating loader (SVC-52) to load $LABI1 into main storage.	$LABEL
Allocate diskette device.	#CAML
Return.	$LABI1
Read VTOC from diskette.	#CSVI
Save diskette format and system date.	$LABI1
Read diskette label track.	Diskette IOS
Move volume ID, owner ID, and system date into headings save area.	$LABI1
Retrieve initial and overflow headings and save for later use.	#MGRET
Build VTOC entry table:	$LABI1
▶ ● Read HDR1s from diskette.	Diskette IOS ◀
● Build entry in VTOC table.	$LABI1
● Convert and place total sectors used by file in table entry for listing.	
● If two sided diskette, process HDR1 on other side.	
● Mask deleted files.	
● Continue processing until all sectors processed.	
Sort VTOC table entries by starting diskette sector addresses.	
Find last entry not deleted to determine available sectors on diskette.	
Display heading lines on SYSLIST device.	#CLST
If no entries to display:	
● Display or print NO ACTIVE FILES message.	
● Display or print END OF DISPLAY message.	
● Go to 4 to terminate program.	$LABI1

Diagram 19.1 (Part 2 of 3). Perform VTOC Display Utility Function ($LABEL)

DESCRIPTION	MODULE/ ROUTINE
If display all files request (LABEL-ALL):	$LABI1
● Move display information for each file to print buffer.	
● Display or print file information.	#CLST
● Continue processing until all files processed.	$LABI1
● Display or print END OF DISPLAY message.	#CLST
● Go to 4 to terminate program.	$LABI1
If display specific file request (LABEL-file label):	
● Find specified file in VTOC table.	
● Move display information to print buffer.	
● Display or print file information.	#CLST
● Search entire table for every file name specified and display all.	
● Display or print END OF DISPLAY message.	
● Go to 4 to terminate program.	$LABI1
Display error messages as needed.	#CLSG
4 Pass control to control storage end-of-job transient ($EOJ) to terminate utility program.	$LABEL or $LABI1

Diagram 19.1 (Part 3 of 3). Perform VTOC Display Utility Function ($LABEL)

Program Organization

Figure 19-1 shows the control flow of the VTOC display utility.

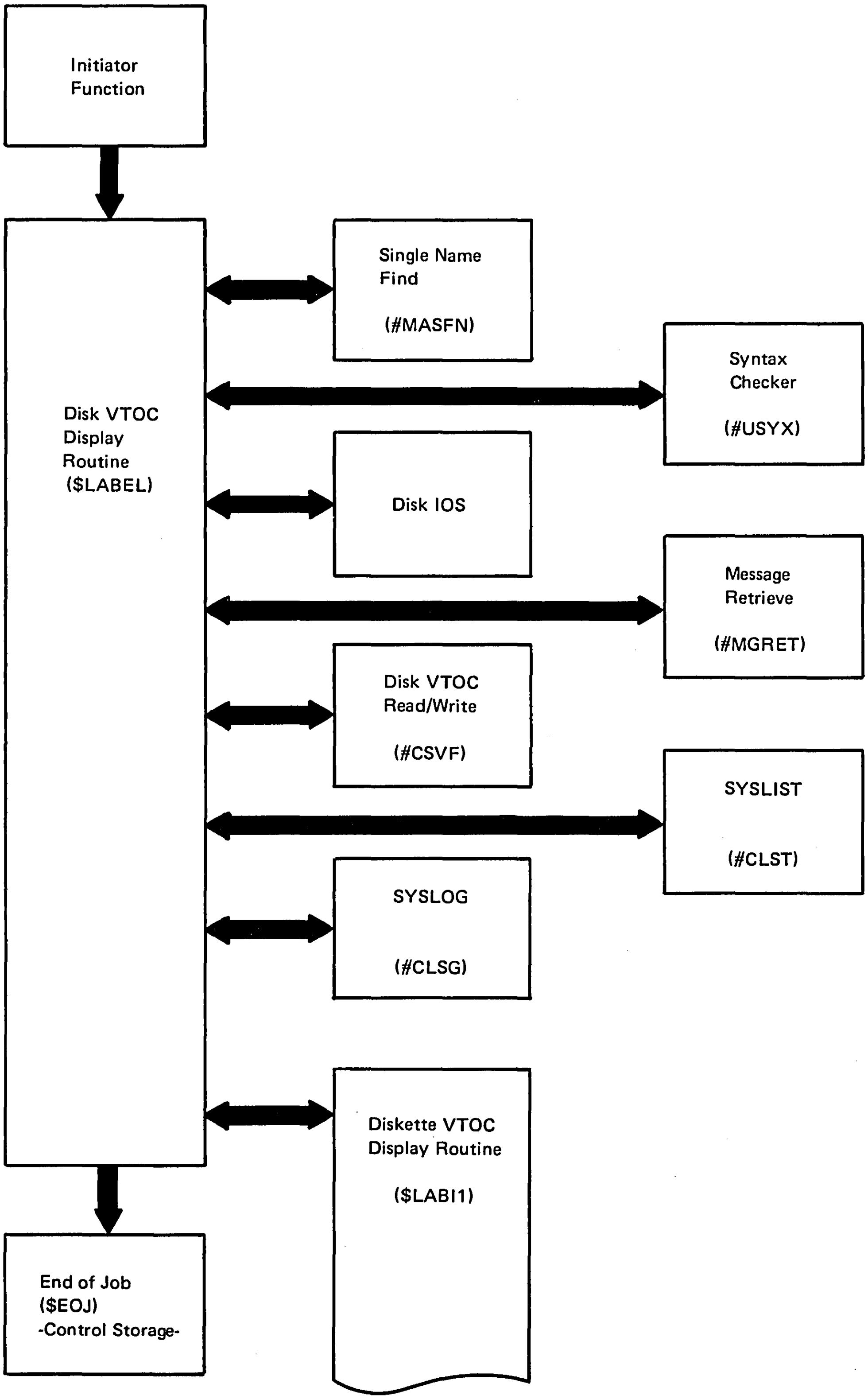

Initiator Function
Disk VTOC Display Routine ($LABEL)
Single Name Find (#MASFN)
Syntax Checker (#USYX)
Disk IOS
Message Retrieve (#MGRET)
Disk VTOC Read/Write (#CSVF)
SYSLIST (#CLST)
SYSLOG (#CLSG)
Diskette VTOC Display Routine ($LABI1)
End of Job ($EOJ) -Control Storage-

Figure 19-1 (Part 1 of 2). VTOC Display Utility Control Flow ($LABEL)

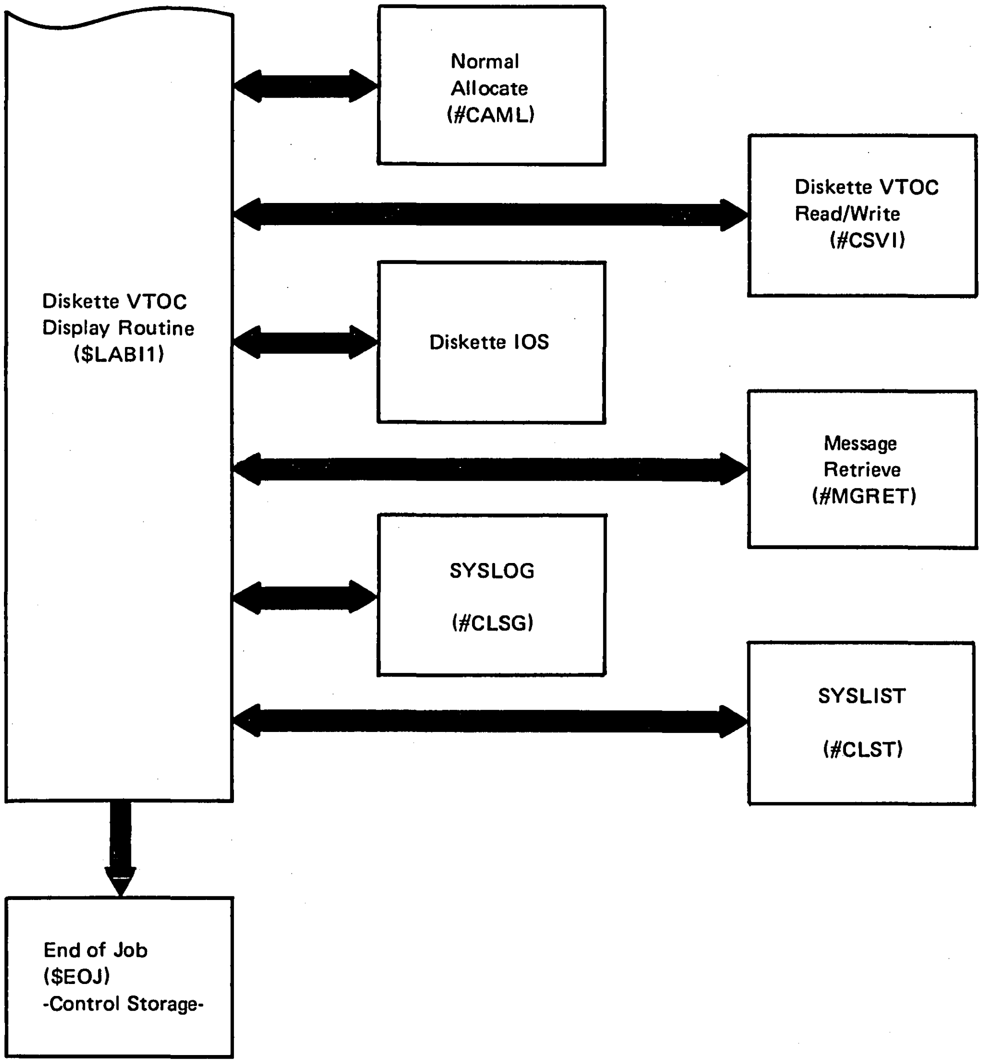


Figure 19-1 (Part 2 of 2). VTOC Display Utility Control Flow ($LABEL)

Chapter 20. Reload Library Utility ($LOADI)

Introduction

The reload library utility reloads the system library from the backup diskette(s) onto disk. The two major functions performed by the utility are the reload command function and the reload library function.

The reload command — reload command routine ($LOADI) ensures #LIBRARY exists on the backup diskette, then it sets the 'IPL-from-diskette' flag in control storage, loads the IPL routine ($IPW), and passes control to the psuedo IPL routine ($IPS) which performs a diskette IPL.

The reload library routine ($LOADI) functions are:

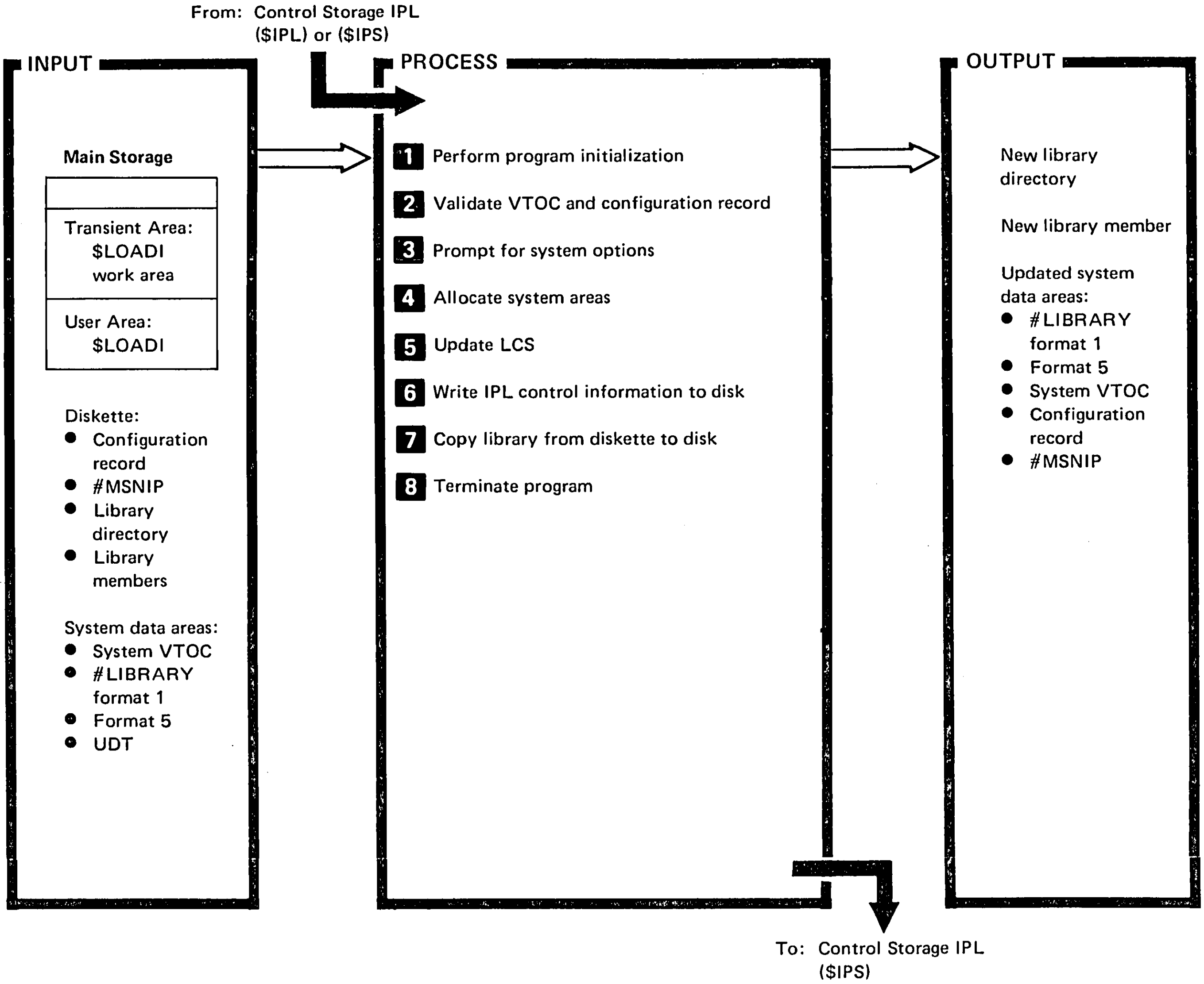
- Determine the master console
- Prepare for diskette operations
- Read the unit definition table (UDT)
- Test for a valid VTOC and configuration record
- Issue operator prompts for system options
- Process the configuration record
- Allocate system areas
- Build volume label, IPL record, and configuration record
- Write library directory and library members to disk
- Pass control to control storage IPL ($IPS) to perform a disk IPL

The reload library utility ($LOADI) is called by the RELOAD procedure or appropriate OCL statements. $LOADI passes control to the pseudo IPL routine ($IPS). $IPS then performs an IPL diskette. This is essentially the same operation performed by setting the MSIPL switch on the CE panel to the Diskette position and pressing the Load key. For more information about calling and executing the reload library utility, see the *System Support Reference Manual.*

The main storage nucleus initialization routine (#MSNIP), the reload library routine ($LOADI), the reload library screen format (##FLOD), and messages must be included on the first backup diskette in order to run the reload library utility. The utility will run dedicated (stand alone) while it is replacing the current system library. The utility requires 30K bytes of main storage for program execution.

Method of Operation

Diagram 20.1 shows the functions of the reload library utility.



From: Control Storage IPL ($IPL) or ($IPS)

INPUT

Main Storage

Transient Area: $LOADI work area

User Area: $LOADI

Diskette:
- Configuration record
- #MSNIP
- Library directory
- Library members

System data areas:
- System VTOC
- #LIBRARY format 1
- Format 5
- UDT

PROCESS

1. Perform program initialization
2. Validate VTOC and configuration record
3. Prompt for system options
4. Allocate system areas
5. Update LCS
6. Write IPL control information to disk
7. Copy library from diskette to disk
8. Terminate program

OUTPUT

New library directory

New library member

Updated system data areas:
- #LIBRARY format 1
- Format 5
- System VTOC
- Configuration record
- #MSNIP

To: Control Storage IPL ($IPS)

DESCRIPTION	MODULE/ ROUTINE
1 Determine master console for reload: ● Test console for errors; if error, wait for request — enter sequence from console. ● Find terminal unit block (TUB) for console. Determine disk extent parameters. Prepare for diskette operations: ● Determine diskette sector size.	$LOADI
● Read diskette volume label.	Diskette IOS
● Ensure proper diskette in use.	$LOADI
Read unit definition table (UDT).	Disk IOS

Diagram 20.1 (Part 1 of 3). Perform Reload Library Function ($LOADI)

DESCRIPTION	MODULE/ ROUTINE
2 Read disk volume label.	Disk IOS
If volume label correct, read configuration record.	
Read system VTOC.	
Verify system format 1s and format 5.	$LOADI
Check format 5 for enough space to expand library.	
Determine if user's VTOC must be moved to new area when system areas are allocated.	
Calculate minimum library requirements for directory.	
Calculate minimum system library size.	
If system is in error, build terminal table from UDT.	
3 Set up prompts based on old system configuration.	
Prompt for: ● Directory sectors. ● Library blocks. ● History file tracks. ● Task work area tracks. ● Delete files from VTOC. ● Use backup configuration record.	Work Station IOCH (WSIOCH)
Accept operator response to prompts.	WSIOCH
Check prompt values received against system data.	$LOADI
4 Initialize library directory area to X'FF'.	
Initialize history file.	
If delete file request, initialize user VTOC.	
Create #LIBRARY format 1.	
Create format 1s for system files in system VTOC (SYSWORK, SYSHIST, SYSTASK).	
Calculate new format 5 based on new library size.	
Write system format 1s and format 5 to disk.	Disk IOS
5 Update library control sector (LCS) fields based on operator requested #LIBRARY space allocation.	$LOADI
Build volume label: ● Put label ID, owner ID, and volume name in volume label.	
● Write volume label to disk.	Disk IOS
6 Set configuration values: ● Disk size option. ● Main storage size. ● Control storage size. ● Security flag if security file on disk.	$LOADI
Write #MSNIP and configuration record to disk.	Disk IOS
Write terminal information table to disk.	

Diagram 20.1 (Part 2 of 3). Perform Reload Library Function ($LOADI)

DESCRIPTION	MODULE/ ROUTINE
7 Copy library directory from diskette to disk:	$LOADI
● Read backup library directory into I/O buffer.	Diskette IOS
● Return.	$LOADI
● Write library directory from I/O buffer to disk.	Disk IOS
Copy library members from diskette to disk:	$LOADI
● Read #LIBRARY members into I/O buffer.	Diskette IOS
● Return.	$LOADI
● Write members from I/O buffer to disk.	Disk IOS
● Return.	$LOADI
● Issue message to change diskette if necessary.	WSIOCH
8 Issue reload complete message to operator.	
Initiate pseudo disk-IPL: ● Evoke control storage loader function. ● Load and pass control to pseudo control storage IPL ($IPS).	$LOADI

Diagram 20.1 (Part 3 of 3). Perform Reload Library Function ($LOADI)

Program Organization

Figure 20-1 shows the control flow of the reload library utility.

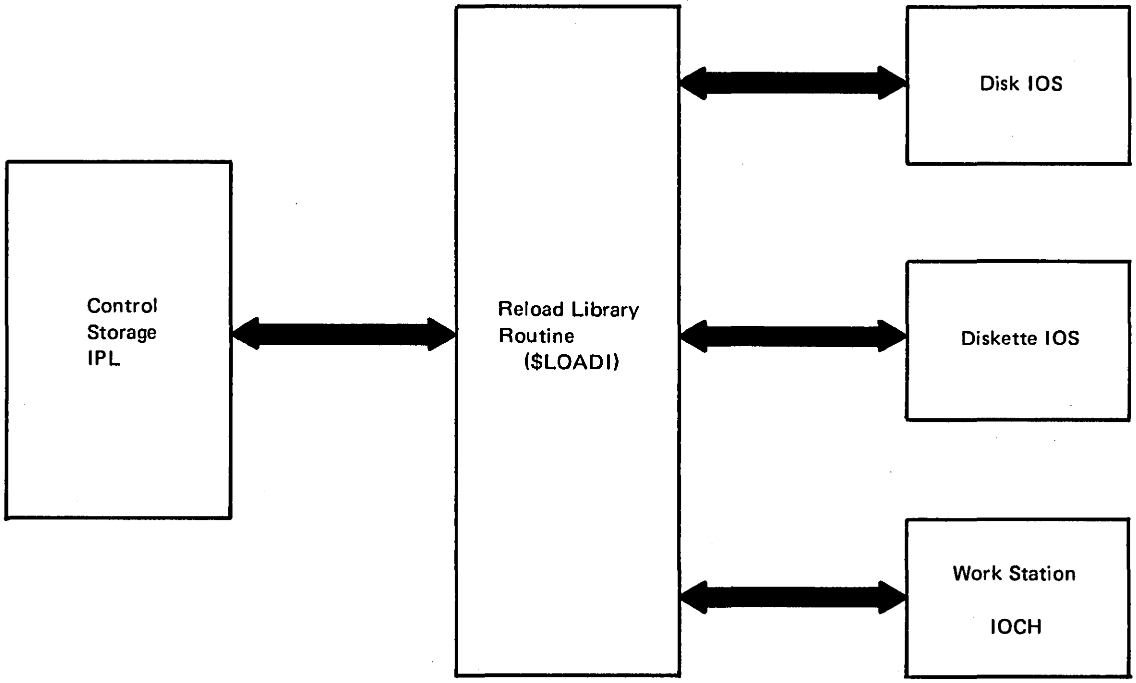


Figure 20-1. Reload Library Routine Control Flow ($LOADI)

Data Areas

$LOADI COMMUNICATION AREA

The $LOADI communication area is a 28-byte area that is read from the backup diskette into location X'4FCO' in storage. It is created by the backup utility and contains information that is used by the reload library utility in reloading the system library from the backup diskettes. Figure 20-2 shows the format and contents of the area.

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes	Description
0	PDIRSECT	1	Number of active disk sectors in the directory
1	PMEMSECT	3	Number of active member disk sectors
4	PLIBIND	1	Reserved
5	PCDATE	3	Backup file creation date
8	PARMRSVD	3	Reserved
B	PFILNAM	8	Backup file name
13	PDIRSPC	3	Number of sectors allocated to the directory
16	PLIBSPC	3	Number of sectors allocated to the library
19	PSCPSIZE	3	Number of sectors in the SSP base

Figure 20-2. $LOADI Communication Area

Chapter 21. Library Maintenance Utility ($MAINT)

Introduction

The library maintenance utility ($MAINT) performs the following four major functions:

- Allocates libraries
  - Allocates user libraries
  - Changes member size of any library
  - Changes directory size of a user library
- Compresses a library
- Deletes members from a library
- Copies library members:
  - Copies members from a library to a file
  - Copies members from a library to a library
  - Copies members, member names, or library status to a display
  - Copies member names from a file to display
  - Copies member from a file to a library
  - Copies members from the reader (SYSIN) to a library

The library maintenance program consists of the following driver modules that reside in the system library:

- Library maintenance mainline ($MAINT)
- Library specification ($MASPC)
- Library allocate ($MALOC)
- Library compress ($MARCK)
- Library compress – phase 2 ($MAR2K)
- Library compress – phase 3 ($MAR3K)
- Library delete ($MADLT)
- Sector mode file display ($MASDF)
- Record mode file display ($MARPF)
- Library print routine ($MADSP)
- Sector mode library to file copy ($MATFS)
- Record mode library to file copy ($MARTF)
- Library to library copy ($MALTL)
- Sector mode file to library copy ($MATLS)
- System/32 sector mode file to library copy ($MAF32)
- Record mode file to library copy ($MARFF)
- Reader to library copy ($MARDR)

Note: Additional librarian facility programs are described in Chapter 6, *System Service.*

$MAINT is the mainline module for the library maintenance utility. It is loaded and given control when the system operator enters the LOAD $MAINT, RUN OCL statements. It runs in a 14K region with a logical start address of X'C800'.

Utility control statements following the RUN statement request the various functions of the library maintenance utility. $MAINT reads and checks the utility control statements by calling the syntax checker (#USYX). $MAINT builds a library common area (CNTLBMSG) from information supplied by the utility control statements, moves the common area over the first sector of itself, and loads the appropriate driver module after the common area.

The loaded module performs the requested function (loading other modules as required) and then overlays itself with the $MAINT module. $MAINT then processes the next utility control statement.

The following is a description of the subfunctions performed by the allocate, compress, delete, and copy functions of the library maintenance utility.

The data areas section of this chapter contains applicable storage usage maps. The first map shows the region when the $MAINT module is executing. For each unique function there is a corresponding storage usage map depicting the region when the requested function is being performed.

Allocate Function ($MALOC):

- Creates a user library with specified size
- Increases or decreases the size of an existing library
- Changes the directory size of a user library (RELOAD is used to change the directory size of the system library)

Compress Function ($MARCK, $MAR2K and $MAR3K):

- Removes gaps in library member space and updates library directory
- Updates system pointers affected by the compress function

Delete Function ($MADLT):

- Deletes non-SSP members, or all non-SSP members having a certain name
- Deletes non-SSP members of one type, or all types having names beginning with certain characters
- Deletes all non-SSP members of one type or all types
- Deletes non-SSP members of one type or all types except members having certain names or having names beginning with certain characters
- Deletes specified members, including SSP members

Copy Function:

- Reader to library ($MARDR) – replaces or adds an S or P member from the system input device
- Library to library ($MALTL):
  - Copies a member of one library type or all library types within a library or between libraries, optionally changing the member's name in the process
  - Copies members of one type or all types within a library or between libraries that have names beginning with certain characters
  - Copies members of one type or all types, omitting members having a certain name or having names beginning with certain characters or omitting all SSP members
  - Copies all members of one type or all types from one library to another (if type is all, only non-SSP members are copied)
- Library to file ($MATFS or $MARTF):
  - Copies to a file a member having a certain name, or all members with that name
  - Copies to a file members of one type, or all types having names beginning with certain characters
  - Copies to a file all members of one type
  - Copies to a file all members of all types, optionally omitting members having a certain name or having names beginning with certain characters (not including SSP members)
  - Copies to a file members of one type, omitting members having a certain name or having names beginning with certain characters, or omitting all SSP members
  - Copies to a file all members with a PTF applies (only sector mode – $MATFS)
  - Adds members to an existing file of library members
  - Copies members to a basic exchange diskette file (only record mode – $MARTF)

- File to library ($MATLS, $MARFF, or $MAF32):
  - Copies members in a file to a library
  - Copies members with special PTF log numbers from a file to a library (only sector mode – $MATLS and $MAF32)
  - Copies members in a basic exchange diskette file directly to a library (only record mode – $MARFF)
  - Copies duplicates of existing members to a library *Note:* When members with PTFs are copied to a library, the PTF is logged in the PTFLOG in that library.

- File to printer ($MASDF or $MARPF) – prints the type and name of all members in a librarian file

- Library to printer ($MADSP):
  - Prints a member or all members having a certain name
  - Prints all members of a certain type
  - Prints members of one type, or all types having names beginning with certain characters
  - Prints members of one type or all types, omitting members having certain names or having names beginning with certain characters, or omitting all SSP members
  - Prints directory entries for members of one type
  - Prints directory entries for members of one type or all types, omitting entries having certain names or having names beginning with certain characters or omitting all entries for SSP members
  - Prints all directory entries and the library status
  - Prints the library status

After completing processing, the driver module reloads $MAINT over the library common area and itself. $MAINT reads the next utility control statement and determines the function to perform. When it reads an END statement, $MAINT calls the end-of-job transient ($EOJ) to terminate the job step.

The library maintenance utility is called by the appropriate procedure or OCL statements. (See the *System Support Reference Manual* for more information about calling and executing $MAINT.)

Method of Operation

Diagrams 21.1 through 21.7 show the functions of the library maintenance utility.

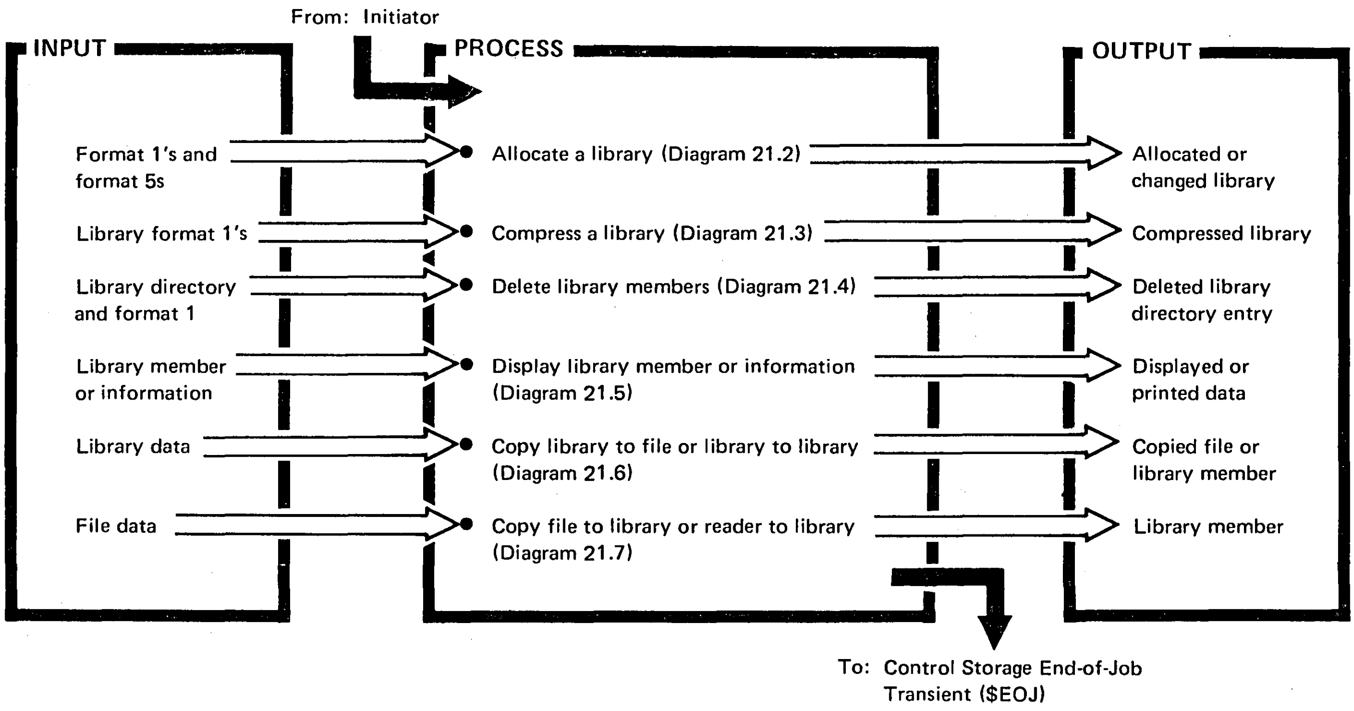
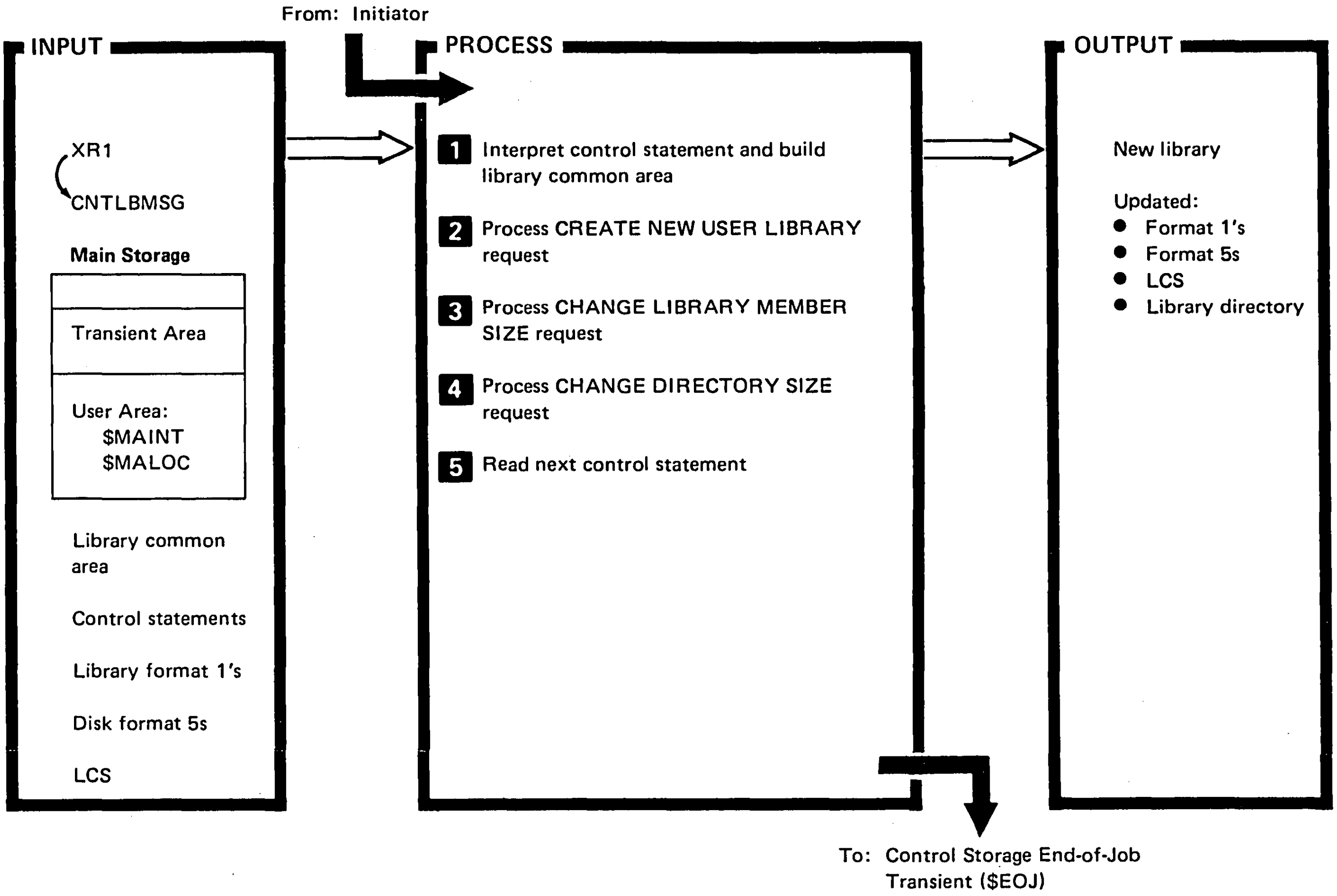

From: Initiator
INPUT
PROCESS
OUTPUT
Format 1's and format 5s
Library format 1's
Library directory and format 1
Library member or information
Library data
File data
Allocate a library (Diagram 21.2)
Compress a library (Diagram 21.3)
Delete library members (Diagram 21.4)
Display library member or information (Diagram 21.5)
Copy library to file or library to library (Diagram 21.6)
Copy file to library or reader to library (Diagram 21.7)
Allocated or changed library
Compressed library
Deleted library directory entry
Displayed or printed data
Copied file or library member
Library member
To: Control Storage End-of-Job Transient ($EOJ)

Diagram 21.1. Overview of Library Maintenance Utility



From: Initiator

INPUT

XR1 → CNTLBMSG

Main Storage

Transient Area

User Area:
$MAINT
$MALOC

Library common area

Control statements

Library format 1's

Disk format 5s

LCS

PROCESS

1 Interpret control statement and build library common area

2 Process CREATE NEW USER LIBRARY request

3 Process CHANGE LIBRARY MEMBER SIZE request

4 Process CHANGE DIRECTORY SIZE request

5 Read next control statement

OUTPUT

New library

Updated:
- Format 1's
- Format 5s
- LCS
- Library directory

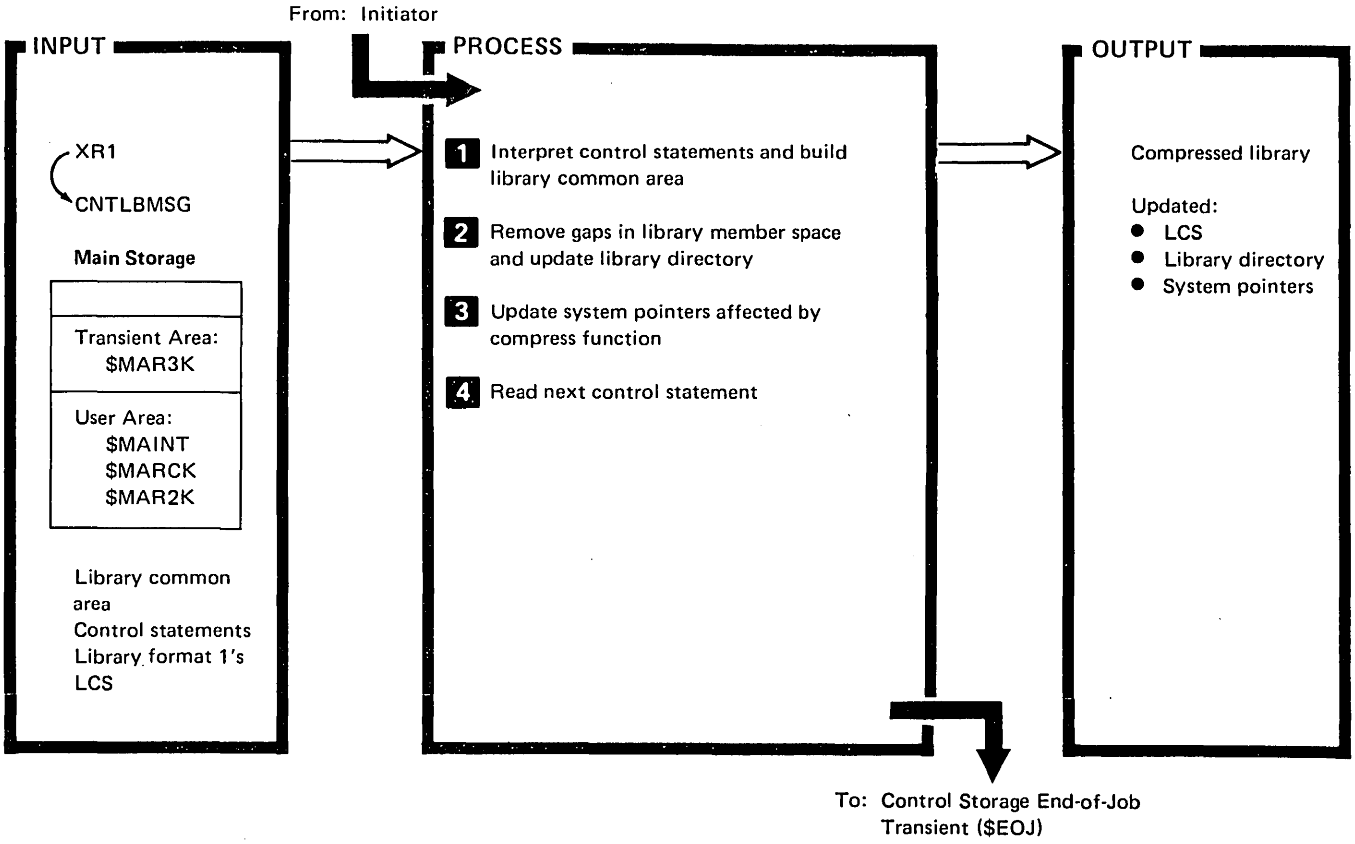
To: Control Storage End-of-Job Transient ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Read and syntax check utility control statement.	#USYX
If END statement, call control storage end-of-job ($EOJ).	$MAINT
Process LIBRNAME parameter:	
● Ensure library exists (if CREATE not given).	#MAFLB
● Save library format 1 address.	$MAINT
Move save area (library common area) to beginning of $MAINT (X'C800').	
Load library allocate module ($MALOC) following save area (X'C900').	
2 Ensure specified library name is valid.	$MALOC
Ensure specified directory size is valid.	
Allocate disk space.	#CAS1
Return.	$MALOC
Issue error message if file name already exists or insufficient space.	#CLXS
Note: Format 1 and VTOC format 5 updated by #CAS1.	$MALOC

Diagram 21.1 (Part 1 of 2). Perform Library Maintenance — Allocate Function ($MAINT)

DESCRIPTION	MODULE/ ROUTINE
Move new file specification block (FSB) from file FSB chain to library FSB chain.	$MALOC
Initialize library control sector (LCS).	
Write LCS to library.	Disk IOS
Return.	$MALOC
Write format 1 to VTOC.	@CSVF
Initialize entire directory to X'FF's.	$MALOC
Write directory, PTFLOG directory entry, and one PTFLOG sector to disk.	Disk IOS
3 Read LCS for library to be changed.	
Return.	$MALOC
Compare directory if needed.	$MACMP
Ensure size change requested is within acceptable range.	$MALOC
If size increase request:	
● Attempt to allocate space after library.	#CAS1
● Return.	$MALOC
● Write LCS back to disk.	Disk IOS
● Return.	$MALOC
● Write format 1 to VTOC.	@CSVF
● Return.	$MALOC
● Issue halt if space not available.	#CLXS
If size decrease request:	$MALOC
● Read format 5s from disk.	Disk IOS
● Return.	$MALOC
● Update and write format 5s back to disk.	Disk IOS
● Return.	
● Update and write format 1's to disk.	@CSVF
● Return.	$MALOC
● Update LCS and write to library.	Disk IOS
● Return.	$MALOC
● Issue error message if not able to decrease library size.	#CLXS
4 If directory size decrease request:	$MALOC
● Ensure size decrease possible.	
● Update relative disk address in each directory entry to agree with new start address.	
● Write updated directory to disk.	Disk IOS
● Return.	$MALOC
● Write LCS to disk.	Disk IOS
● Return.	$MALOC
● Issue error message if not able to decrease size.	#CLXS
If directory size increase request:	$MALOC
● Attempt to allocate space ahead of library.	#CAS1
● Return.	$MALOC
● Update LCS and write to disk.	Disk IOS
● Return.	$MALOC
● Write format 1's to VTOC.	@CSVF
● Return.	$MALOC
● Shift directory (left) to new location.	$MACMP
● Update disk address relative to directory start.	
● Return.	$MALOC
● Issue halt if space not available.	#CLXS

Diagram 21.1 (Part 2 of 2). Perform Library Maintenance — Allocate Function ($MAINT)



1 Interpret control statements and build library common area

2 Remove gaps in library member space and update library directory

3 Update system pointers affected by compress function

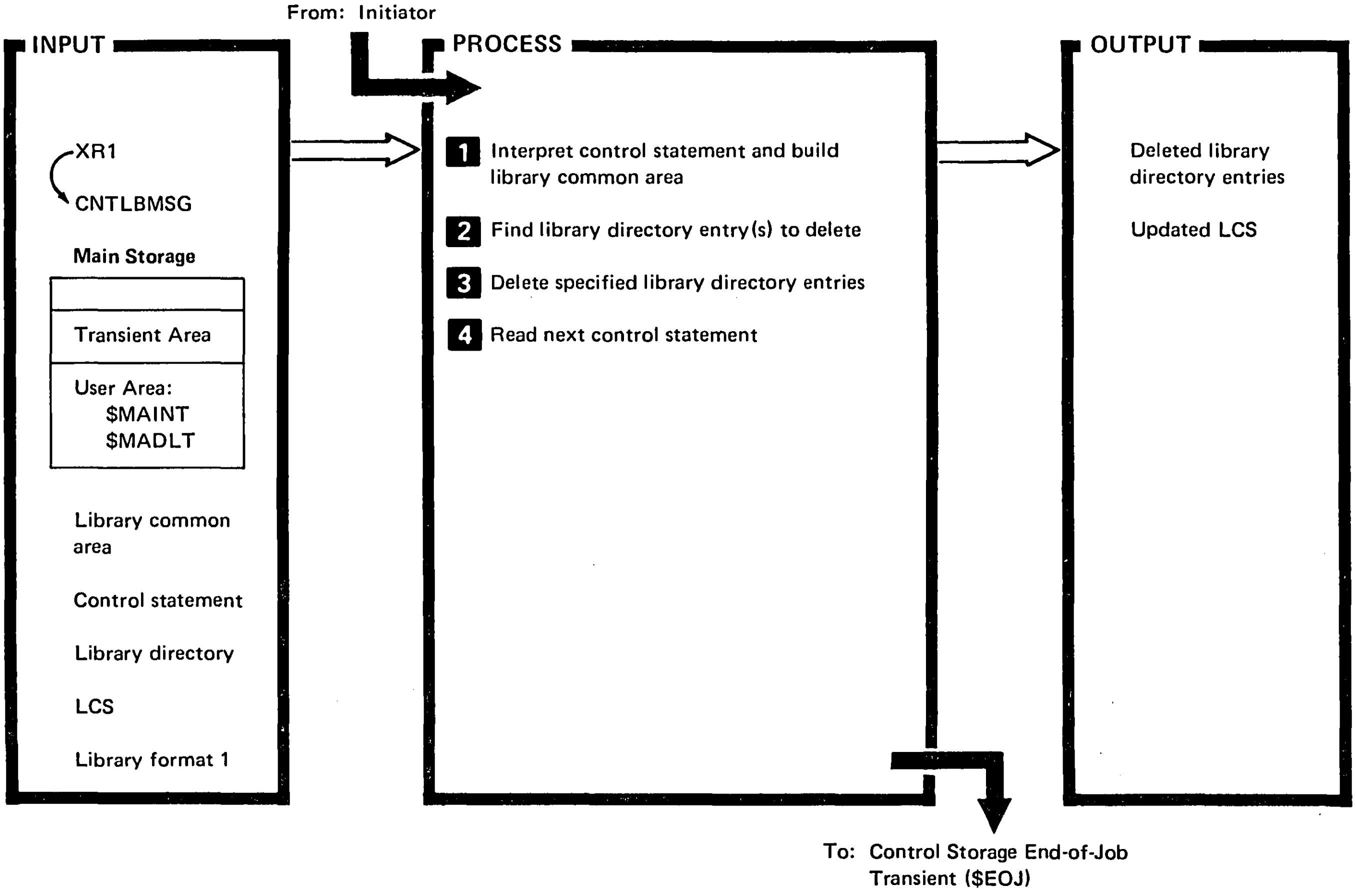
4 Read next control statement

DESCRIPTION	MODULE/ ROUTINE
1 Read and check syntax of utility control statement.	#USYX
If END statement, call control storage end-of-job ($EOJ).	$MAINT
Process LIBRNAME parameter:	
● Ensure library exists.	#MAFLB
● Save library format 1 address.	$MAINT
Move save area (CNTLBMSG) to beginning of $MAINT (X'C800').	
Load library compress module ($MARCK) following save area (X'C900').	
2 Ensure exclusive use of library to be compressed (first pass only).	$MARCK
If #LIBRARY, ensure dedicated system. Suspend command processor	$MAR3K
Read library control sector (LCS).	Disk IOS
Run compactor to remove deleted directory entries.	$MACMP
Build matrix table from library directory information.	$MARCK
Build hole table from matrix table information.	
Move library member sectors across gaps indicated in hole table.	Disk IOS

Diagram 21.2 (Part 1 of 2). Perform Library Maintenance – Compress Function ($MAINT)

DESCRIPTION	MODULE/ ROUTINE
Update LCS to show regained member space.	$MARCK
Update library directory (relative addresses) using hole table information.	
Write LCS and directory back to disk.	Disk IOS
Load library compress phase 2 ($MAR2K) over $MARCK (X'C900').	$MARCK
3 Update message member addresses in job control block (JCB).	$MAR2K
Update menu member pointers in JCB.	
If any procedures active, update message member and procedure disk addresses in program procedure save area (PPSA), using hole table information.	
If not #LIBRARY, go to 3 A.	
Update SSP message member addresses in system communication area (SCA).	
Update format index tables and WTG tables in SSP modules.	#MAXRF
Update system transient table in control storage.	$MAR2K
A If second pass needed, reload $MARCK and go to 2.	
Dequeue library just compressed.	
If #LIBRARY, restart command processor.	
4 Load $MAINT back into user area at location X'C800'.	
Read next control statement.	$MAINT
When END read, call end-of-job ($EOJ).	

Diagram 21.2 (Part 2 of 2). Perform Library Maintenance — Compress Function ($MAINT)

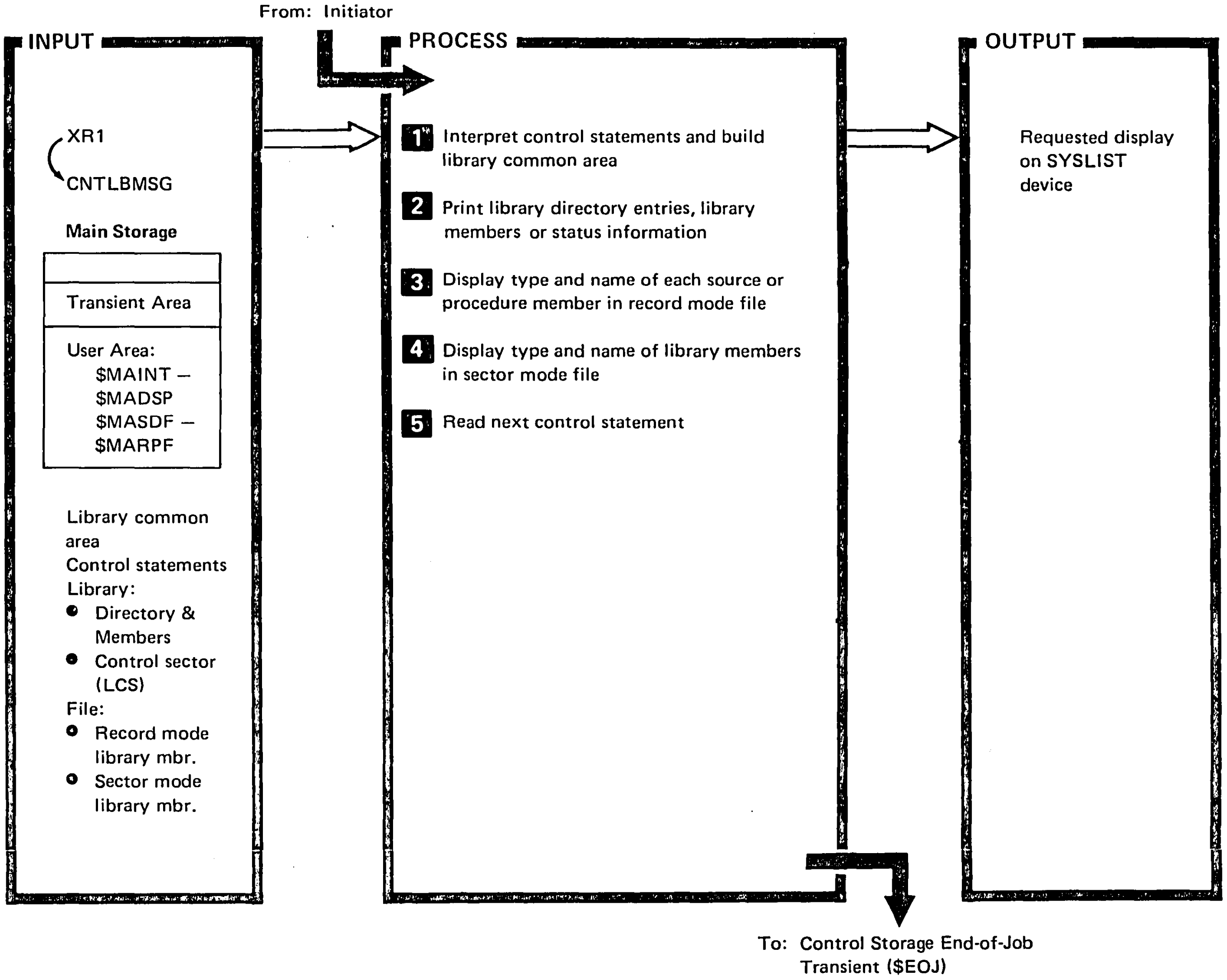

From: Initiator
INPUT
XR1
CNTLBMSG
Main Storage
Transient Area
User Area:
$MAINT
$MADLT
Library common area
Control statement
Library directory
LCS
Library format 1
PROCESS
1 Interpret control statement and build library common area
2 Find library directory entry(s) to delete
3 Delete specified library directory entries
4 Read next control statement
OUTPUT
Deleted library directory entries
Updated LCS
To: Control Storage End-of-Job Transient ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Read and syntax check utility control statements.	#USYX
If END statement, call end of job ($EOJ). Process LIBRNAME parameter:	$MAINT
● Ensure library exists.	#MAFLB
● Save library format 1 address. Encode and save NAME and OMIT parameters. Move save area (CNTLBMSG) to beginning of $MAINT (X'C800'). Load library delete module ($MADLT) following save area (X'C900').	$MAINT
If deleting ALL or SSP from #LIBRARY, ensure proper dedication **2** If deleting ALL from a user library, ensure only one user.	$MADLT
Read library control sector (LCS) from disk.	Disk IOS
Load librarian find routine ($MALFN) into main storage.	$MADLT

Diagram 21.3 (Part 1 of 2). Perform Library Maintenance — Delete Function ($MAINT)

DESCRIPTION	MODULE/ ROUTINE
Determine and save information for librarian find routine: ● Library type (O, R, S, or P). ● Library name, partial name, or ALL. ● If SSP modules. ● OMIT parameter, if any.	$MADLT
Find directory entries of modules to be deleted and pass entry addresses back to $MADLT.	$MALFN
3 Check returned entry for delete request. Delete module if delete specified: ● Set total number of sectors in directory entry to zero to show entry deleted. ● Update active and available directory entry count.	$MADLT
● Delete PTF information if necessary.	$MAPTF
Write LCS back to disk.	Disk IOS
Issue error message if unable to delete requested module.	#CLXS
4 Load $MAINT back into user area at location X'C800'.	$MADLT
Read next control statement. When END read, call control storage end of job ($EOJ).	$MAINT

Diagram 21.3 (Part 2 of 2). Perform Library Maintenance — Delete Function ($MAINT)



From: Initiator

INPUT

XR1

CNTLBMSG

Main Storage

Transient Area

User Area:
$MAINT –
$MADSP
$MASDF –
$MARPF

Library common area

Control statements

Library:
- Directory & Members
- Control sector (LCS)

File:
- Record mode library mbr.
- Sector mode library mbr.

PROCESS

1 Interpret control statements and build library common area

2 Print library directory entries, library members or status information

3 Display type and name of each source or procedure member in record mode file

4 Display type and name of library members in sector mode file

5 Read next control statement

OUTPUT

Requested display on SYSLIST device

To: Control Storage End-of-Job Transient ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Read and syntax check utility control statement.	#USYX
If END statement, call end-of-job ($EOJ).	$MAINT
Process LIBRNAME parameter (if displaying from a library):	
● Ensure library exists.	#MAFLB
● Save library format 1 address.	$MAINT
Encode and save NAME and OMIT parameters (if any).	
If RECL parameter given, set record mode switch and if not given, set record length to 8.	
If copy from library:	
● Move save area (CNTLBMSG) to beginning of $MAINT (X'C800').	
● Load library print routine ($MADSP) following save area (X'C900').	
● Go to **2**.	

Diagram 21.4 (Part 1 of 4). Library Maintenance – Display Function ($MAINT)

DESCRIPTION	MODULE/ ROUTINE
If copy from file: ● Determine if device is disk or diskette.	$MAINT
● Allocate specified file.	#CMAL
● Move save area (CNTLBMSG) to beginning of $MAINT (X'C800'). ● If record mode, load record mode file display routine ($MARPF) following save area (X'C900') and go to 3. ● If sector mode, load sector mode file display routine ($MASDF) following save area (X'C900') and go to 4.	$MAINT
2 Load librarian find routine ($MALFN) into main storage. Load SYSLIST (#CLSP or #CLSW). Allocate printer (#CLST calls printer allocate).	$MADSP
Read configuration record to get print image.	Disk IOS
Return.	$MADSP
Retrieve all headings.	#MGRET
Get date for headings from job control block (JCB) and get time. If request to print status information:	$MADSP
● Get library control sector (LCS).	Disk IOS
▶ ● Compact library directory if necessary.	$MACMP ◀
● Return.	$MADSP
● Get status information heading messages and put in print buffer.	#MGRET
● Convert LCS data to printable data.	$MADSP
● Print library status information.	#CLSP or #CLSW
If request to print directory entries:	$MADSP
● Find directory entries to display.	$MALFN
● Put directory entries in work area and indicate names to omit. ● Convert hexadecimal to printable characters.	$MADSP
● Print headings and specified directory entries.	#CLSP or #CLSW
If request to print library members:	$MADSP
● Find directory entries to display. ● Put directory entries in work area and indicate names to omit. ● Format directory entries to printable characters.	$MAFLN
● Print specified directory entries.	#CLSP or #CLSW
● If module is type O or R, convert binary data to dump-format.	$MADSP
● If module is type S or P, get source or procedure (S or P) modules.	@MASYL
● Print specified library members. *Note:* Directory entry is printed followed by associated library member. Then next directory entry and member.	#CLSP or #CLSW
Display messages as needed.	#CLXS
Go to 5 to read next control statement.	$MADSP
3 Load SYSLIST (#CLSP or #CLSW).	$MARPF
Retrieve heading messages. Get date and time.	#MGRET

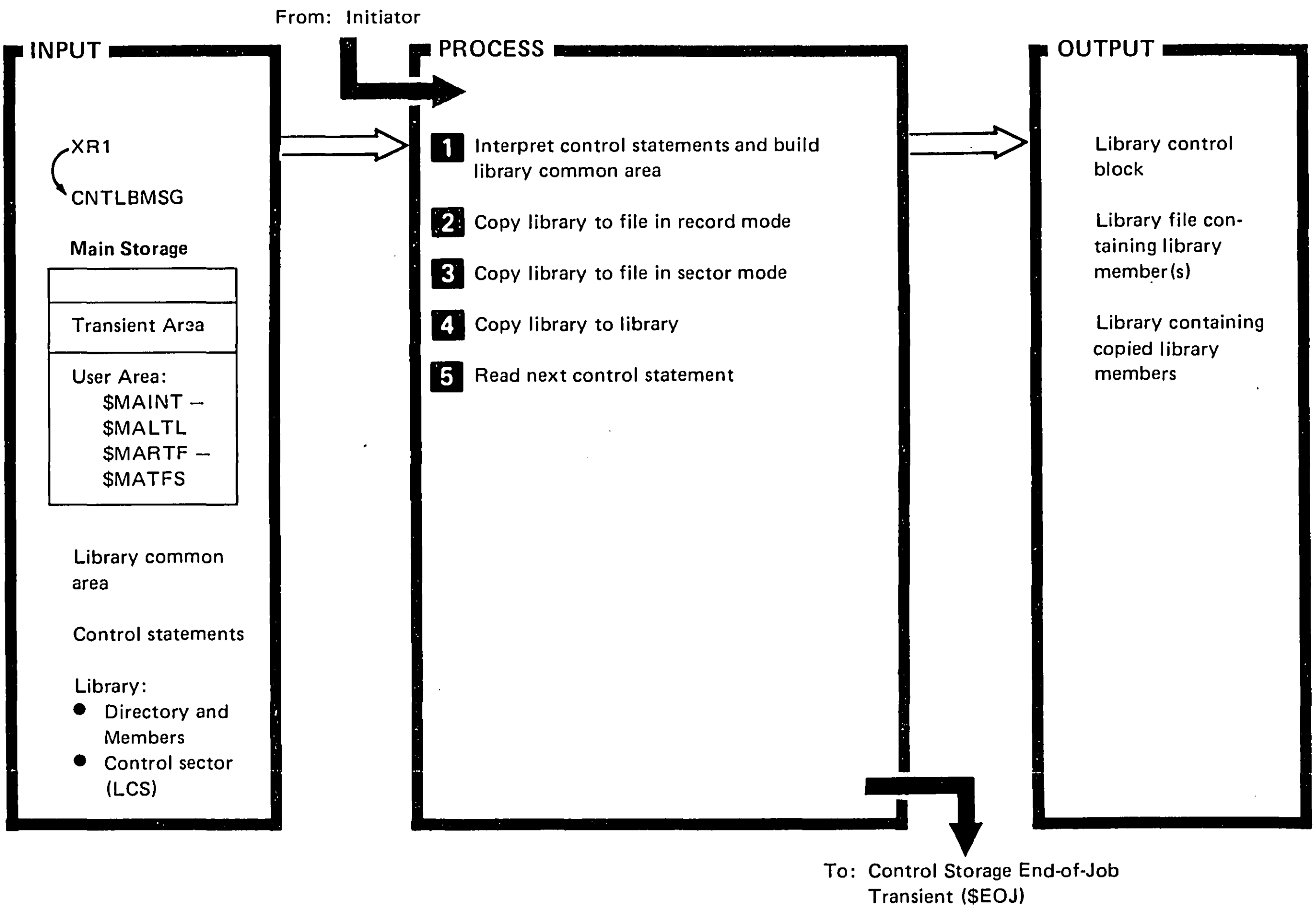
Diagram 21.4 (Part 2 of 4). Library Maintenance – Display Function ($MAINT)

DESCRIPTION	MODULE/ ROUTINE
▶ Determine device type (diskette or disk): ● If diskette, load diskette data management (#DRDM) into main storage. ● If disk, disk data management (#DDDM) is resident.	$MARPF ◀
Open input file.	#DMOP
Print headings previously retrieved.	#CLSP or #CLSW
Read COPY record from input file (diskette or disk).	#DRDM or #DDDM
Ⓐ Check record for valid copy control statement.	$MARPF
Print type and name from copy control statement.	#CLSP or #CLSW
Return.	$MARPF
Read records from file (diskette or disk) until CEND control record is read.	#DRDM or #DDDM
If another COPY control statement is read, return to 3 Ⓐ.	$MARPF
Close input file.	#DMCL
Go to 5 to read next control statment.	$MARPF
4 Load SYSLIST (#CLSP or #CLSW).	$MASDF
Retrieve heading messages. Get date and time	#MGRET
▶ Determine device type (diskette or disk): ● If diskette, load diskette sector data management (#DRSM) into main storage. ● If disk, load disk sector data management (#DDSM) into main storage.	$MASDF ◀
Open input file.	#DMOP
Return.	$MASDF
Read data from specified file and place in output buffer.	#DRSM or #DDSM
Put file name in headings.	$MASDF
Print headings.	#CLSP or #CLSW
Find next control record and check validity.	$MASDF

Diagram 21.4 (Part 3 of 4). Library Maintenance – Display Function ($MAINT)

DESCRIPTION	MODULE/ ROUTINE
Display type and name.	#CLSP or #CLSW
Return.	$MASDF
When end-of-file is reached, close file.	#DMCL
Go to 5 to read next control statement.	$MASDF
5 Load $MAINT back into user area at location X'C800'.	
Read next control statement	$MAINT
When END read, control storage call end of job ($EOJ).	

Diagram 21.4 (Part 4 of 4). Library Maintenance — Display Function ($MAINT)



From: Initiator

INPUT

XR1 → CNTLBMSG

Main Storage

Transient Area

User Area:
$MAINT –
$MALTL
$MARTF –
$MATFS

Library common area

Control statements

Library:
- Directory and Members
- Control sector (LCS)

PROCESS

1 Interpret control statements and build library common area

2 Copy library to file in record mode

3 Copy library to file in sector mode

4 Copy library to library

5 Read next control statement

OUTPUT

Library control block

Library file containing library member(s)

Library containing copied library members

To: Control Storage End-of-Job Transient ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Read and syntax check utility control statements.	#USYX
If END statement, call control storage end-of-job ($EOJ).	$MAINT
Process FROM parameter: ● Ensure library exists.	#MAFLB
● Save library format 1 address. Encode and save NAME and OMIT parameters (if any). If RECL parameter given, set record mode switch and if not given, set record length to 8. Determine if device is disk or diskette. If copy from library to file: ● Determine if copy to new file or add to existing file.	$MAINT
▶ ● Allocate file.	#CAML ◀
● Ensure RECL valid (if applicable).	$MAINT
● Issue error messages if necessary.	#CLXS
● Move save area (CNTLBMSG) to beginning of $MAINT (X'C800'). ● If record mode, load record mode from library to file copy routine ($MARTF) following save area (X'C900') and go to **2**.	$MAINT

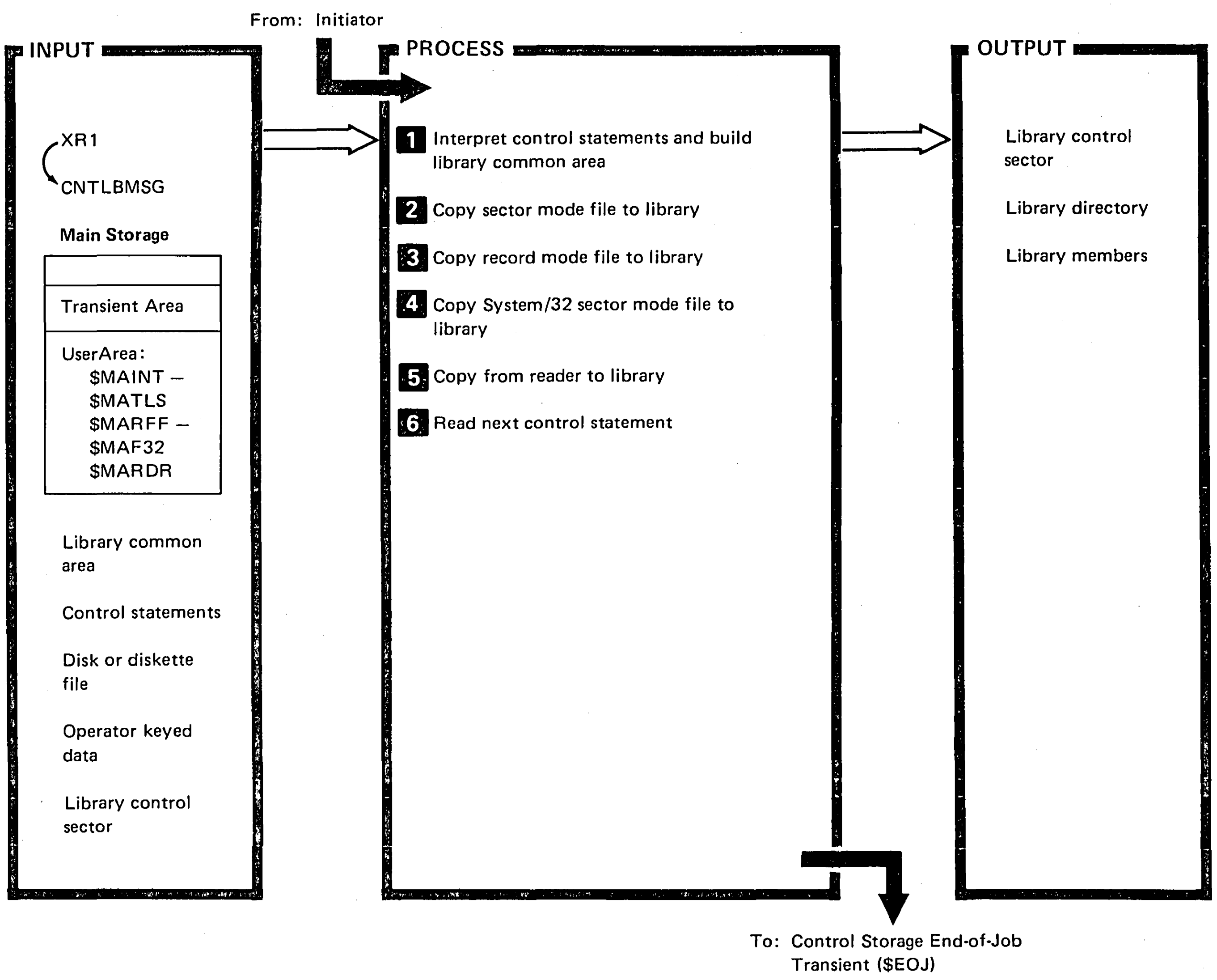
Diagram 21.5 (Part 1 of 3). Perform Library Maintenance – Copy FROM – Library Functions ($MAINT)

DESCRIPTION	MODULE/ ROUTINE
● If sector mode, load sector mode from library to file copy routine ($MATFS) following save area (X'C900') and go to **3** . If copy from library to library: ● Ensure valid parameters specified. ● Move save area (CNTLBMSG) to beginning of $MAINT (X'C800'). ● Load library to library copy routine ($MALTL) following save area (X'C900') and go to **4** .	$MAINT
2 Use included source get routine (#MASYL) and load librarian find routine ($MALFN) in main storage. Determine device type (diskette or disk): ● If diskette, load record mode diskette data management (#DRDM) into main storage. ● If disk, record mode disk data management (#DDDM) is resident.	$MARTF
Open output file.	#DMOP
Return..	$MARTF
Find directory entry of library member to be copied.	$MALFN
Move type (S or P) and directory name into copy control statement.	$MARTF
Write copy control statement to file (disk or diskette).	#DDDM or #DRDM
Return.	$MARTF
Get records from library member.	@MASYL
Return.	$MARTF
Write record to file. Write CEND to file when at end of each member.	#DDDM or #DRDM
Continue processing until all requested members copied to file.	$MARTF
Display error messages as necessary.	#CLXS
Close output file.	#DMCL
Go to **5** to read next control statement.	$MARTF
3 Load library sector get/put routine ($MAPGS) and librarian find routine ($MALFN) into main storage. Determine device type (diskette or disk): ● If diskette, load diskette sector data management (#DRSM) into main storage. ● If disk, load disk sector data management (#DDSM) into main storage. Set DTF attributes for required function.	$MATFS
Open output file.	#DMOP
Return.	$MATFS
Find directory entry of library member to be copied.	$MALFN
Move directory entry and PTF data (if any) into data management buffer.	$MATFS
Get library member sectors and place in data management buffer.	$MAPGS
Continue processing until buffer is full.	$MATFS

Diagram 21.5 (Part 2 of 3). Perform Library Maintenance — Copy FROM — Library Functions ($MAINT)

DESCRIPTION	MODULE/ ROUTINE
Write buffer to file (disk or diskette).	#DDDM or #DRDM
Continue processing until all requested members copied to file.	$MATFS
Display error messages as necessary.	#CLXS
Close output file.	#DMCL
Go to **5** to read next control statement.	$MATFS
4 Load sector get/put ($MAPGS), librarian find ($MALFN), librarian open/close ($MALCO), and library directory fast insert ($MAFIR) into main storage.	$MALTL
Set up buffer to receive library directory and member.	$MALTL
Find directory entry of library member to copy.	$MALFN
Return.	$MALTL
Open member being copied (FROM library).	$MAPGS
Issue halt if copying SSP load member to system library and system not dedicated, or copying SSP load member to a user library.	#CLXS
If name not ALL, move partial or full new name into library control block (LCB).	$MALTL
If first member in a stack, enque library directory and read LCS.	
Move PTF data from FROM LCB to TO LCB.	Disk IOS
Open library member being created (TO library).	$MAPGS
Retrieve library member sectors from FROM library and place in buffer.	
Copy library member sectors from buffer to TO library.	
Return.	$MALTL
Close library member.	$MALCO
When stack is full or all members are processed:	$MALTL
● Call fast insert to update directory.	$MAFIR
● Return to function mainline.	$MALTL
● Write LCS.	Disk IOS
● Dequeue library directory.	$MALTL
Continue processing until all requested library members copied to TO library.	
Display error messages as necessary.	#CLXS
Go to **5** to read next control statement.	$MALTL
5 Load $MAINT back into user area to location X'C800'.	
Read next control statement.	$MAINT
When END read, call control storage end-of-job ($EOJ).	

Diagram 21.5 (Part 3 of 3). Perform Library Maintenance — Copy FROM — Library Functions ($MAINT)



From: Initiator

INPUT

XR1

CNTLBMSG

Main Storage

Transient Area

UserArea:
$MAINT –
$MATLS
$MARFF –
$MAF32
$MARDR

Library common area

Control statements

Disk or diskette file

Operator keyed data

Library control sector

PROCESS

1 Interpret control statements and build library common area

2 Copy sector mode file to library

3 Copy record mode file to library

4 Copy System/32 sector mode file to library

5 Copy from reader to library

6 Read next control statement

OUTPUT

Library control sector

Library directory

Library members

To: Control Storage End-of-Job Transient ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Read and syntax check utility control statements.	#USYX
If END statement, call end of job ($EOJ). Process TO parameter:	$MAINT
● Ensure library exists.	#MAFLB
● Save library format 1 address. Encode NAME and OMIT parameters (if any). If RECL parameter given, set record mode switch and if not given, set record length to 8 (120 if reader). Determine if device is disk or diskette. Ensure RECL valid (if applicable).	$MAINT

Diagram 21.6 (Part 1 of 4). Perform Library Maintenance – Copy TO – Library Functions ($MAINT)

DESCRIPTION	MODULE/ ROUTINE
If copy FROM file, allocate file.	#CAML
Issue error messages if necessary.	#CLSG
Move save area (CNTLBMSG) to beginning of $MAINT (X'C800').	$MAINT
Load appropriate COPY routine into main storage following save area (X'C900'): ● If copy sector mode file to library, load $MATLS and to to 2. ● If copy record mode file to library, load $MARFF and go to 3. ● If copy System/32 sector mode file to library, load $MAF32 and go to 4. ● If copy from reader to library, load $MARDR and go to 5.	
2 Load library sector get/put ($MAPGS), librarian open/close ($MALCO), and librarian directory fast insert ($MAFIR) routines into main storage.	$MATLS
Determine device type (disk or diskette): ● If disk, load disk sector mode data management (#DDSM) into main storage. ● If diskette, load diskette sector mode data management (#DRSM) into main storage.	
Set DTF attributes for requested function.	
Open input file.	#DMOP
Read control records, PTF information, and data records from file and place in buffer (use disk or diskette data management as appropriate).	#DDSM or #DRSM
If an SSP object (O) module is being copied to system library, ensure system is dedicated.	$MATLS
If an SSP load module is copied to a user library, issue warning.	$MATLS
Enqueue the library.	
Read the library control sector (LCS) into main storage.	FDIOS
Open library and member being copied.	$MAPGS
Copy member sectors from buffer to library.	
Continue processing until member copied to library.	$MATLS
Close library member.	$MALCO
When stack is full or all members processed:	$MATLS
● Call fast insert to update directory.	$MAFIR
● Return to routine mainline.	$MATLS
● Write LCS.	FDIOS
● Dequeue library directory.	$MATLS
Continue processing members until entire file copied to library or error encountered.	
Issue error messages as necessary.	#CLXS
If SSP O members copied to system library, run cross reference resolver to update where-to-go (WTG) table and format index tables.	#MAXRF
Update system transient table.	$MATLS
Close file.	#DMCL

Diagram 21.6 (Part 2 of 4). Perform Library Maintenance — Copy TO — Library Functions ($MAINT)

DESCRIPTION	MODULE/ ROUTINE
Go to 6 to read next control statement.	$MATLS
3 Load library record put ($MAPUR), librarian open/close ($MALCO), and librarian directory insert ($MALIL) routines into main storage. Determine device type (disk or diskette). ● If diskette, load diskette record mode data management (#DRDM) into main storage. ● If disk, disk data management (#DDDM) resident in main storage. Set DTF attributes for requested function.	$MARFF
Open input file.	#DMOP
Get COPY record from file and place in buffer (use disk or diskette data management as appropriate).	#DDDM or #DRDM
Check copy control record syntax.	$MARFF
Open library for member in file.	$MAPUR
Get record from file member and place in buffer (use disk or diskette data management as appropriate).	#DDDM or #DRDM
Copy member record from buffer to library.	$MAPUR
Continue processing member record until CEND record is read or error occurs.	$MARFF
Close library member.	$MAPUR
Continue processing until all file members copied to library, end-of-file reached, or error condition encountered.	$MARFF
Issue error messages as necessary.	#CLXS
Close input file.	#DMCL
Go to 6 to read next control statement.	$MARFF
4 Load library sector get/put ($MAPGS), librarian open/close ($MALOC), and librarian directory insert ($MALIL) routines into main storage.	$MAF32
Determine device type (disk or diskette): ● If disk, load disk sector mode data management (#DDSM) into main storage. ● If diskette, load diskette sector data management (#DRSM) into main storage. Set DTF attributes for requested function.	$MAF32
Open input file.	#DMOP
Return.	$MAF32
Read control records, PTF information, and data from file and place in buffer (use disk or diskette data management as appropriate).	#DDSM or #DRSM
Convert System/32 directory entries to System/34 directory entries and put in library control block (LCB). Put System/32 PTFs in PTFLOG.	$MAF32
If SCP (System/32) member, do not copy to System/34 library (issue halt).	#CLXS
Open library and member being copied.	$MAPGS

Diagram 21.6 (Part 3 of 4). Perform Library Maintenance – Copy TO – Library Functions ($MAINT)

DESCRIPTION	MODULE/ ROUTINE
Copy member sectors from buffer to library.	$MAPGS
If source or proc member, convert compression format.	$MAF32
Continue processing until file member copied to library.	
Update directory and close library member.	$MALCO
Continue processing file members until entire file copied to library or error encountered.	$MAF32
Issue error messages as necessary.	#CLXS
Close file.	#DMCL
Go to 6 to read next control statement.	$MAF32
5 Load SYSIN (#CLSS) at location X'DD00'.	$MARDR
Load library record put ($MAPUR) and librarian open/close ($MALCO) before SYSIN in main storage.	
Calculate SYSIN record buffer address (buffer also used by record put and insert transient).	
Ensure valid name parameter specified.	
Open library for new member.	$MAPUR
Return.	$MARDR
Get record (data from keyboard or procedure) using SYSIN.	#CLSS
Check for CEND record:	$MARDR
● If CEND not read, put record to library and continue processing records. ● If CEND read, close library.	$MAPUR
If no source records entered before CEND, issue halt.	#CLXS
Go to 6 to read next control statement.	$MARDR
6 Load $MAINT back into user area at location X'C800'.	
Read next control statement	$MAINT
When END read, all control storage end of job ($EOJ).	
Note: $EOJ can be called by $MATLS, $MAF32, $MARFF, and $MARDR when SYSLOG returns 3 option halt.	

Diagram 21.6 (Part 4 of 4). Perform Library Maintenance — Copy TO — Library Functions ($MAINT)

Program Organization

Figures 21-1 through 21-6 show the control flow of the library maintenance utility.

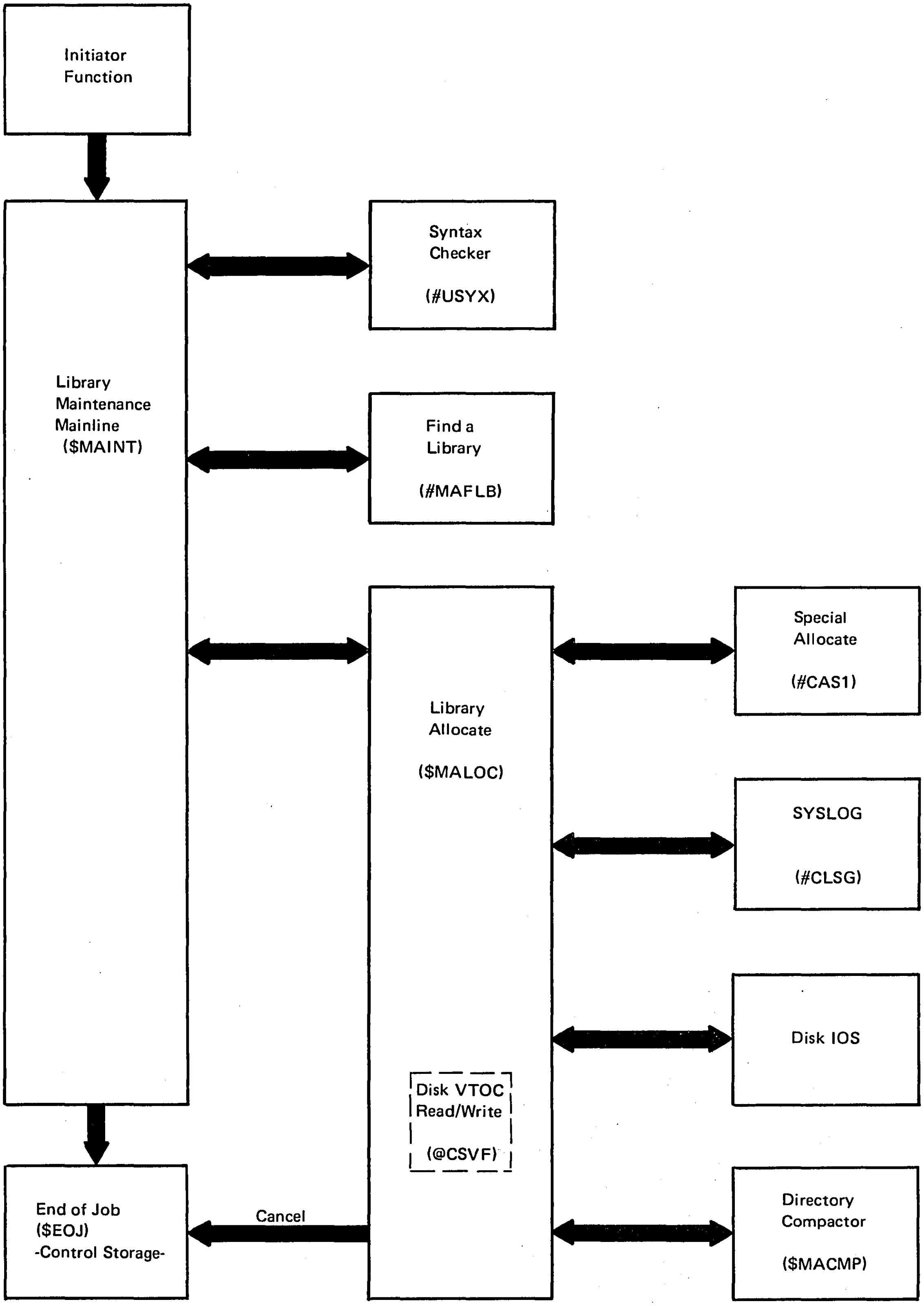


Figure 21-1. Library Maintenance — Allocate Function Control Flow ($MAINT)

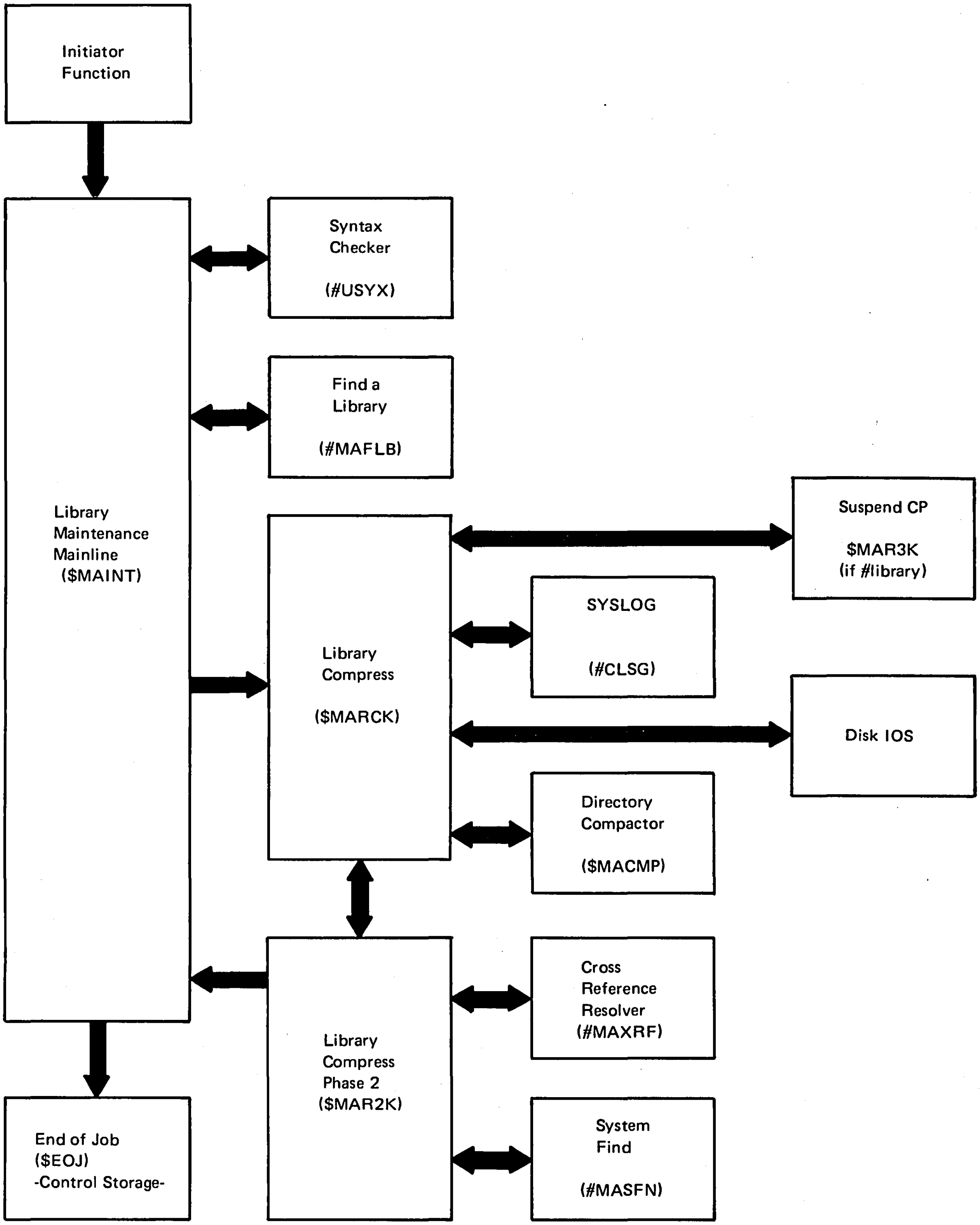

Initiator Function
Library Maintenance Mainline ($MAINT)
Syntax Checker (#USYX)
Find a Library (#MAFLB)
Library Compress ($MARCK)
Suspend CP $MAR3K (if #library)
SYSLOG (#CLSG)
Disk IOS
Directory Compactor ($MACMP)
Library Compress Phase 2 ($MAR2K)
Cross Reference Resolver (#MAXRF)
System Find (#MASFN)
End of Job ($EOJ) -Control Storage-

Figure 21-2. Library Maintenance — Compress Function Control Flow ($MAINT)

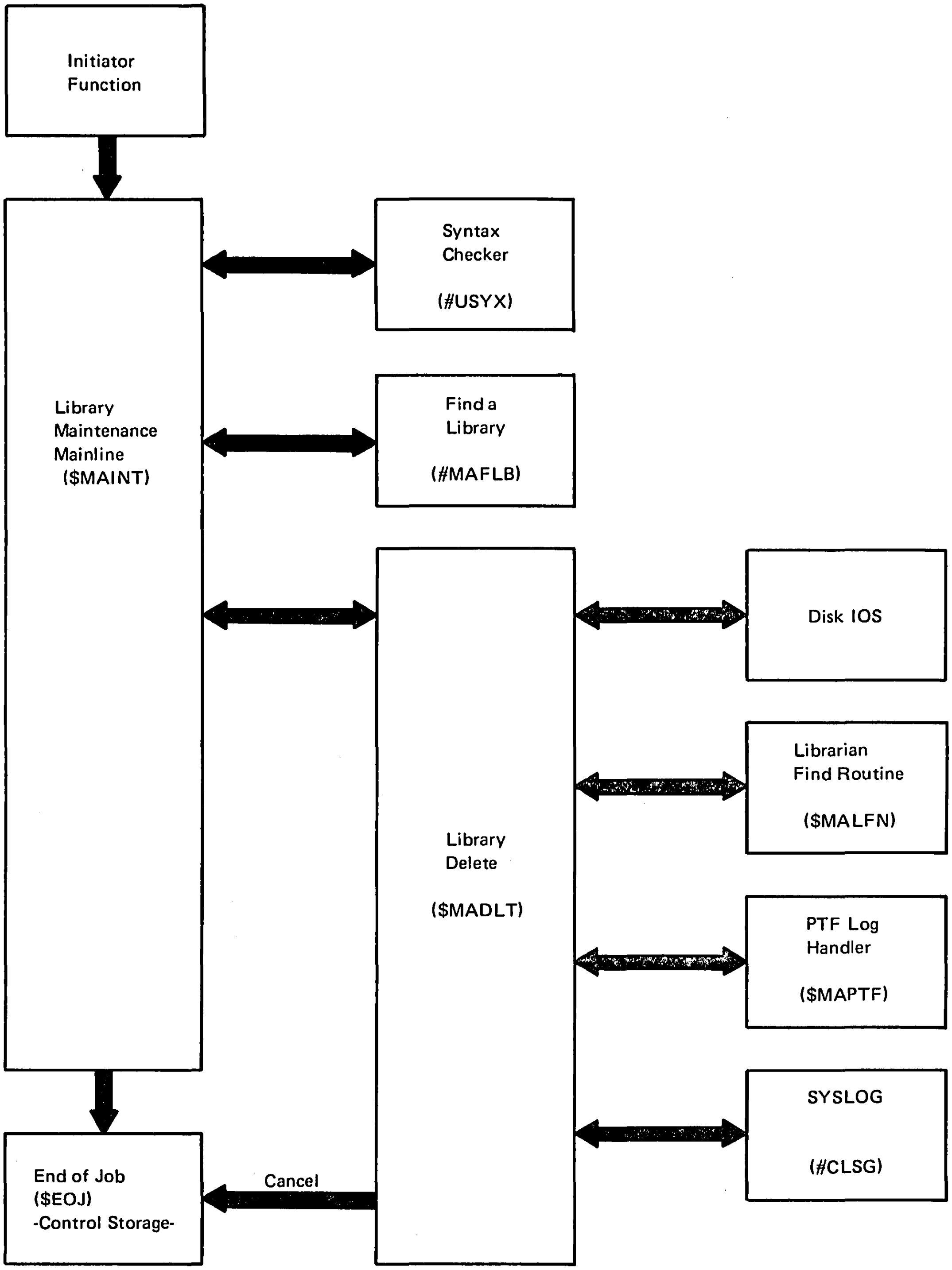


Figure 21-3. Library Maintenance — Delete Function Control Flow ($MAINT)

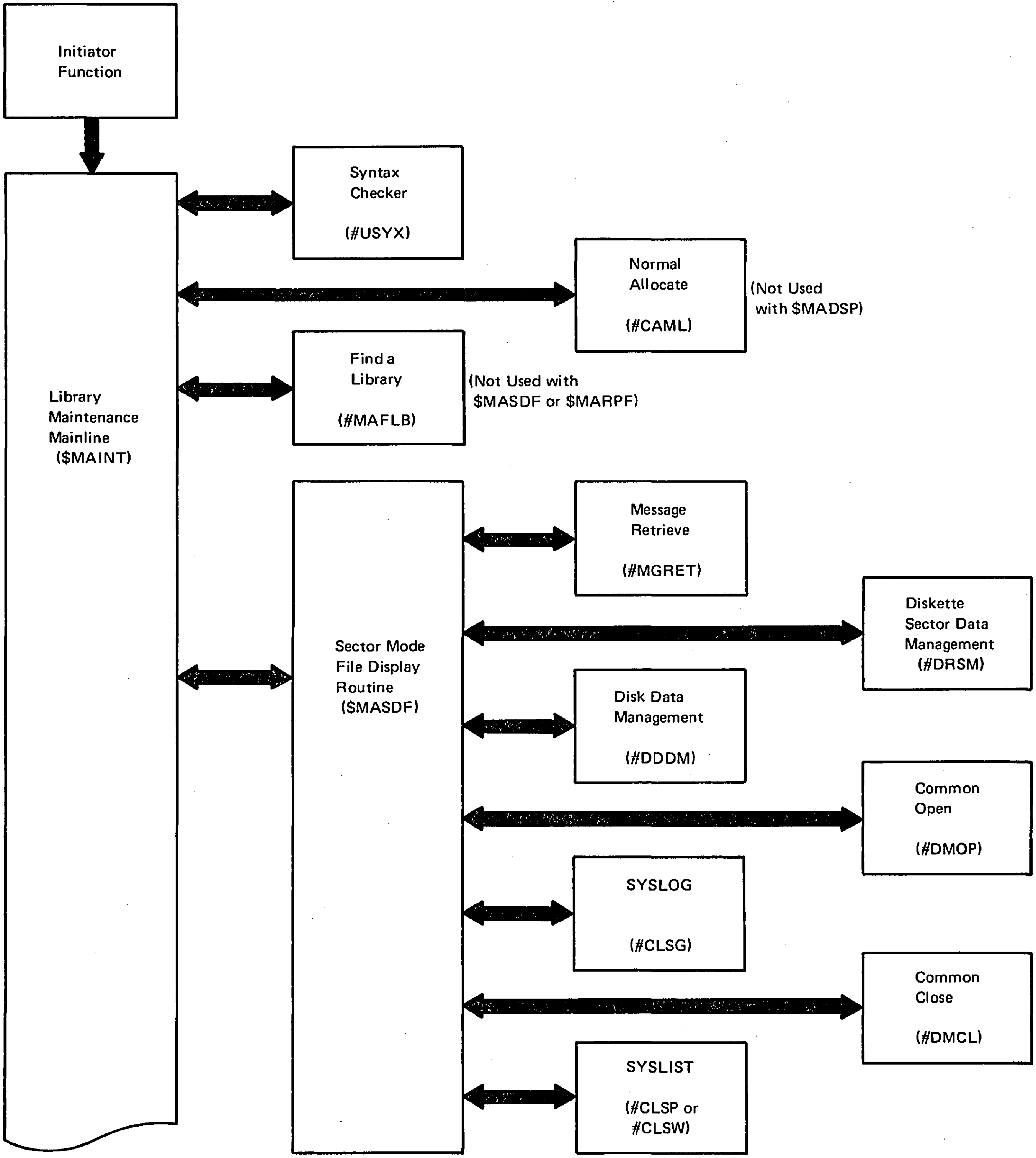
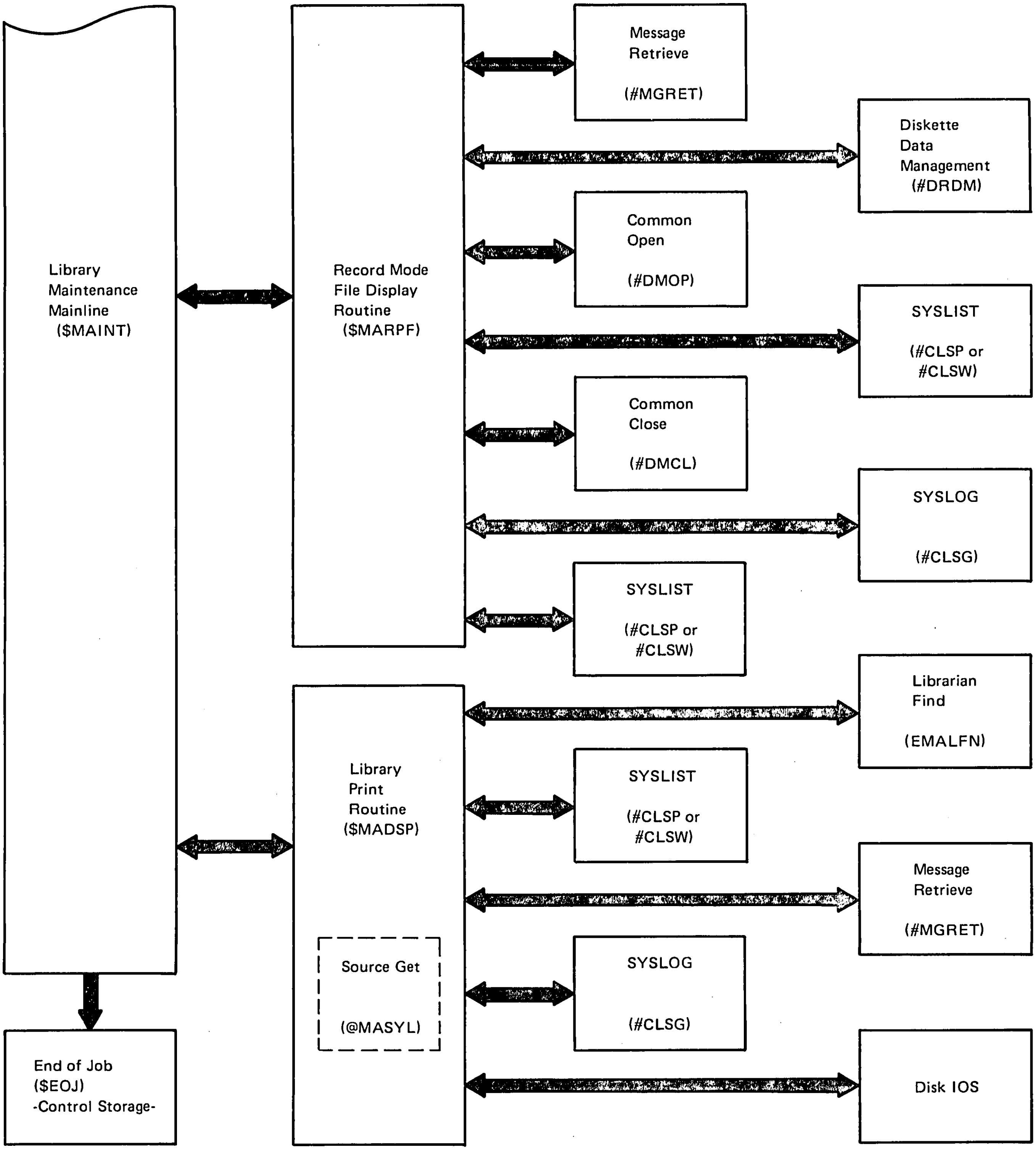

Initiator Function
Library Maintenance Mainline ($MAINT)
Syntax Checker (#USYX)
Normal Allocate (#CAML)
(Not Used with $MADSP)
Find a Library (#MAFLB)
(Not Used with $MASDF or $MARPF)
Sector Mode File Display Routine ($MASDF)
Message Retrieve (#MGRET)
Diskette Sector Data Management (#DRSM)
Disk Data Management (#DDDM)
Common Open (#DMOP)
SYSLOG (#CLSG)
Common Close (#DMCL)
SYSLIST (#CLSP or #CLSW)

Figure 21-4 (Part 1 of 2). Library Maintenance – Display Function Control Flow ($MAINT)


Library Maintenance Mainline ($MAINT)
Record Mode File Display Routine ($MARPF)
Message Retrieve (#MGRET)
Diskette Data Management (#DRDM)
Common Open (#DMOP)
SYSLIST (#CLSP or #CLSW)
Common Close (#DMCL)
SYSLOG (#CLSG)
SYSLIST (#CLSP or #CLSW)
Librarian Find (EMALFN)
Library Print Routine ($MADSP)
SYSLIST (#CLSP or #CLSW)
Message Retrieve (#MGRET)
Source Get (@MASYL)
SYSLOG (#CLSG)
Disk IOS
End of Job ($EOJ) -Control Storage-

Note: $MADSP, $MASDF, and $MARPF can call $EOJ with SYSLOG 3 option halt.

Figure 21-4 (Part 2 of 2). Library Maintenance — Display Function Control Flow ($MAINT)

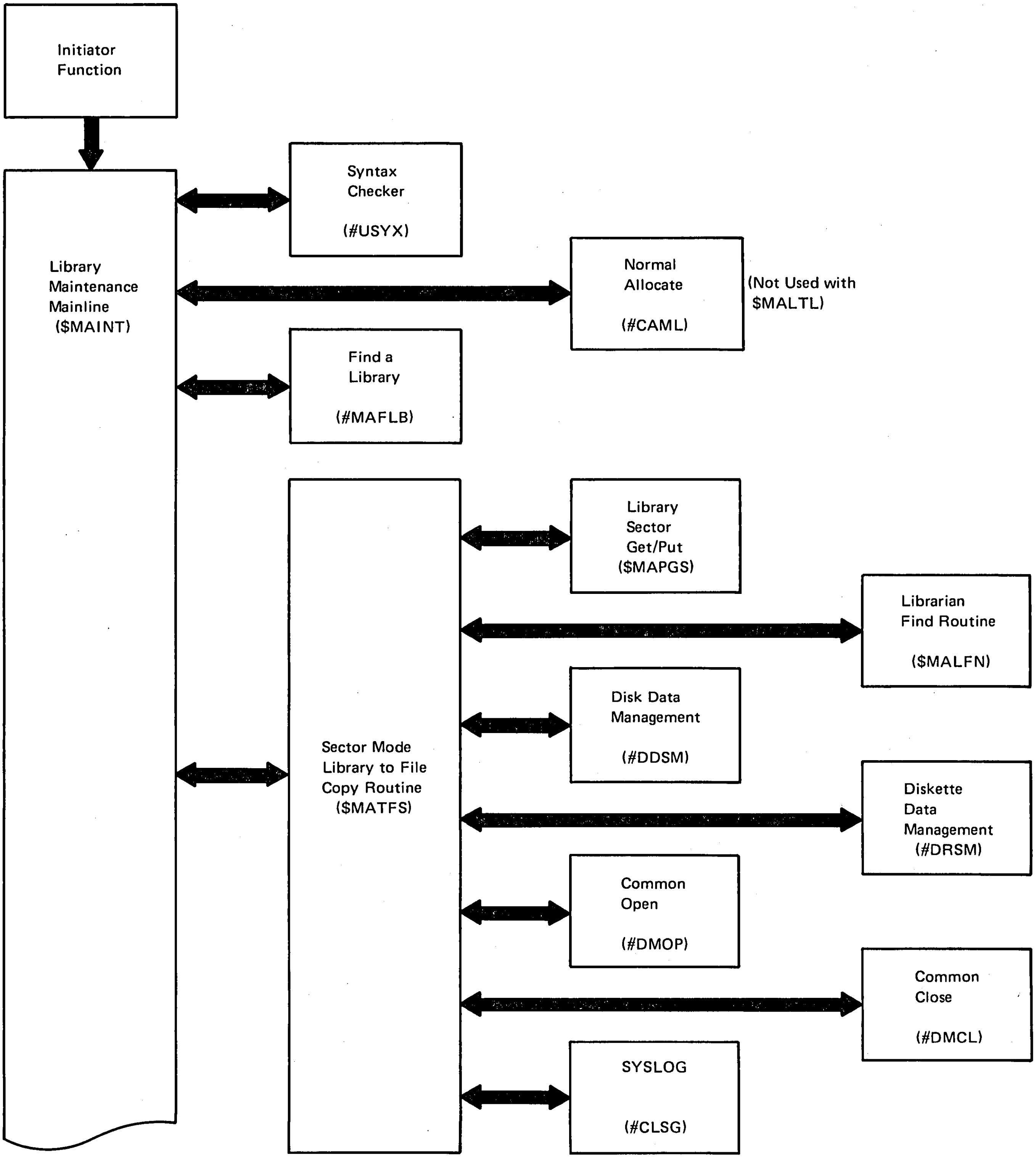
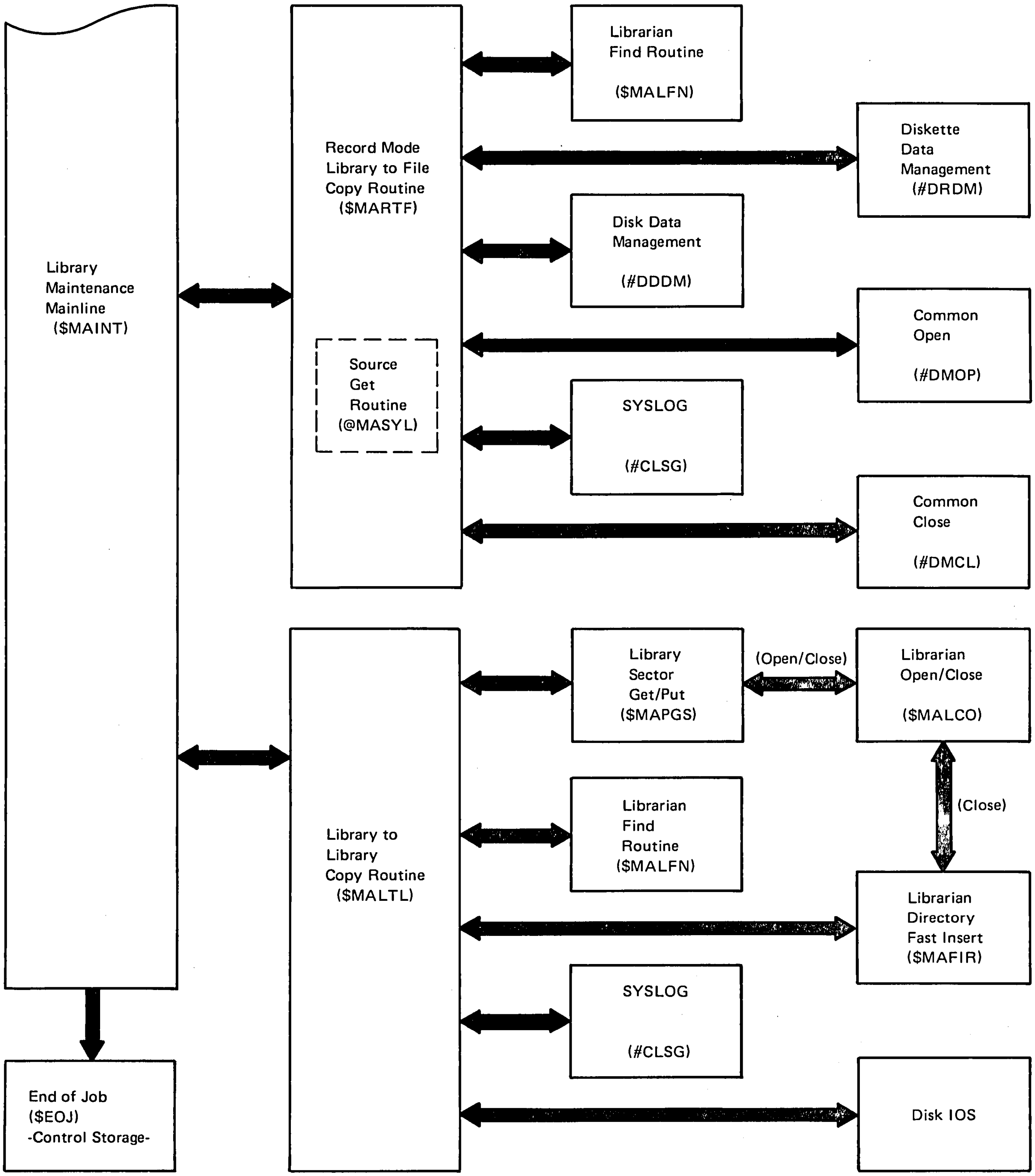

Initiator Function
Library Maintenance Mainline ($MAINT)
Syntax Checker (#USYX)
Normal Allocate (#CAML)
(Not Used with $MALTL)
Find a Library (#MAFLB)
Sector Mode Library to File Copy Routine ($MATFS)
Library Sector Get/Put ($MAPGS)
Librarian Find Routine ($MALFN)
Disk Data Management (#DDSM)
Diskette Data Management (#DRSM)
Common Open (#DMOP)
Common Close (#DMCL)
SYSLOG (#CLSG)

Figure 21-5 (Part 1 of 2). Library Maintenance – Copy FROM-Library Functions Control Flow ($MAINT)


Library Maintenance Mainline ($MAINT)
Record Mode Library to File Copy Routine ($MARTF)
Source Get Routine (@MASYL)
Librarian Find Routine ($MALFN)
Diskette Data Management (#DRDM)
Disk Data Management (#DDDM)
Common Open (#DMOP)
SYSLOG (#CLSG)
Common Close (#DMCL)
Library to Library Copy Routine ($MALTL)
Library Sector Get/Put ($MAPGS)
(Open/Close)
Librarian Open/Close ($MALCO)
(Close)
Librarian Find Routine ($MALFN)
Librarian Directory Fast Insert ($MAFIR)
SYSLOG (#CLSG)
Disk IOS
End of Job ($EOJ) -Control Storage-

Note: $MATFS, $MARTF, and $MALTL can call $EOJ with SYSLOG 3 option halt.

Figure 21-5 (Part 2 of 2). Library Maintenance — Copy FROM-Library Function Control Flow ($MAINT)

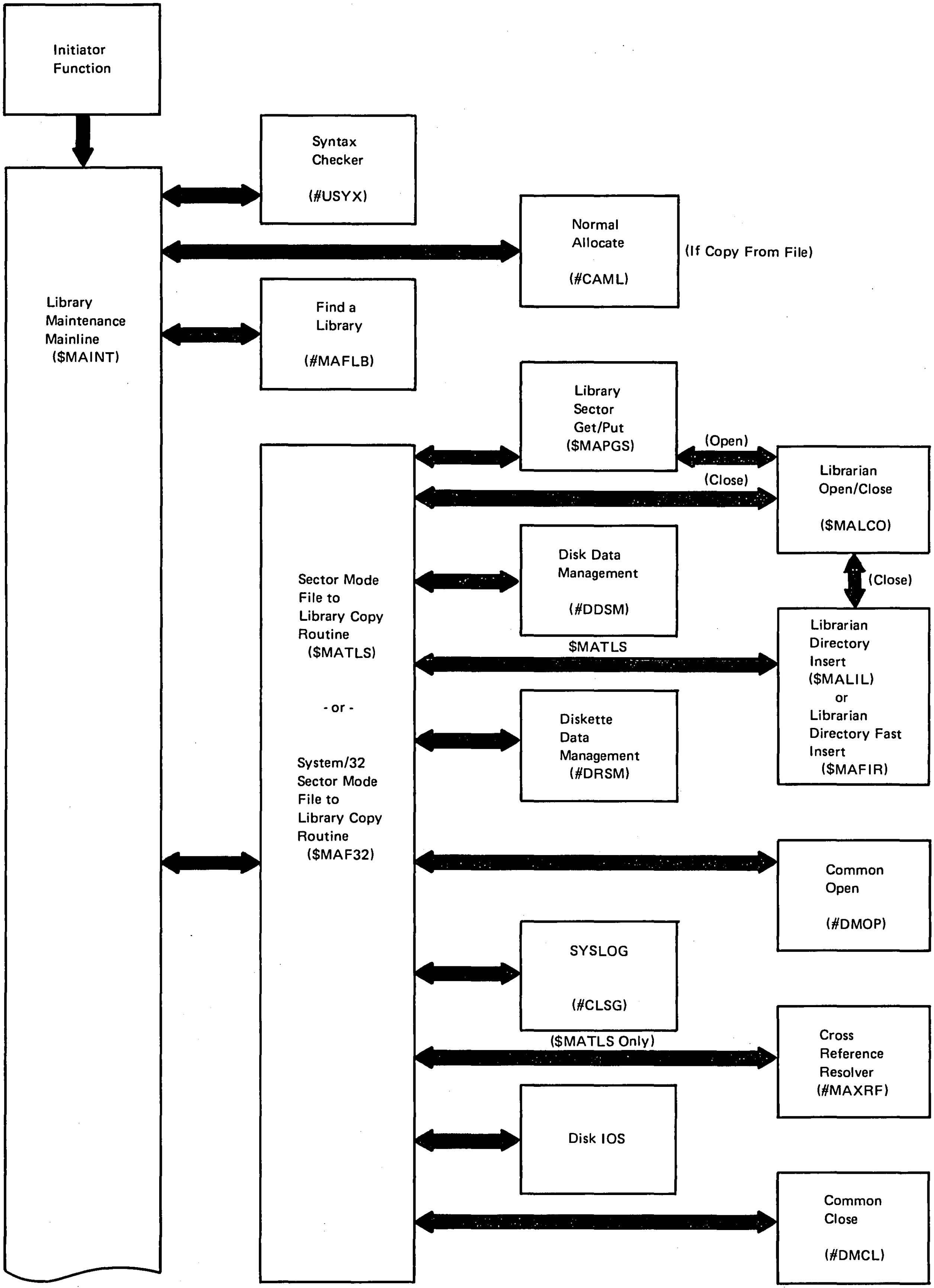
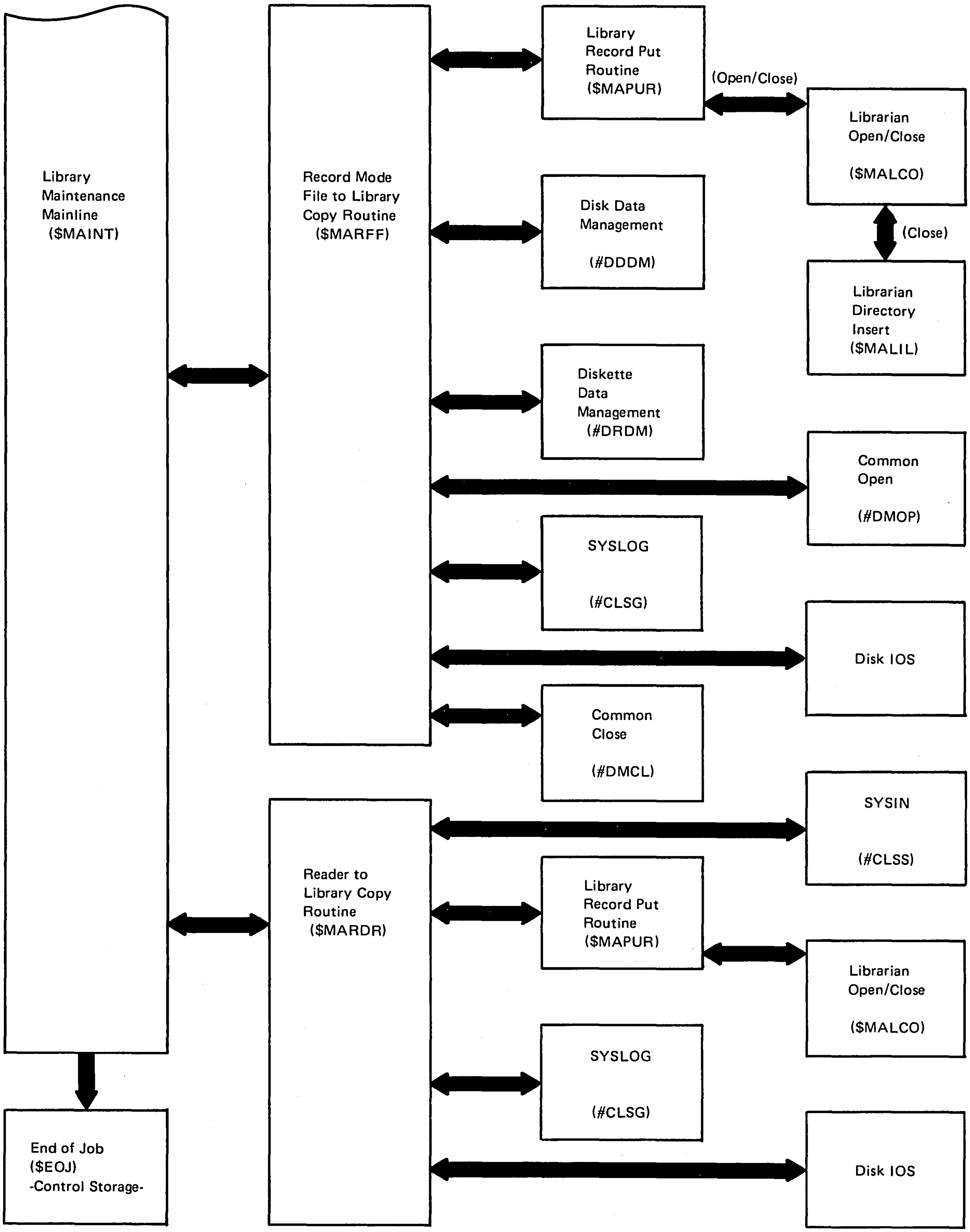

Initiator Function
Library Maintenance Mainline ($MAINT)
Syntax Checker (#USYX)
Normal Allocate (#CAML)
(If Copy From File)
Find a Library (#MAFLB)
Library Sector Get/Put ($MAPGS)
(Open)
(Close)
Librarian Open/Close ($MALCO)
(Close)
Sector Mode File to Library Copy Routine ($MATLS)
- or -
System/32 Sector Mode File to Library Copy Routine ($MAF32)
Disk Data Management (#DDSM)
$MATLS
Librarian Directory Insert ($MALIL) or Librarian Directory Fast Insert ($MAFIR)
Diskette Data Management (#DRSM)
Common Open (#DMOP)
SYSLOG (#CLSG)
($MATLS Only)
Cross Reference Resolver (#MAXRF)
Disk IOS
Common Close (#DMCL)

Figure 21-6 (Part 1 of 2). Library Maintenance – Copy TO-Library Function Control Flow ($MAINT)



Note: $MATLS, $MAF32, $MARFF, and $MARDR can call $EOJ with SYSLOG 3 option Halt.

Figure 21-6 (Part 2 of 2). Library Maintenance – Copy TO-Library Function Control Flow ($MAINT)

Data Areas

LIBRARY COMMON AREA – CNTLBMSG

The library common area, CNTLBMSG, is a 256-byte area that begins at location X'C800'. It is created by the library mainline routine ($MAINT) and used as a communication area for the library maintenance routines. Figure 21-7 shows its format and contents.

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes in Decimal	Description	Routines that Change Data
0	LBFROMNM	8	Library name	$MAINT
8	LBFROM	1	From parameter	$MAINT
			X'20' = F1 (LIBRARY)	
			X'21' = Disk (FILE)	
			X'22' = Reader	
9	LBFROMLB	2	F1 address if first byte is X'20'	$MAINT
B	LBLIB	1	Member type O, R, S, P	$MAINT
			X'23' = System	
			X'27' = All	
C	LBNAME11	11	Member name[1]	$MAINT
17	LBTONM	8	To parameter or X'25' = Print	$MAINT
1F	LBTO	1		
20	LBTLB	2		
22	LBNEW	8	New name	$MAINT
2A	LBOMIT11	11	Names to omit[1]	$MAINT

[1] If no name is present, the first (leftmost) byte of this field contains one of the following flags:

X'27' = All
X'23' = System
X'24' = Directory

If a partial name is present, the eighth byte of this field contains the length of the partial name.

Figure 21-7 (Part 1 of 2). CNTLBMSG Library Common Area ($MAINT)

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes in Decimal	Description	Routines that Change Data

35	LBRETAIN1	1	Retain flag
			P=Permanent
			R=Replace
			S=SSP (can be deleted)
				$MAINT

36	LBRECL	1	Record length X'20' or X'28' through X'78'	$MAINT
37	LBFILE	8	File name	$MAINT
3F	LBSIZE	2	Number of blocks – total	$MAINT
41	LBINCR	2	Number of blocks – increase	$MAINT
43	LBDECR	2	Number of blocks – decrease	$MAINT
45	LBDIRSIZ	2	Number of sectors for directory size	$MAINT
47	LBPTF	5	Decimal PTF ID number	$MAINT
4C	LBADD	1	Add to existing file	$MAINT
4D	LBDEV	1	Device code	$MAINT
4E	LBCREATE	1	Create new	$MAINT
4F	LBLOC	1	Spindle preference	$MAINT
50	LBBASIC	1	Basic exchange format for new diskette	$MAINT
51	LBMRT	1	MRT procedure	$MAINT
52	LBHIST	1	Log OCL from procedure	$MAINT
59	LBBLOCK	7	Block number for location by block	$MAINT
5A	LBPDATA	1	Data for procedure	–
5B	–	43	Reserved	–
86	LBDMDTF	128	DTF passed to drivers using data management	$MAINT

Figure 21-7 (Part 2 of 2). CNTLBMSG Library Common Area ($MAINT)

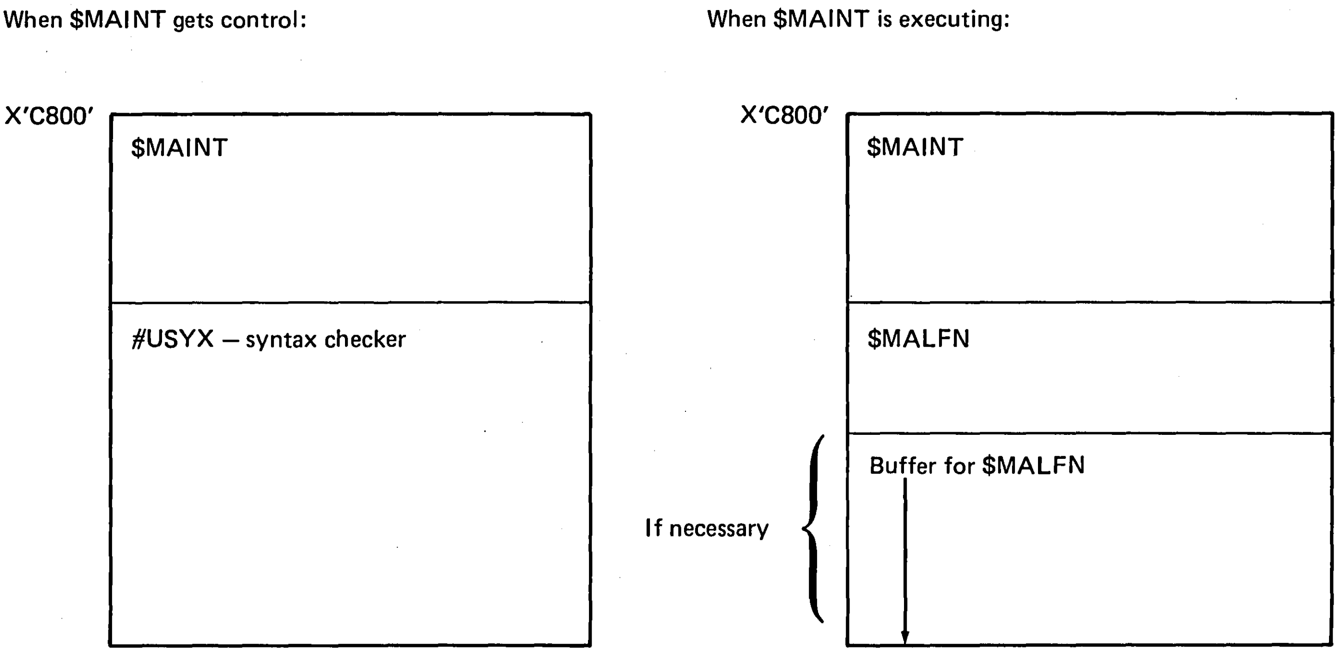


Figure 21-8 (Part 1 of 15). Library Maintenance Utility Storage Maps

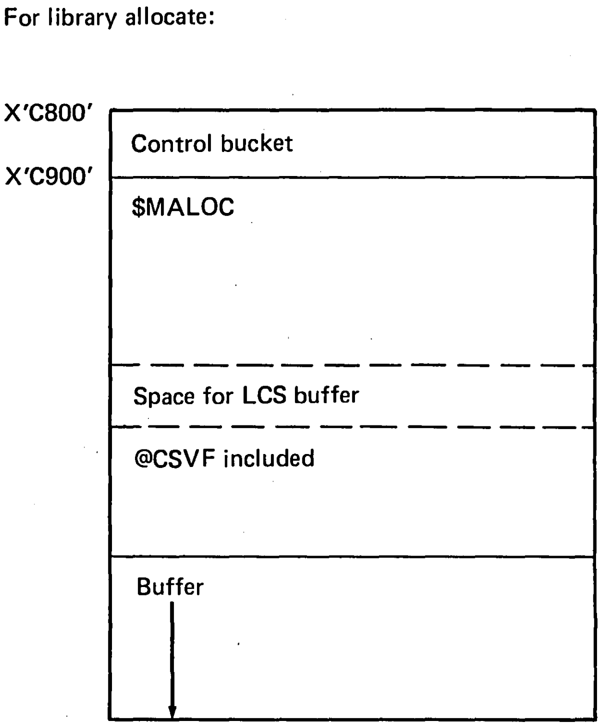
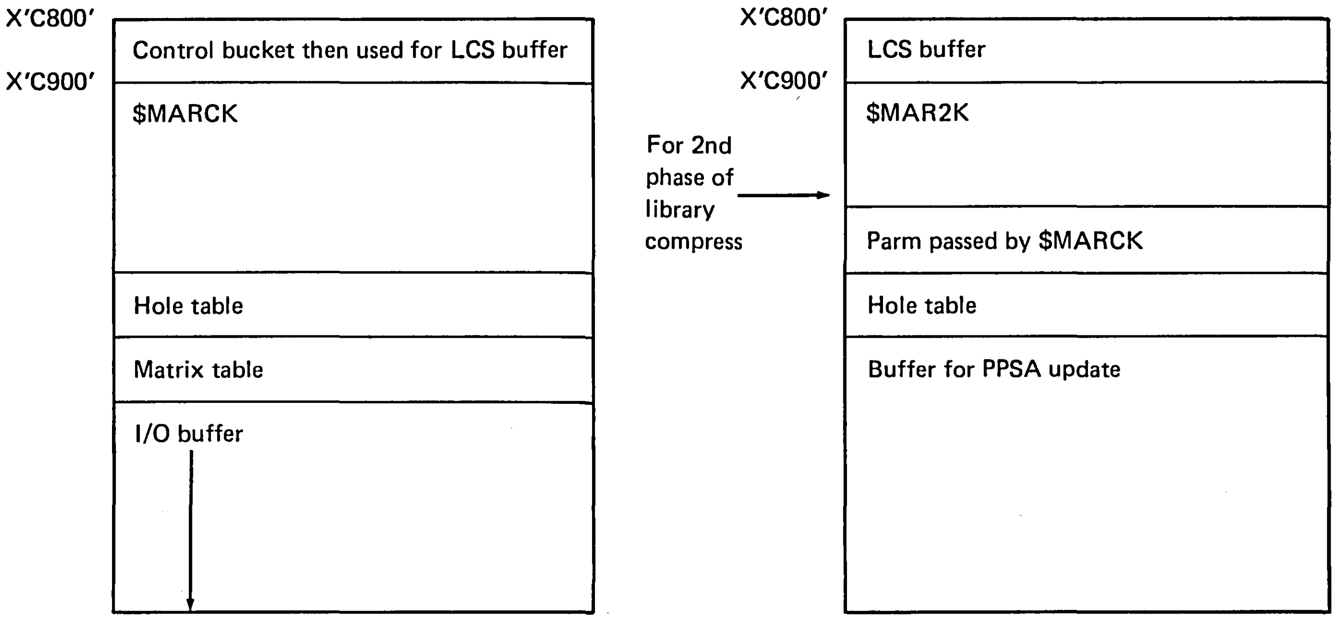


Figure 21-8 (Part 2 of 15). Library Maintenance Utility Storage Maps

For library compress:



For the system library additional processing is necessary. The storage maps for the additional processing are below. If another compress pass is needed, $MAR2K loads $MARCK rather than $MAINT.

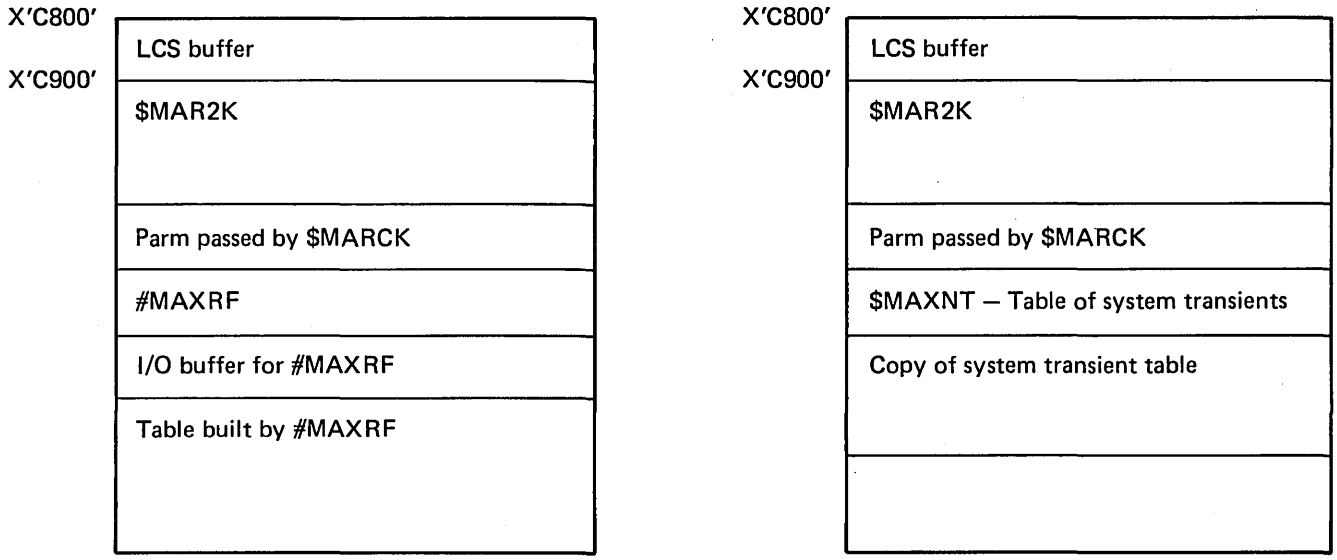


Figure 21-8 (Part 3 of 15). Library Maintenance Utility Storage Maps

For library member delete:

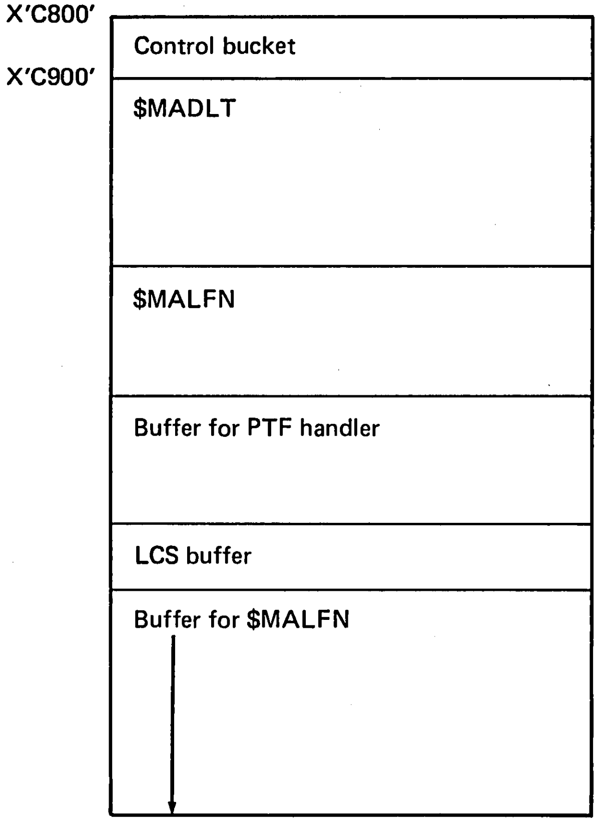


Figure 21-8 (Part 4 of 15). Library Maintenance Utility Storage Maps

For library to display copy:

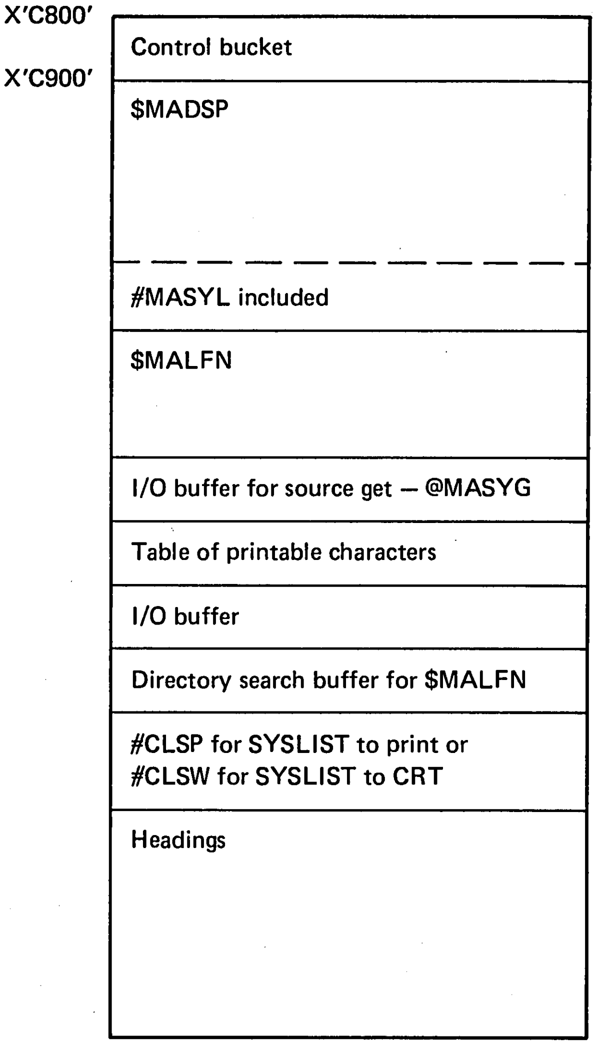

X'C800'
X'C900'
Control bucket
$MADSP
#MASYL included
$MALFN
I/O buffer for source get — @MASYG
Table of printable characters
I/O buffer
Directory search buffer for $MALFN
#CLSP for SYSLIST to print or
#CLSW for SYSLIST to CRT
Headings

Figure 21-8 (Part 5 of 15). Library Maintenance Utility Storage Maps

For librarian file to display copy:

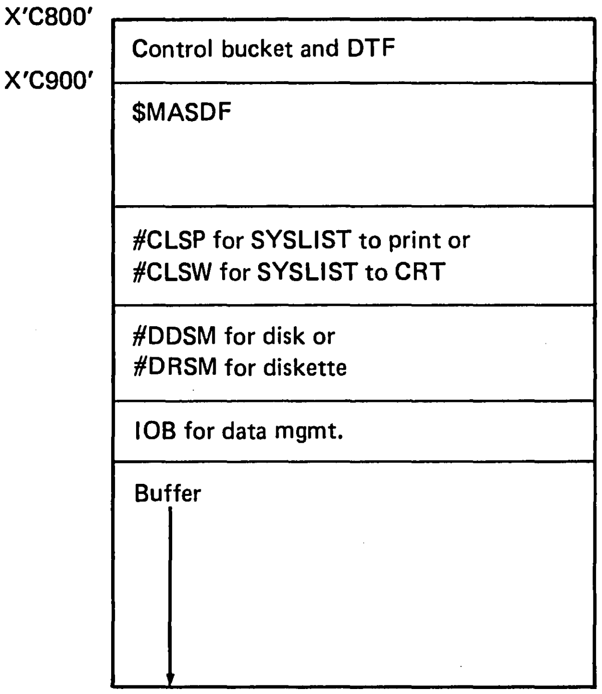


Figure 21-8 (Part 6 of 15). Library Maintenance Utility Storage Maps

For record mode librarian file to display copy:

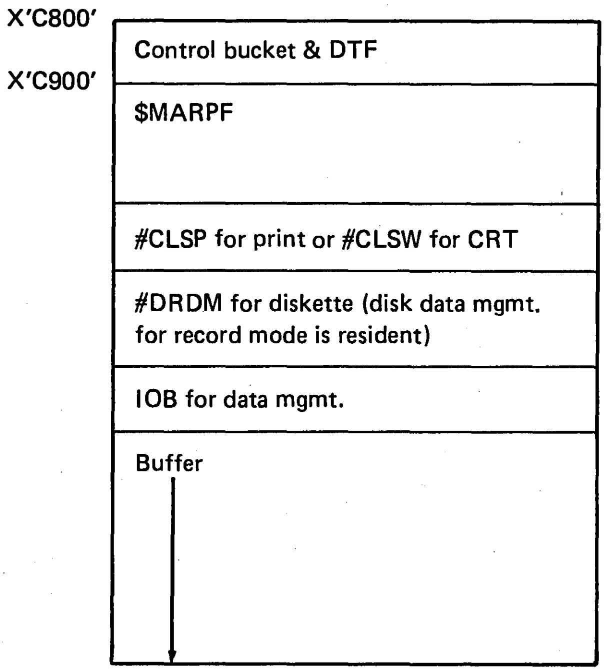

X'C800'
X'C900'
Control bucket & DTF
$MARPF
#CLSP for print or #CLSW for CRT
#DRDM for diskette (disk data mgmt.
for record mode is resident)
IOB for data mgmt.
Buffer

Figure 21-8 (Part 7 of 15). Library Maintenance Utility Storage Maps

For sector mode library to a file copy:

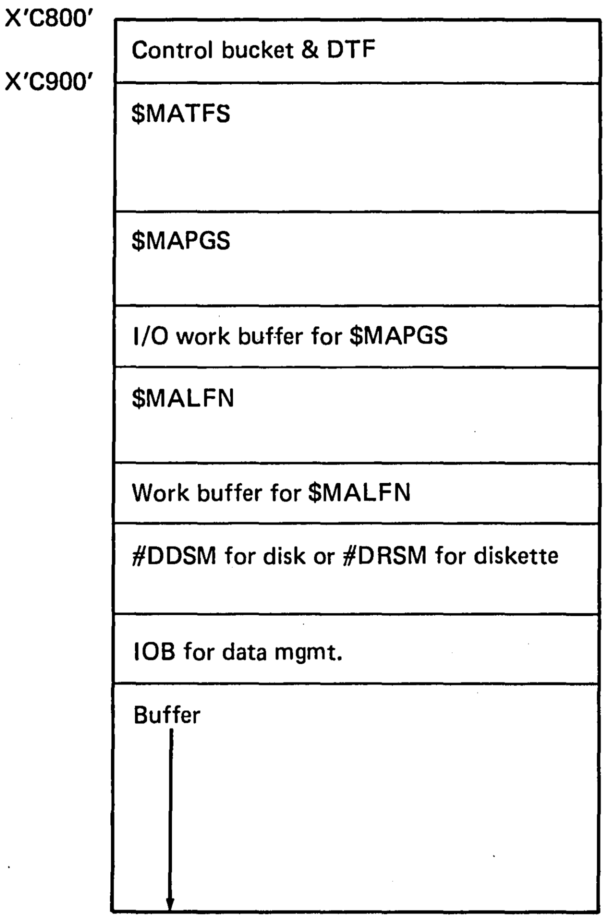


Figure 21-8 (Part 8 of 15). Library Maintenance Utility Storage Maps

For record mode library to a file copy:

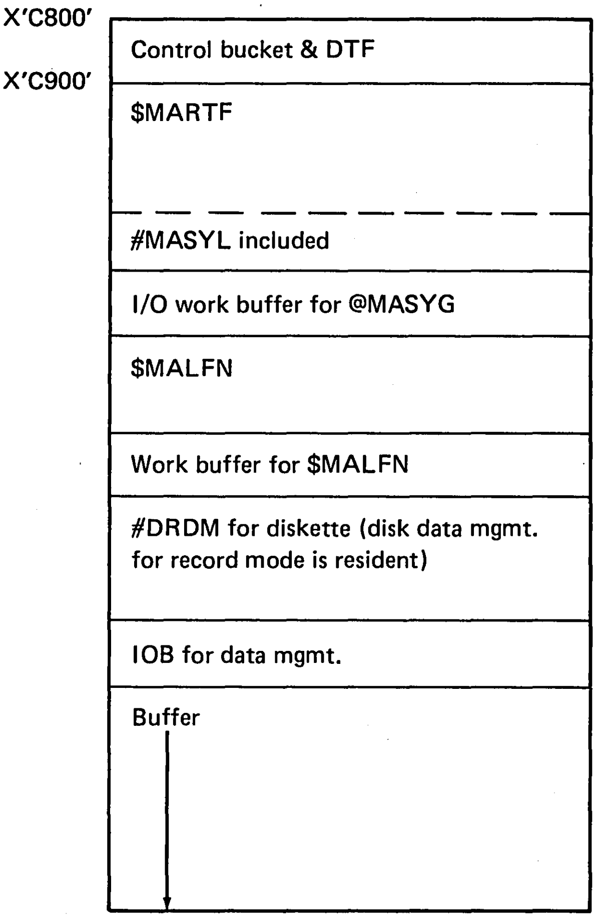


Figure 21-8 (Part 9 of 15). Library Maintenance Utility Storage Maps

For library to library copy:

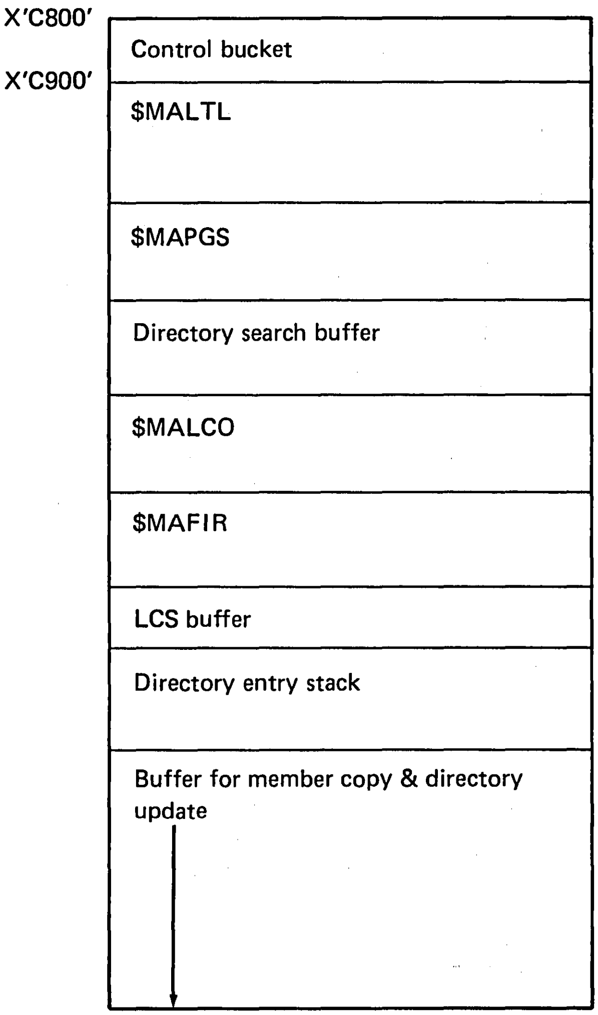


Figure 21-8 (Part 10 of 15). Library Maintenance Utility Storage Maps

For sector mode file to a library copy: When $MAFIR is called, the part of the region indicated is pushed to disk. The map on the right is the result. When $MAFIR is finished, the pushed area is restored.

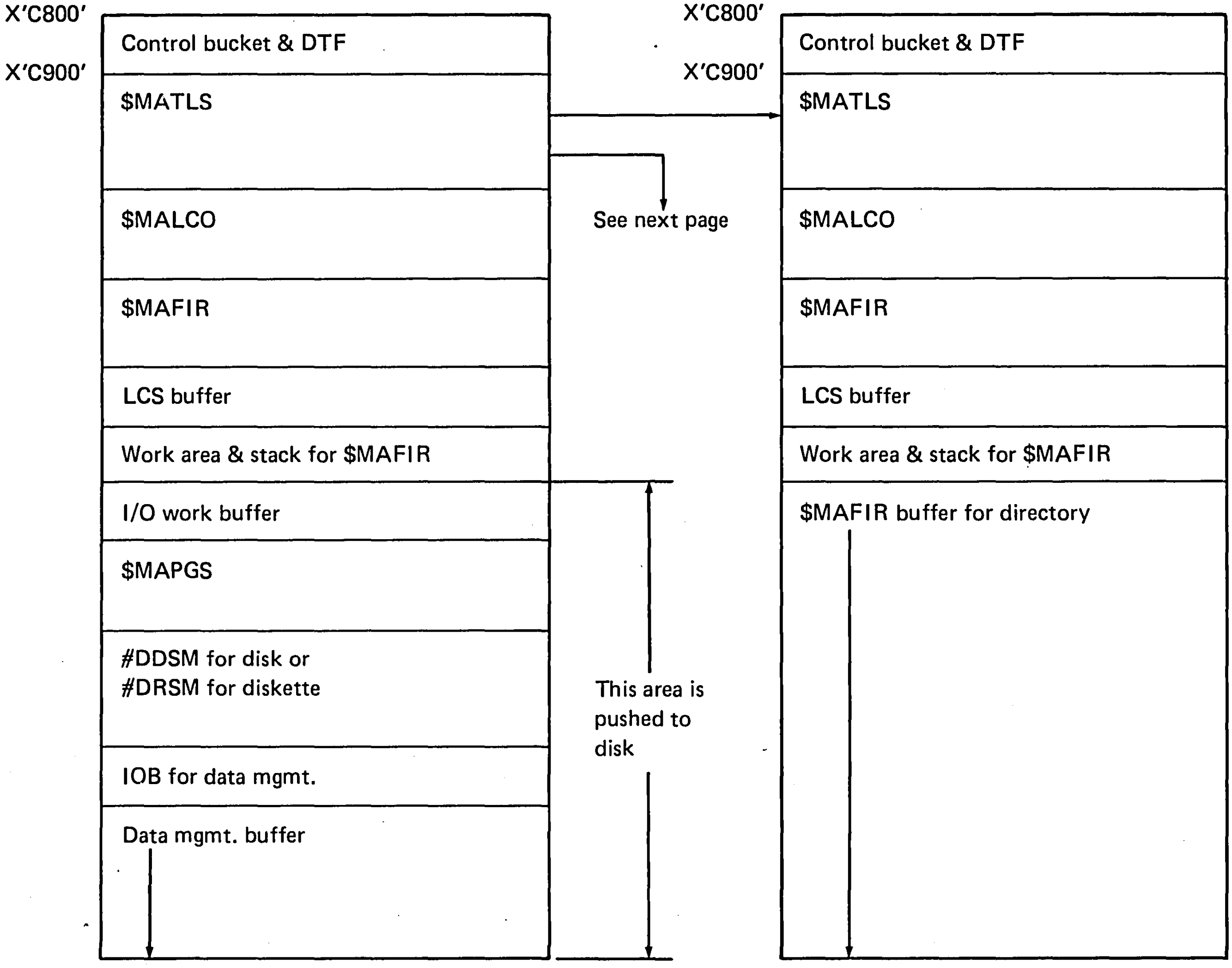


Figure 21-8 (Part 11 of 15). Library Maintenance Utility Storage Maps

For SSP load members copied to the system library, additional processing is necessary. The storage maps for the additional processing are given below.

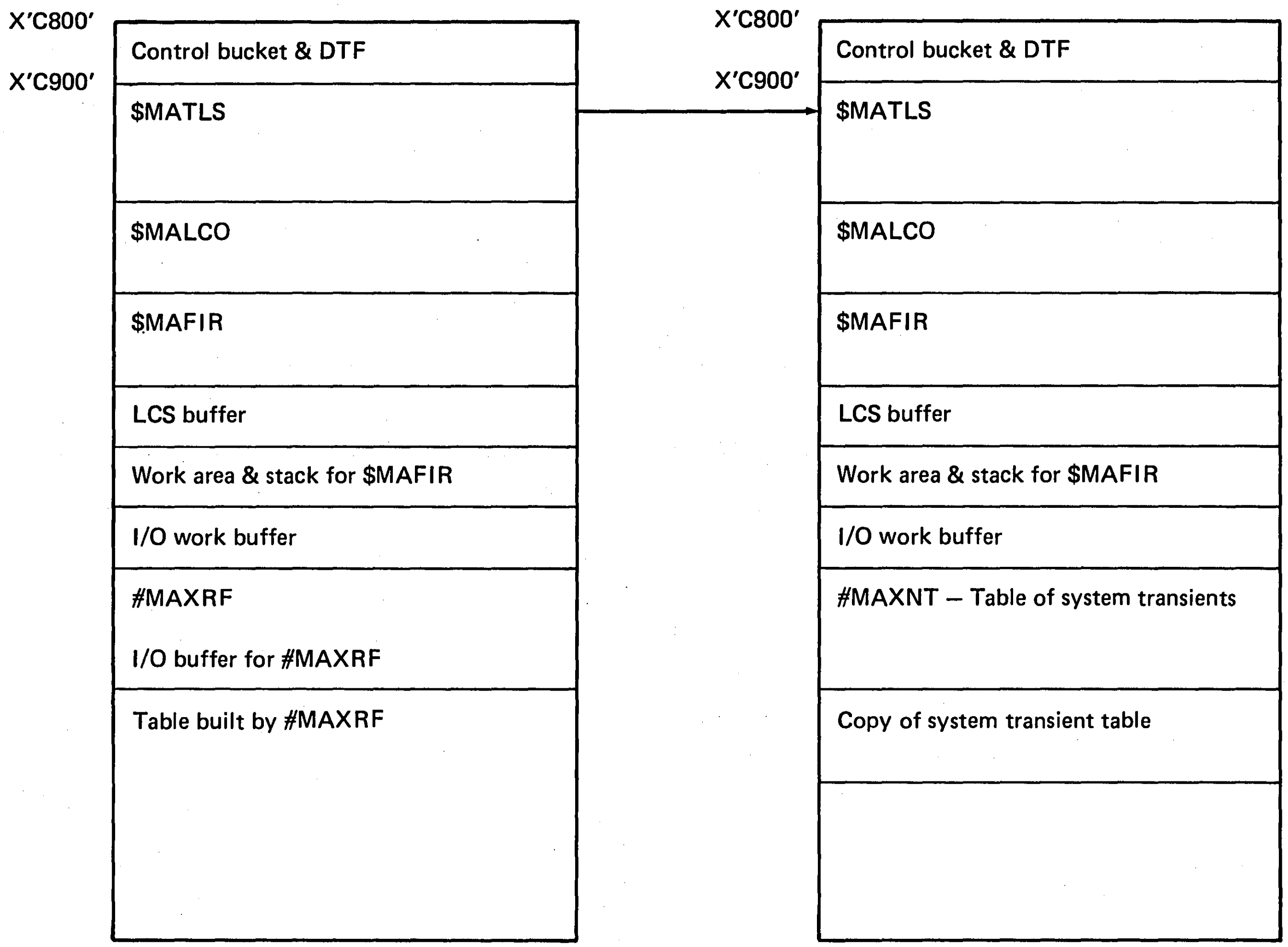


Figure 21-8 (Part 12 of 15). Library Maintenance Utility Storage Maps

For record mode file to a library copy:

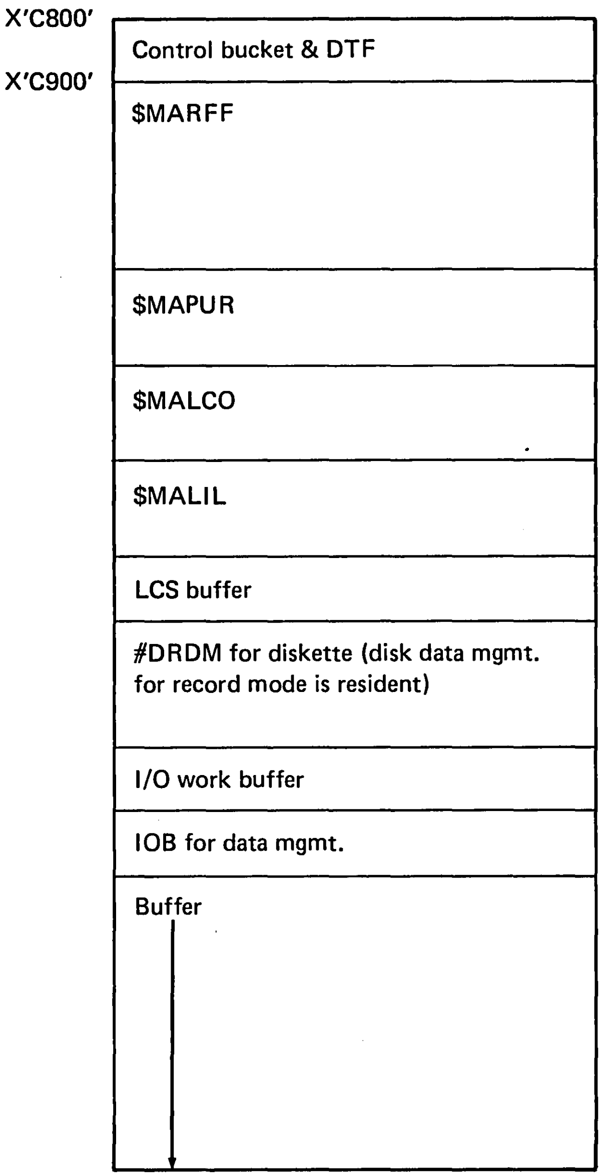


Figure 21-8 (Part 13 of 15). Library Maintenance Utility Storage Maps

For S/32 sector mode file to a library copy:

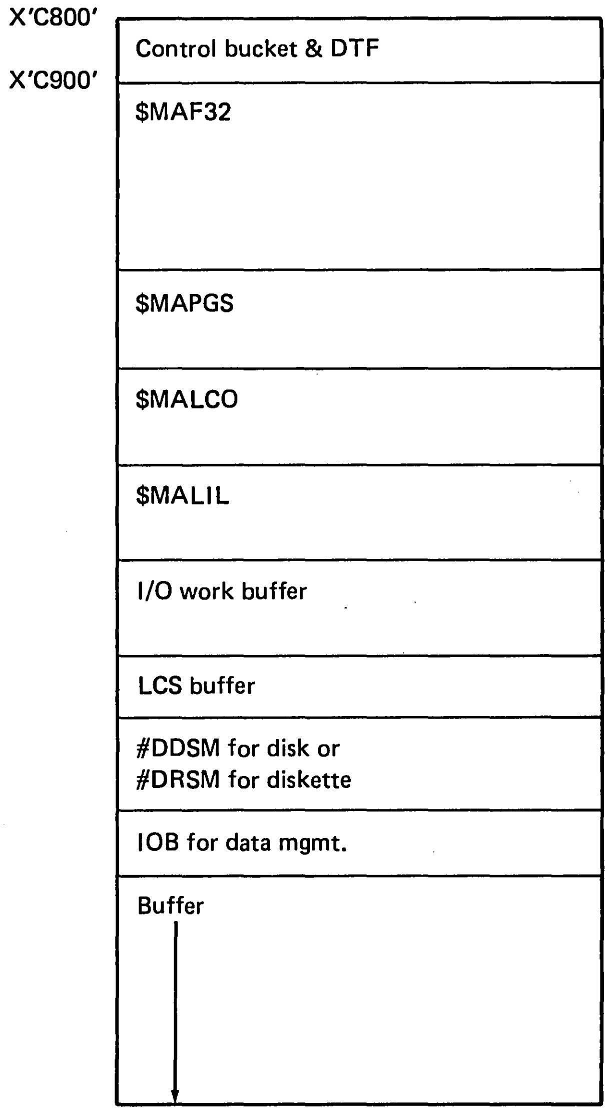


Figure 21-8 (Part 14 of 15). Library Maintenance Utility Storage Maps

For reader to library copy:

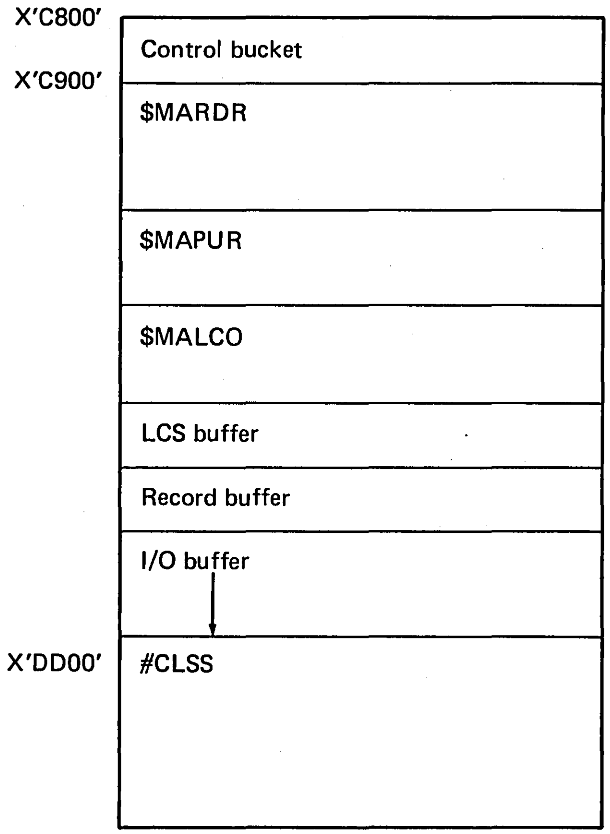


Figure 21-8 (Part 15 of 15). Library Maintenance Utility Storage Maps

Chapter 22. Message Build Utility ($MGBLD)

Introduction

The message build utility ($MGBLD) creates message load members in a disk library. A message load member is a special library load member from which the system support program retrieves the text associated with the message identification code (MIC) specified by the calling program.

The message build utility formats message text records from message text statements in a source member. The utility places the formatted message text records into a work file, allocates disk space for the message member, and copies the formatted messages from the work file to the message load member on disk.

The message build utility consists of the $MGBLD phase and $MGTAB syntax specification module. The program resides in the system library.

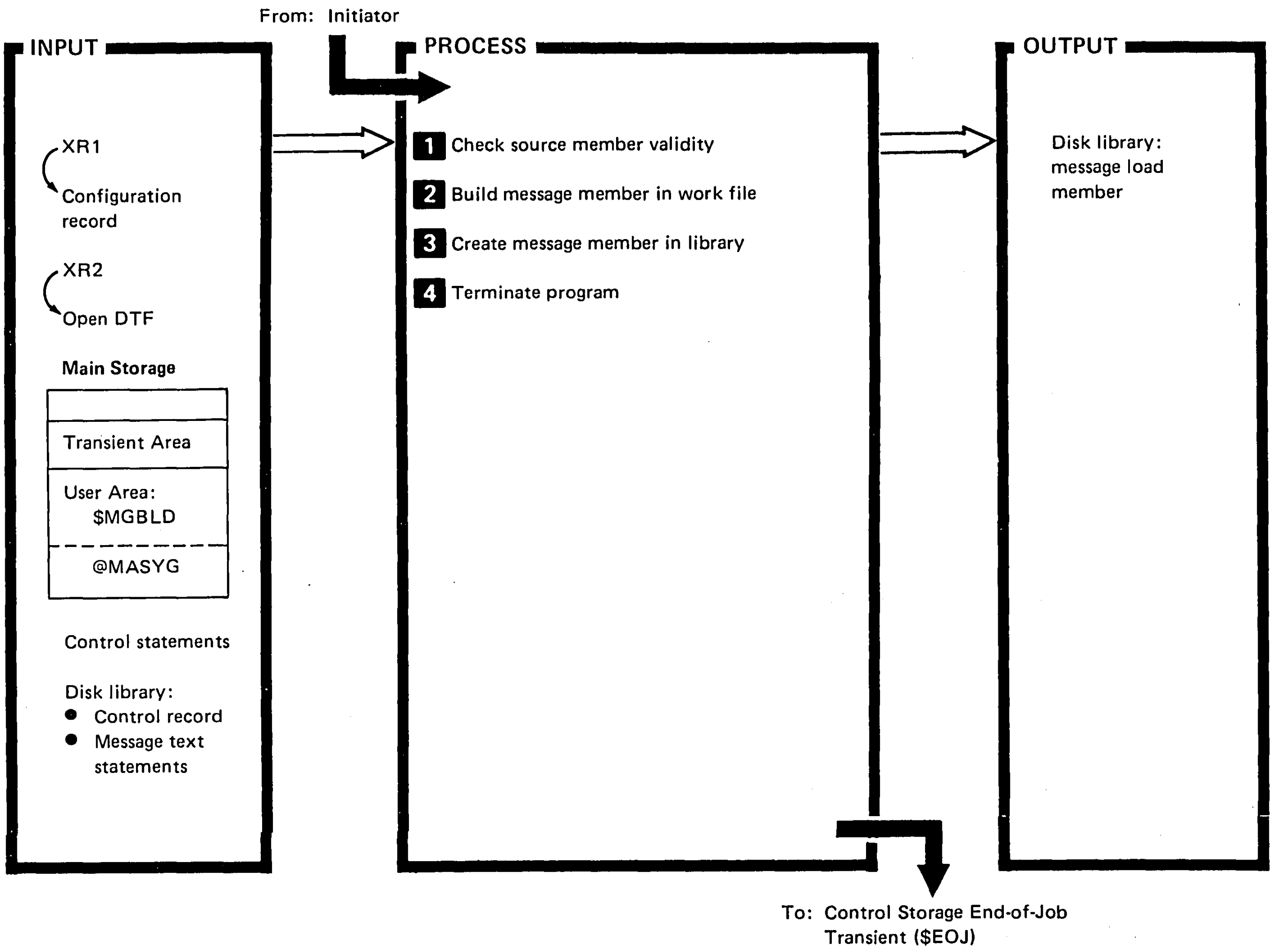
$MGBLD is called by the CREATE procedure or appropriate OCL statements. (See the *System Support Reference Manual* for more information about calling and executing $MGBLD.)

Each sector in the message load member begins with a 2-byte MIC followed by the message text. To issue a message, a program indicates the MIC associated with the required message and uses the message retrieve routine (#MGRET) to retrieve the message text.

$MGBLD requires 14K bytes of main storage for program execution. The system operator may, however, take advantage of more main storage by increasing the region size beyond 14K bytes.

Method of Operation

Diagram 22.1 shows the function of the message build utility.


From: Initiator
INPUT
XR1
Configuration record
XR2
Open DTF
Main Storage
Transient Area
User Area: $MGBLD
@MASYG
Control statements
Disk library:
● Control record
● Message text statements
PROCESS
1 Check source member validity
2 Build message member in work file
3 Create message member in library
4 Terminate program
OUTPUT
Disk library: message load member
To: Control Storage End-of-Job Transient ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Find syntax checker (#USYX).	#MASFN
Read and syntax check control statements.	#USYX
Save REPLACE and SSP parameters in library control block (LCB) for message member.	$MGBLD
If user library specified, find library.	#MAFLB
Return.	$MGBLD
Find source member in user library or #LIBRARY as appropriate.	#MASFN
Get source member from library.	@MASYG
Process source member control statement checking for errors.	$MGBLD
Save message member name and level in LCB.	

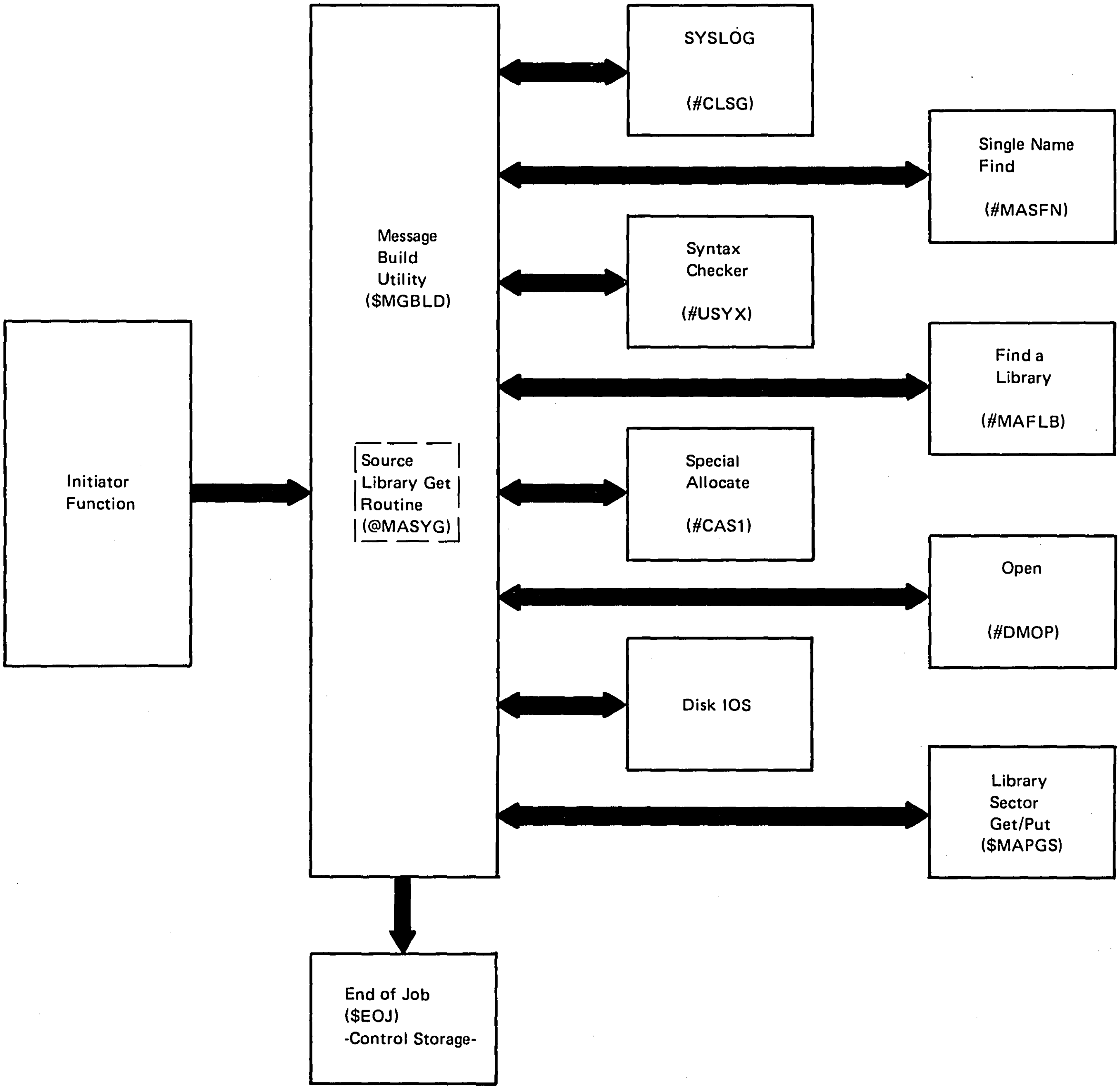
Diagram 22.1 (Part 1 of 2). Perform Message Build Utility Function ($MGBLD)

DESCRIPTION	MODULE/ ROUTINE
If REPLACE not specified, ensure message member does not already exist.	#MASFN
Return.	$MGBLD
2 Allocate work file on disk.	#CAS1
Return.	$MGBLD
Perform dummy open for work file.	#DMOP
Calculate number of work blocks needed to contain work file, then assign largest possible extents.	$MGBLD
Read message text statement from input source member.	@MASYG
Convert MIC to packed decimal format.	$MGBLD
Build message records in I/O buffer in format required by #MGRET.	
Write message text from buffer to work file as buffer becomes full.	Disk IOS
Close work file for output: ● Set up delimiter record.	$MGBLD
▶ ● Write out any data left in work buffer to work file.	Disk IOS ◀
3 Set up work file for sector input by changing IOB fields.	$MGBLD
Set up I/O buffer.	
Find library sector get/put routine (#MAPGS).	#MASFN
Set up message member as object member in library.	#MGBLD
Set up work file IOB for sector input and message member LCB for sector output.	
Fill I/O buffer with sectors from work file.	Disk IOS
Return.	$MGBLD
Write sectors from I/O buffer to message member.	#MAPGS
4 Issue error messages as needed.	#CLXS
Pass control to control storage end-of-job transient ($EOJ) to return control to system.	$MAPGS

Diagram 22.1 (Part 2 of 2). Perform Message Build Utility Function ($MGBLD)

Program Organization

Figure 22-1 shows the control flow of the message build utility.



Initiator Function
Message Build Utility ($MGBLD)
Source Library Get Routine (@MASYG)
SYSLOG (#CLSG)
Single Name Find (#MASFN)
Syntax Checker (#USYX)
Find a Library (#MAFLB)
Special Allocate (#CAS1)
Open (#DMOP)
Disk IOS
Library Sector Get/Put ($MAPGS)
End of Job ($EOJ) -Control Storage-

Figure 22-1. Message Build Utility Control Flow ($MGBLD)

Chapter 23. Disk Reorganization Utility ($PACK/$FREE)

Introduction

The disk reorganization utility reorganizes disk files so that free space is accumulated into one area. The utility uses one of two base modules:

- $PACK provides functional S/32 compatibility; it is called by the COMPRESS procedure or by user OCL.
- $FREE is the S/34 disk reorganization utility; it is called by user OCL and allows the user to specify the spindle and the location of the compressed free area.

The utility consists of four phases: $PACK or $FREE, $FRE1, $FRE2, and $FRE3. Each phase resides in the system library and performs its function before overlaying itself (either totally or partially) with the next phase.

$PACK/$FREE creates a table named FRETB that contains the start and end sector addresses for the current files on disk. (See Figure 23-2 for a description and contents of FRETB.) FRETB entries are then sorted in ascending sequence by start sector addresses. $PACK/$FREE determines the next phase to call:

- If COMPRESS ALL or HIGH is specified, $FREE loads and passes control to $FRE1
- If COMPRESS LOW is specified, $FREE loads and passes control to $FRE2.

$FRE1 or $FRE2 moves the files so they begin at the highest or lowest (as requested) possible sector addresses and all unused space is accumulated at the lowest or highest sector addresses.

$FRE1 may call $FRE2 if COMPRESS ALL was specified. $FRE1 and $FRE2 call $FRE3 to update the disk format-5s.

$FRE3 rebuilds the format-5s to reflect the new disk usage. $FRE3 also indicates in the system communication area that file compression is complete and then passes control to end of job ($EOJ).

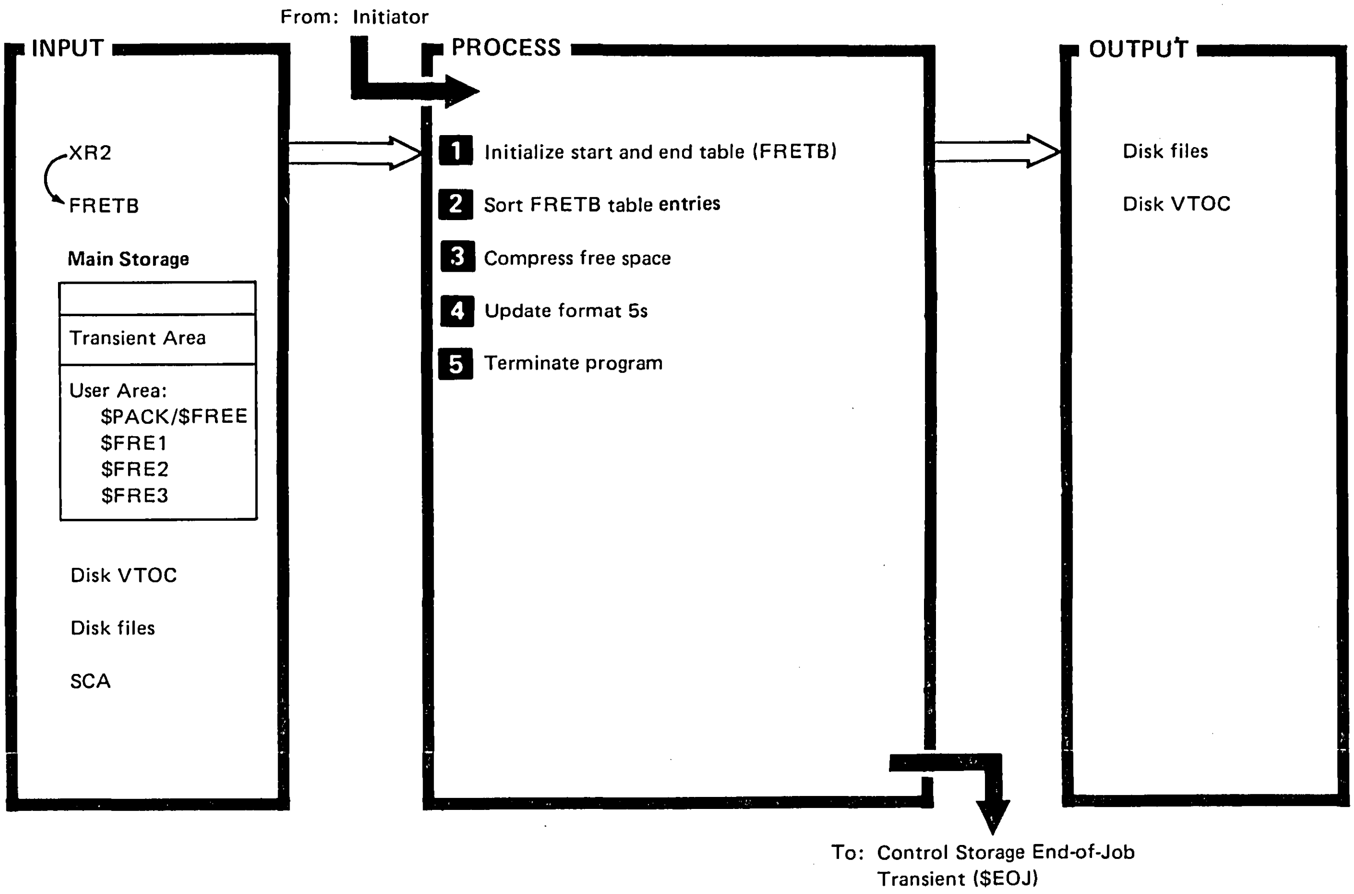
The disk reorganization utility takes necessary precautions to recover data if a system failure occurs. The utility may require rerunning to recover data not permanently in place. If IPL is required to restart the system, the utility is automatically invoked when necessary.

The disk reorganization utility is called by the COMPRESS procedure or by appropriate OCL statements. (See the *System Support Reference Manual* for information about calling and executing this utility.)

The disk compress utility requires 14K bytes of main storage for program execution. The utility must run dedicated and the request must be from the system console.

Method of Operation

Diagram 23.1 shows the function of the disk reorganization utility.



DESCRIPTION	MODULE/ ROUTINE
1 Find syntax checker (#USYX).	#MASFN
Read and syntax check control statements.	#USYX
Check for dual spindle support.	$PACK/ $FREE
Save disk VTOC address in FRETB.	
Read #LIBRARY format 1 (AFA format 1).	
Put begin and end extend in first entry of F1TABLE (in FRETB).	
Use end extend from #LIBRARY format 1 to indicate start of A1 user area.	
Read system format 1s.	Disk IOS
Develop start and end addresses from system format 1s and place in FRETB.	$PACK/ $FREE
Read User format 1s.	Disk IOS
Develop start and end addresses from user format 1s and place in FRETB.	$PACK/ $FREE

Diagram 23.1 (Part 1 of 2). Perform Disk Reorganization Function ($PACK/$FREE)

DESCRIPTION	MODULE/ ROUTINE
2 Sort FRETB table entries in ascending order by start sector addresses. Indicate compress is in process, in case of system failure:	$PACK/ $FREE
● Read format 5s from disk.	Disk IOS
● Save restart information in format 5 and macro save area. ● If restart in process, replace present request with prefailure request.	$PACK/ $FREE
● Write updated format back to disk. Determine module to load next:	Disk IOS
● If compress-all-or-high request, load $FRE1 and go to 3. ● If compress-low request, load $FRE2 and go to 3.	$PACK/ $FREE
3 Set high and low limits for spindle A1 and spindle A2. Locate applicable gaps on requested spindles (first entry in FRETB address table is #LIBRARY format 1 and second entry is first user format 1). Compute gap and file sizes.	$FRE1 or $FRE2
Read format 1s corresponding to files to be moved.	#CSVF
Update format 1 with information needed to move data: ● If restart, show new TO/FROM address and sectors yet to be moved. ● Indicate if file must be moved in segments (file larger than gap). ● Update start of data and start of index (if indexed file).	$FRE1 or $FRE2
Write updated format 1s back to disk.	#CSVF
Return.	$FRE1 or $FRE2
Move data from old disk location to new disk location: ● Read data into I/O buffer. ● Write data from I/O buffer to new location.	Disk IOS
Determine module to load next: ● If compress low load $FRE2 and return to 3. ● If rebuild format 5, load $FRE3, and go to 4.	$FRE1 or $FRE2
4 Read format 5s into main storage.	Disk IOS
Update format 5s to reflect new disk usage.	$FRE3
Write updated format 5s back to disk.	Disk IOS
Indicate compress complete in system communication area (SCA). 5 Pass control to control storage end-of-job ($EOJ). *Note:* Call #CLSG to issue messages as required.	$FRE3

Diagram 23.1 (Part 2 of 2). Perform Disk Reorganization Function ($PACK/$FREE)

Program Organization

Figure 23-1 shows the control flow of the disk reorganization utility. Figure 23-2 shows the displacement equates for the format 1 table.

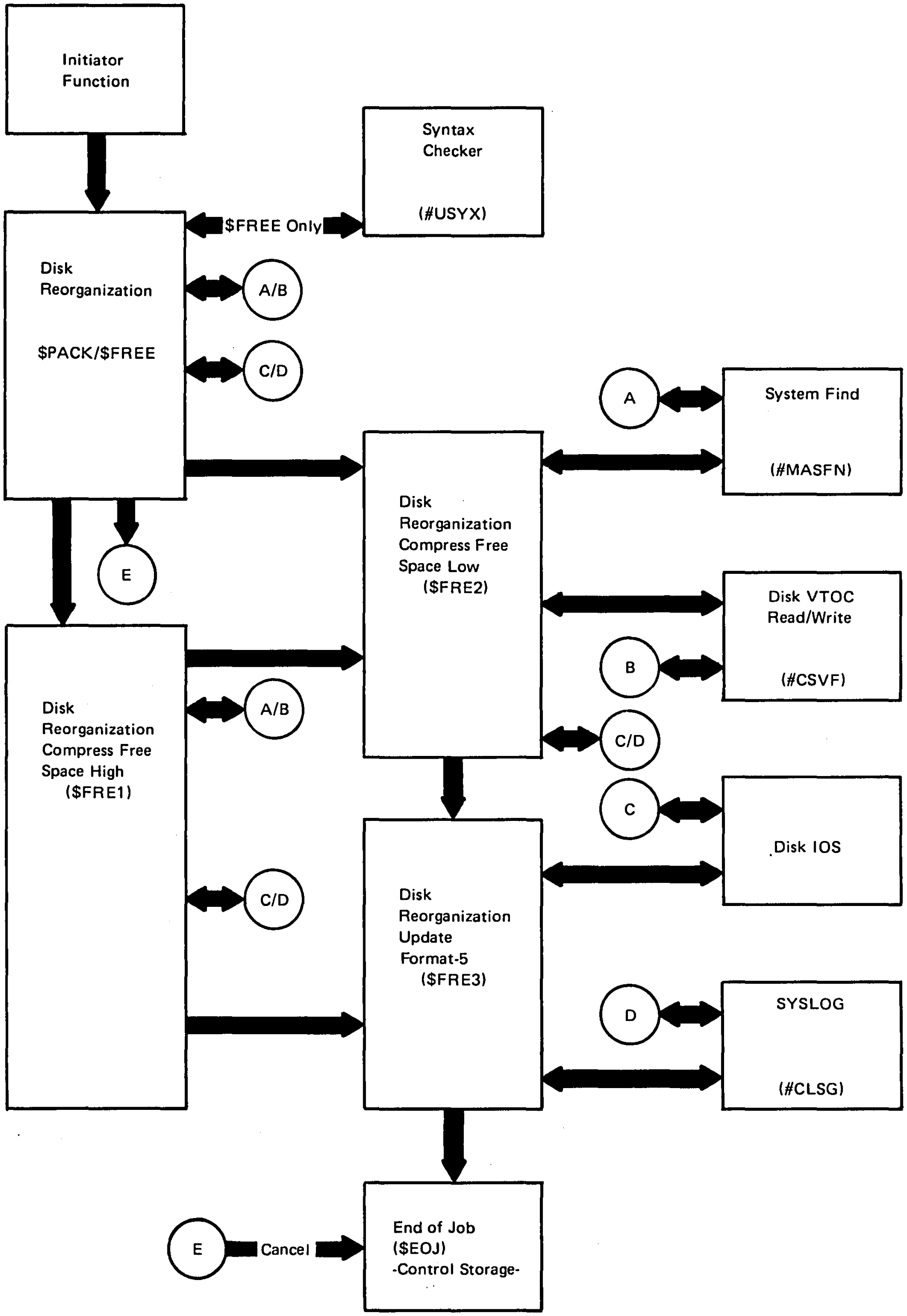


Figure 23-1. Disk Reorganization Utility Control Flow ($PACK/$FREE)

Data Areas

FRETB TABLE

FRETB is a table containing start and end addresses of each file on disk. It is created by $PACK or $FREE and is referenced by $FRE1, $FRE2, and $FRE3. FRETB begins at the start of the user area. Figure 23-2 shows a diagram of the save area and format 1 table within FRETB as well as the contents of SAVEAREA and F1TABLE.

Displacement Equates for Save Area

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes	Description
0	PAKPPSAT	1	Procedure nesting level
1	PAKDSKST	1	Disk status save area: X'8n' = dual spindle X'n0' = 9 megabyte disk X'n1' = 13 megabyte disk
2	PAKSPNDL	1	Spindle request indicator: ALL, X'06' = Request for 'compress all' A1, X'04' = Request for spindle A1 A2, X'02' = Request for spindle A2
3	PAKFREE	1	Free space request indicator: HILOW, X'03' = Request for 'compress all' HIGH, X'02' = Request for free space high LOW, X'01' = Request for free space low
3	PAKSYNTX	2	Syntax save area
5	PAKLSTE@	2	Address of last entry in SSS table
7	PAKACTF1	2	Number of active f-1's (files)

Figure 23-2 (Part 1 of 2). FRETB-File Start and End Address Table

Displacement of Leftmost Byte in Hexadecimal	Label	Length in Bytes	Description
A	PAKVTOC@	3	Start SSS of disk VTOC
0D	PAKSTRA1	3	Start address of A1 user area
10	PAKSTRA2	3	Start address of A2 user area or end SSS address +1 of A1 if A2 not supported
13	PAKENDA2	3	End SSS address +1 of A2 user area
Displacement Equates for Format 1 Table			
2	PAKBEGEX	3	Beginning extent displacement
5	PAKENDEX	3	End extent displacement
7	PAKSDISP	2	VTOC sector displacement
8	PAKSPPRF	1	Spindle preference displacement
8	PAKELDSP	1	Element swap displacement

Figure 23-2 (Part 2 of 2). FRETB-File Start and End Address Table

Chapter 24. Security File Utilities

Introduction

Three security file utilities provide security file maintenance for the System/34 customer. The security file utility ($PROF) updates the security file. The security file save utility ($PRSV) copies the security file from disk to diskette thus providing a backup copy. The security file restore utility ($PRST) copies the backup security file from diskette to disk.

$PROF allows the security officer to:

- Add or delete system and work station operators to the security file
- Change the work station and system operator passwords
- Change the security officer's own password

$PROF allows the master security officer to:

- Add or delete security officers
- Change the security officer's password
- Allocate a new security file if a larger security file is desired

The security file utilities are called by the appropriate OCL statements or procedures. The procedures are:

- PROF to invoke $PROF
- PRSAVE to invoke $PRSV
- PRESTOR to invoke $PRST

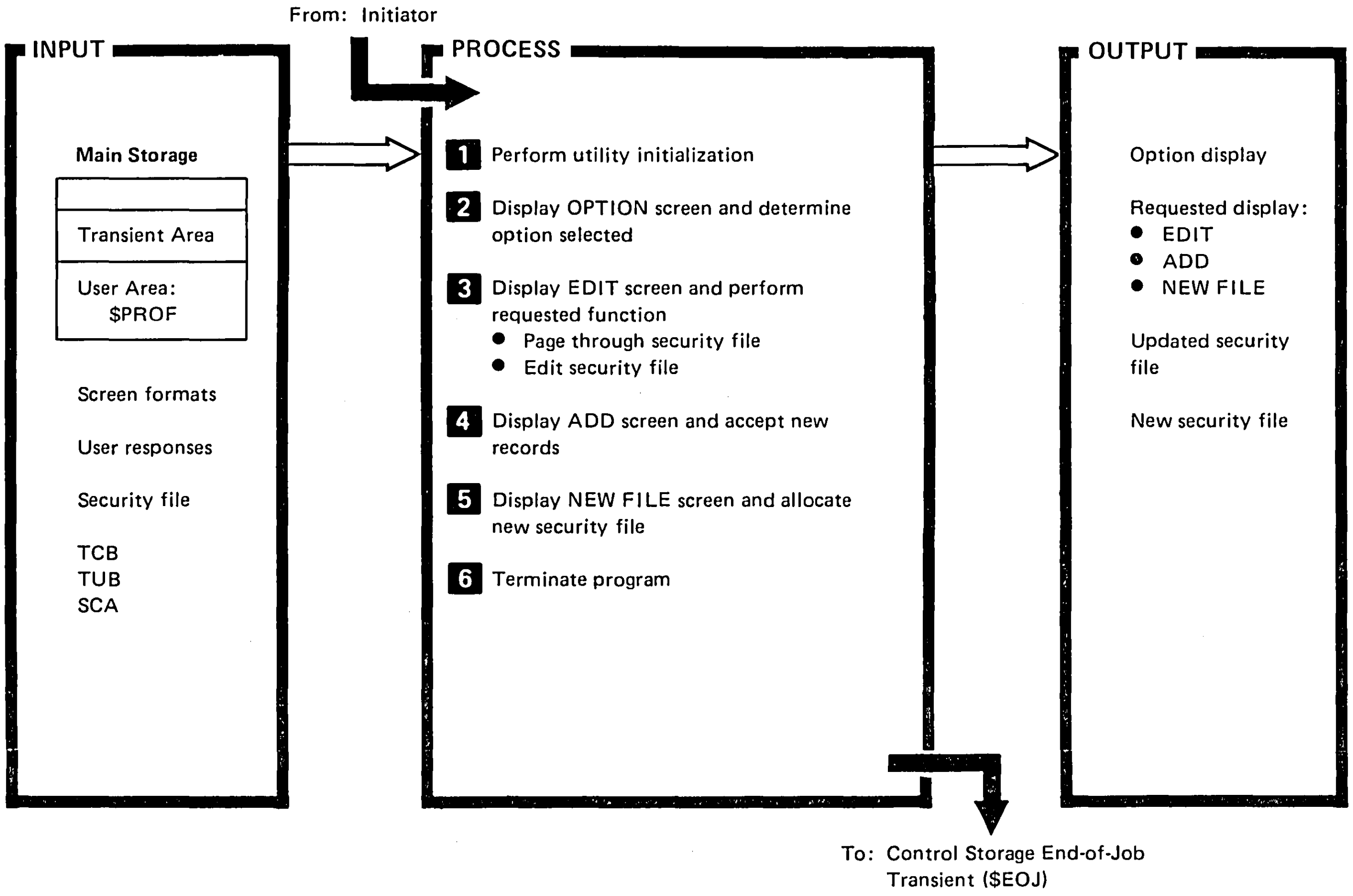
Password security must be in effect on the system before the security file utilities can be run. The system operator must also have the proper security clearance to execute the requested utility function.

$PROF, $PRSV, and $PRST each require 14K bytes of main storage for program execution.

For more information about calling and executing the security file utility programs, see the *System Support Reference Manual.*

Method of Operation

Diagrams 24.1 through 24.3 show the functions of the security file utilities.

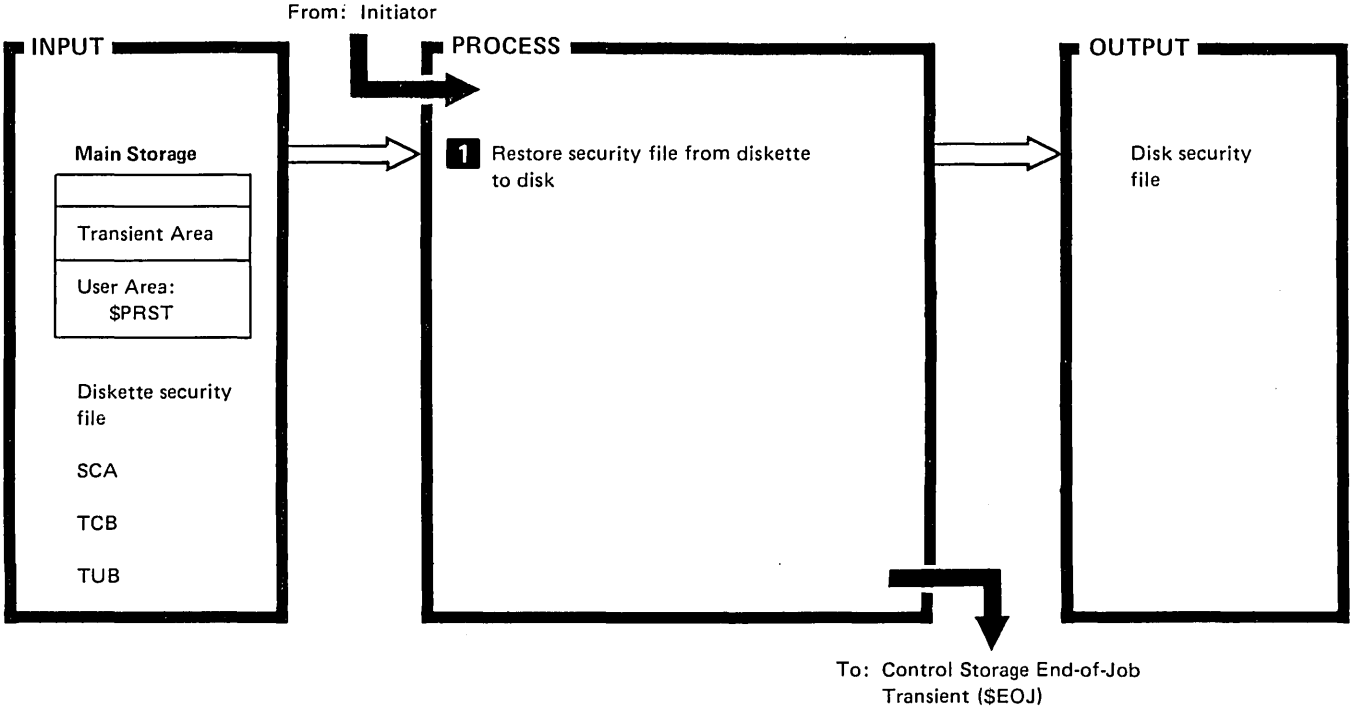

From: Initiator
INPUT
Main Storage
Transient Area
User Area: $PROF
Screen formats
User responses
Security file
TCB
TUB
SCA
PROCESS
1 Perform utility initialization
2 Display OPTION screen and determine option selected
3 Display EDIT screen and perform requested function
● Page through security file
● Edit security file
4 Display ADD screen and accept new records
5 Display NEW FILE screen and allocate new security file
6 Terminate program
OUTPUT
Option display
Requested display:
● EDIT
● ADD
● NEW FILE
Updated security file
New security file
To: Control Storage End-of-Job Transient ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Check system communication area (SCA) to ensure security active (SCAMSEC).	$PROF
Check terminal unit block (TUB) to ensure proper user ID and security class.	
Issue error messages if necessary.	#CLXS
Open work station DTF.	#DMOP
Initialize security file pointers.	$PROF
2 Display OPTIONS screen.	#DWDM
Accept operator response: ● If option 1, EDIT, go to **3**. ● If option 2, ADD, go to **4**. ● If option 3, NEWFILE, go to **5**. ● If option 4, END, go to **6**.	$PROF

Diagram 24.1 (Part 1 of 2). Perform Security File Utility Function ($PROF)

DESCRIPTION	MODULE/ ROUTINE
3 Ensure security file defined.	$PROF
Get sector (4 records) from security file.	Disk IOS
Return.	$PROF
Display EDIT screen and accept operator request:	#DWDM
• If F, FORWARD request, increment pointer into security file and return to 3 to process next security file sector. • If B, BACK request, decrement pointer into security file and return to 3 to process previous security file sector. • If C, CANCEL request, return to 2 to display OPTIONS screen. • If R, RETURN, reset pointer to start of security file and return to 3 to process first sector. • If I, ID advance, find SSS of sector containing requested ID and return to 3 to process sector. • If U, UPDATE:	$PROF
– If D entered, write cleared record to security file after checking user authority. – If update, write updated record to security file after checking user authority.	Disk IOS
Return.	$PROF
Issue error messages if necessary.	#CLXS
Return to 3 to display updated sector.	$PROF
4 Ensure security profile defined. Display ADD screen and accept operator input: • If U, UPDATE, write new record to security file after finding space and checking user authority.	#DWDM
• If C, CANCEL, return to 2 to display options screen.	$PROF
Issue error messages if necessary.	#CLXS
5 Ensure user is master security officer.	$PROF
Display NEW FILE screen and accept operator input:	#DWDM
• If C, CANCEL, return to 2 to display options screen.	$PROF
• If U, UPDATE: – Check input validity. – If user enters new override user-ID and password:	$PROF
a. Read the library member #PRSD.	Disk IOS
b. Insert the new override user-ID and password.	$PROF
c. Write #PRSD backto #LIBRARY.	Disk IOS
– If user requests new security file:	$PROF
a. Deallocate old security file if one exists.	#CAD1
Return.	$PROF
b. Allocate new security file from new data.	#CAS1
Return.	$PROF
c. Write record for master security officer into new security file.	Disk IOS
d. Update security file size in configuration record. – If user requests new master security officer user-ID or password only:	$PROF
a. Read master security officer record.	#Disk IOS
b. Insert new user-ID and password.	$PROF
c. Write master security officer record back to security file.	#Disk IOS
d. Return to 2 to display options screen.	$PROF
6 Close work station DTF.	#DMCL
Pass control to control storage end-of-job transient ($EOJ).	$PROF

Diagram 24.1 (Part 2 of 2). Perform Security File Utility Function ($PROF)

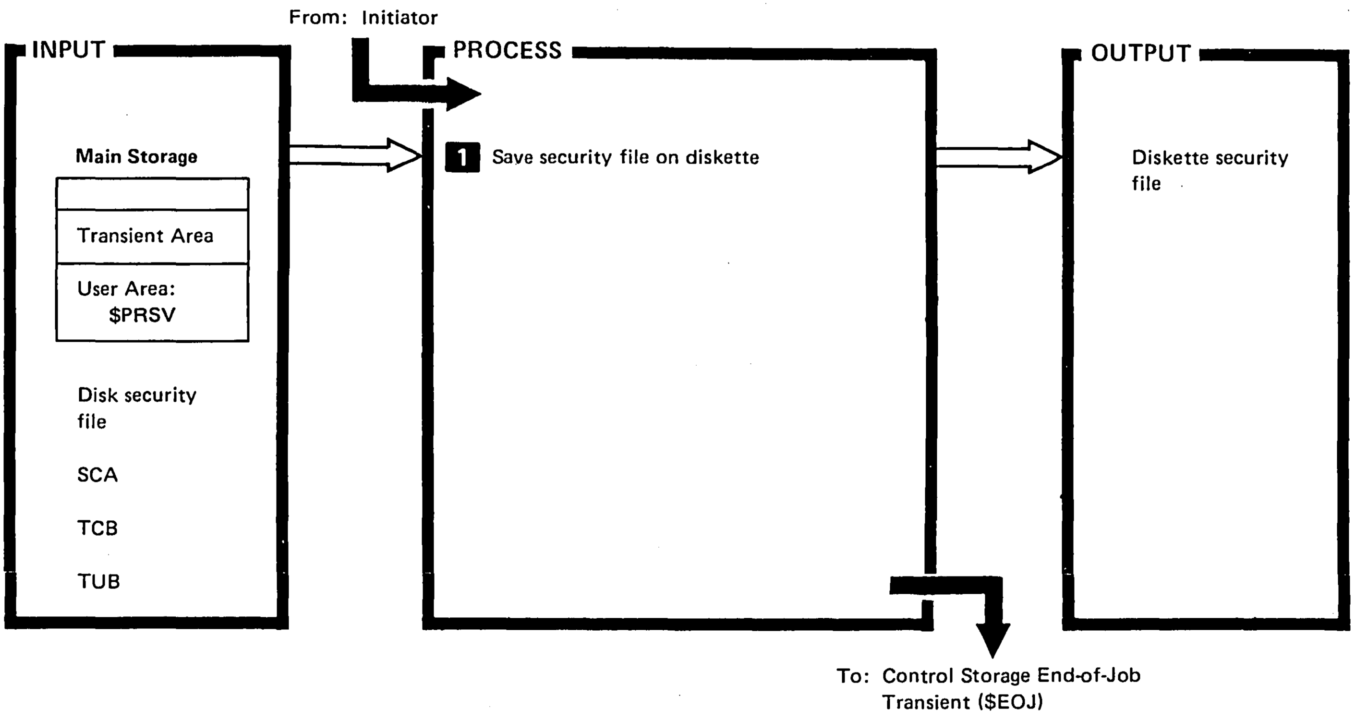


DESCRIPTION	MODULE/ ROUTINE
1 Check system communication area (SCA) to ensure security active (SCZMSEC). Check terminal unit block (TUB) to ensure proper user ID and security class.	$PRST
Issue error message if any.	#CLXS
Allocate diskette file.	#CAML
Return.	$PRST
Read VTOC from disk.	#CSVF
Update VTOC format 1 for security file.	$PRST
Write updated VTOC back to disk.	#CSVF
Return.	$PRST
Open diskette DTF and read first buffer.	#DMOP
Check for errors (see note).	$PRST
Allocate old security file on disk.	#CAS1
Return.	$PRST
Open disk DTF.	#DMOP
Return.	$PRST

Diagram 24.2 (Part 1 of 2). Perform Security File Restore Utility Function (#PRST)

DESCRIPTION	MODULE/ ROUTINE
Read security file from diskette into I/O buffer.	#DRSM
Return.	$PRST
Write security file from I/O buffer to disk.	#DDSM
Return.	$PRST
Close diskette file.	#DMCL
Issue error message as necessary.	#CLXS
Pass control to end of job transient.	$PRST
Note: $PRST must ensure diskette file is valid before special allocate of old disk file. Special allocate clears the disk file.	

Diagram 24.2 (Part 2 of 2). Perform Security File Restore Utility Function (#PRST)



DESCRIPTION	MODULE/ ROUTINE
1 Check system communication area (SCA) to ensure security active (SCAMSEC).	$PRSV
Check terminal unit block (TUB) to ensure proper user ID and security class.	
Issue error message if necessary.	#CLXS
Allocate diskette file.	#CAML
Return.	$PRSV
Read VTOC from disk.	#CSVF
Update VTOC format 1 for security file.	$PRSV
Write update VTOC back to disk.	#CSVF
Return.	$PRSV
Allocate security file on disk.	#CAS1
Return.	$PRSV
Open disk and diskette DTF's.	#DMOP
Return.	$PRSV
Read security file from disk into I/O buffer.	#DDSM
Return.	$PRSV

Diagram 24.3 (Part 1 of 2). Perform Security File Save Utility Function ($PRSV)

DESCRIPTION	MODULE/ ROUTINE
Write security file from I/O buffer to diskette.	#DRSM
Return.	$PRSV
Close disk file.	#DMCL
Issue error messages as necessary.	#CLXS
Pass control to end of job transient.	$PRSV

Diagram 24.3 (Part 2 of 2). Perform Security File Save Utility Function ($PRSV)

Program Organization

Figures 24-1 through 24-3 show the control flow of the security file utilities.

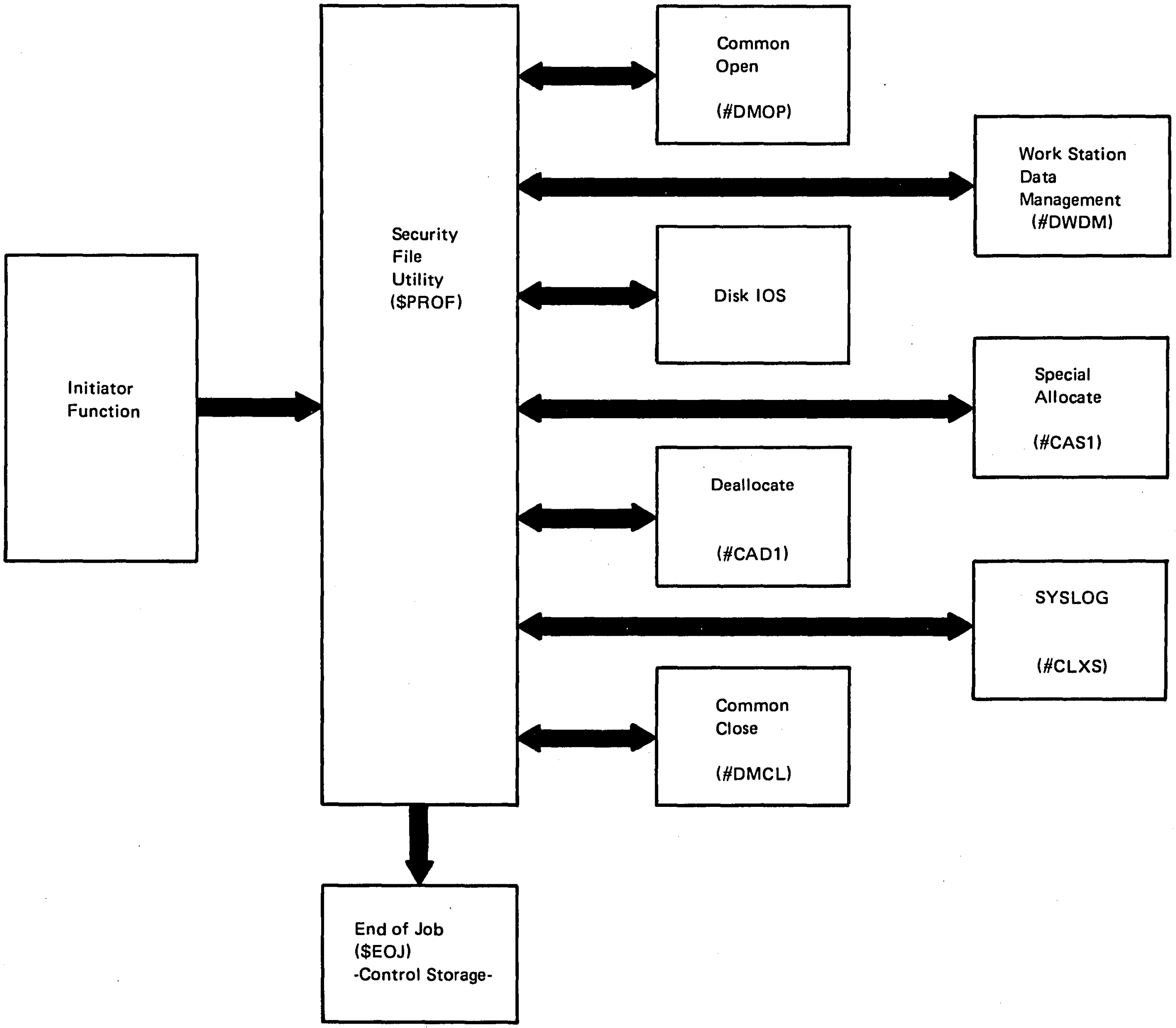


Figure 24-1. Security File Utility Control Flow ($PROF)

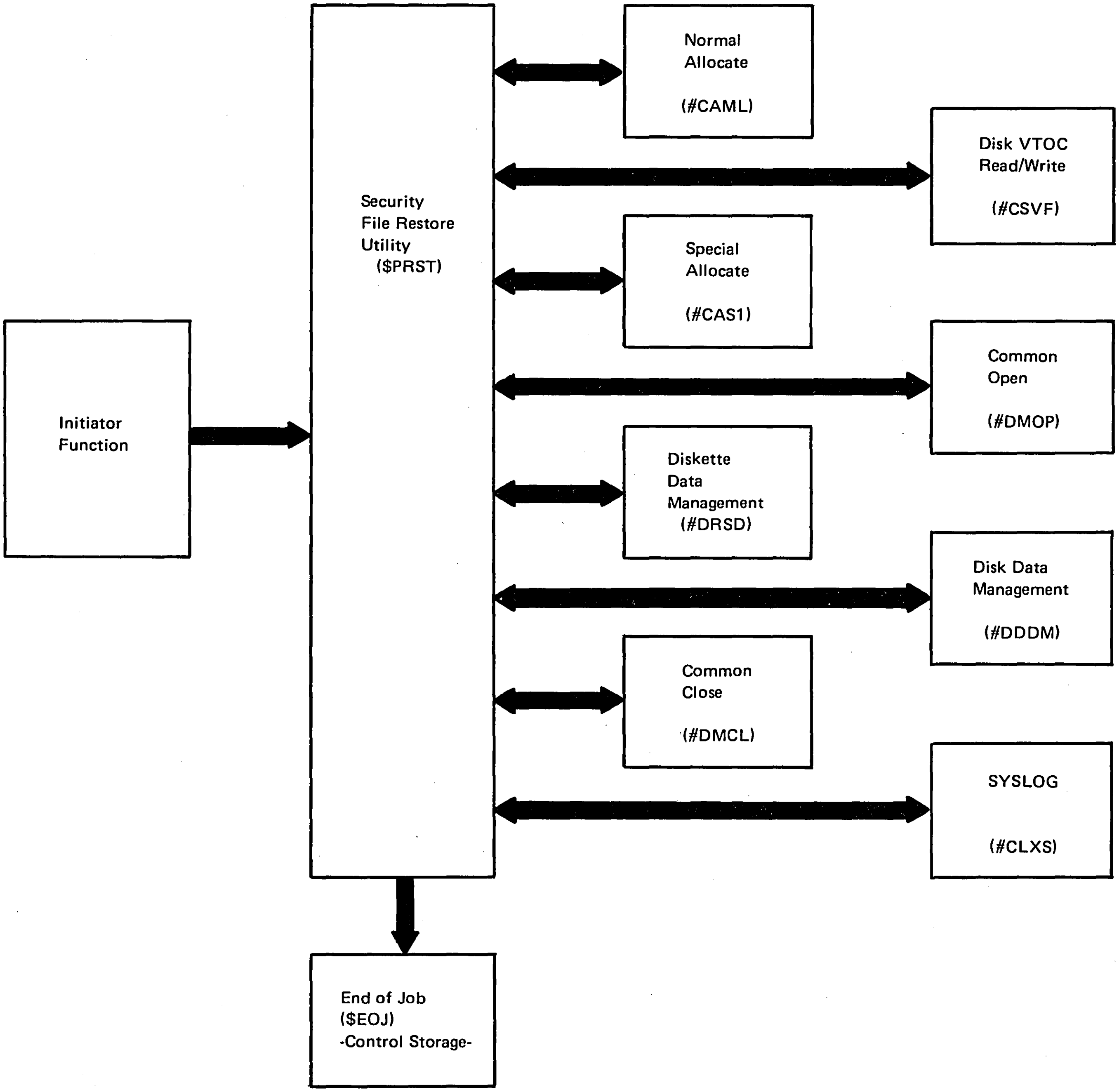


Figure 24-2. Security File Restore Utility Control Flow ($PRST)

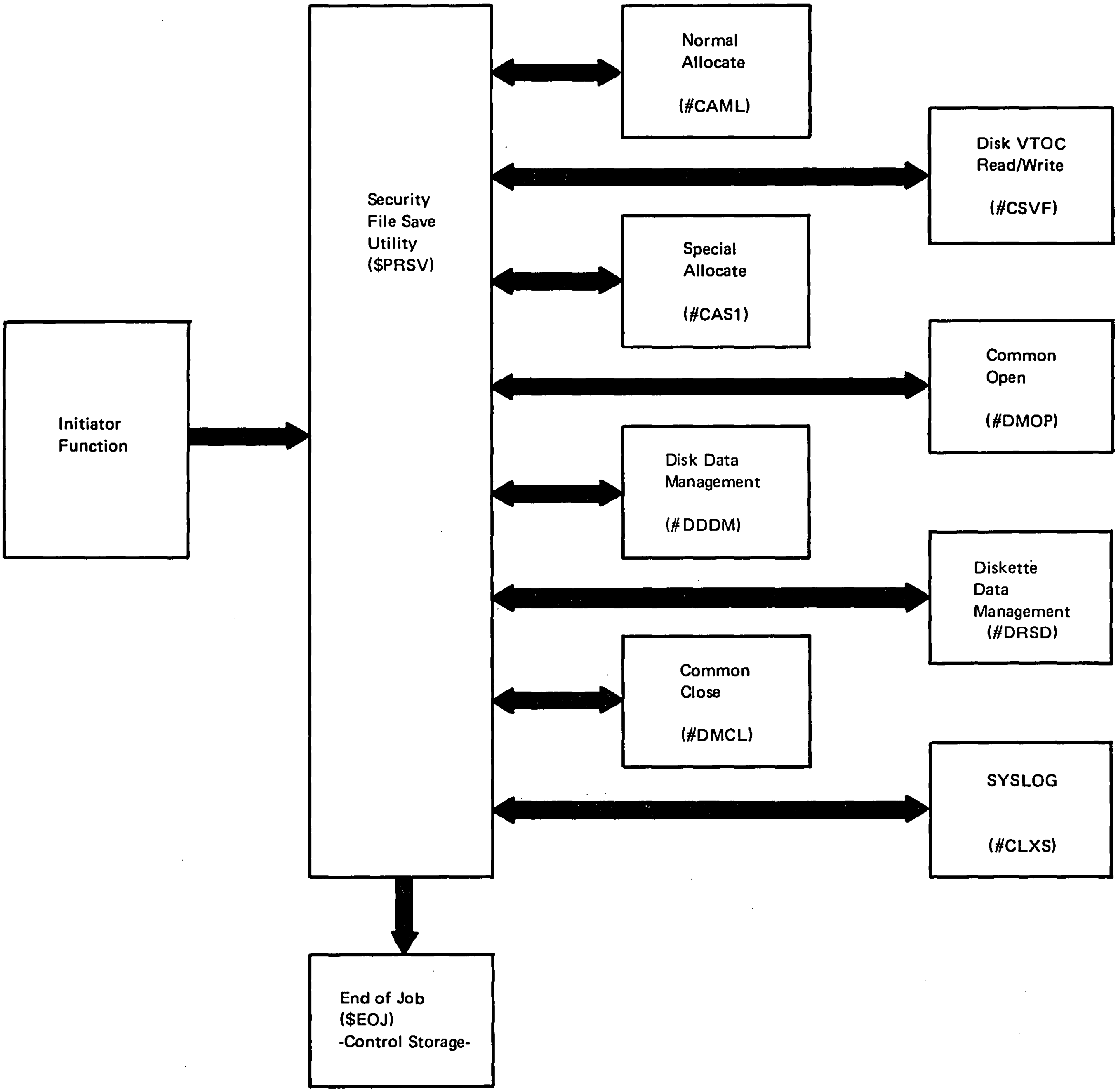


Figure 24-3. Security File Save Utility Control Flow ($PRSV)

Chapter 25. File Rename Utility ($RENAM)

Introduction

The file rename utility ($RENAM) provides a way to change a disk file name. The file identified by label, and optionally by date, is renamed to the specified new name. If no date is specified, the most recently created file is renamed.

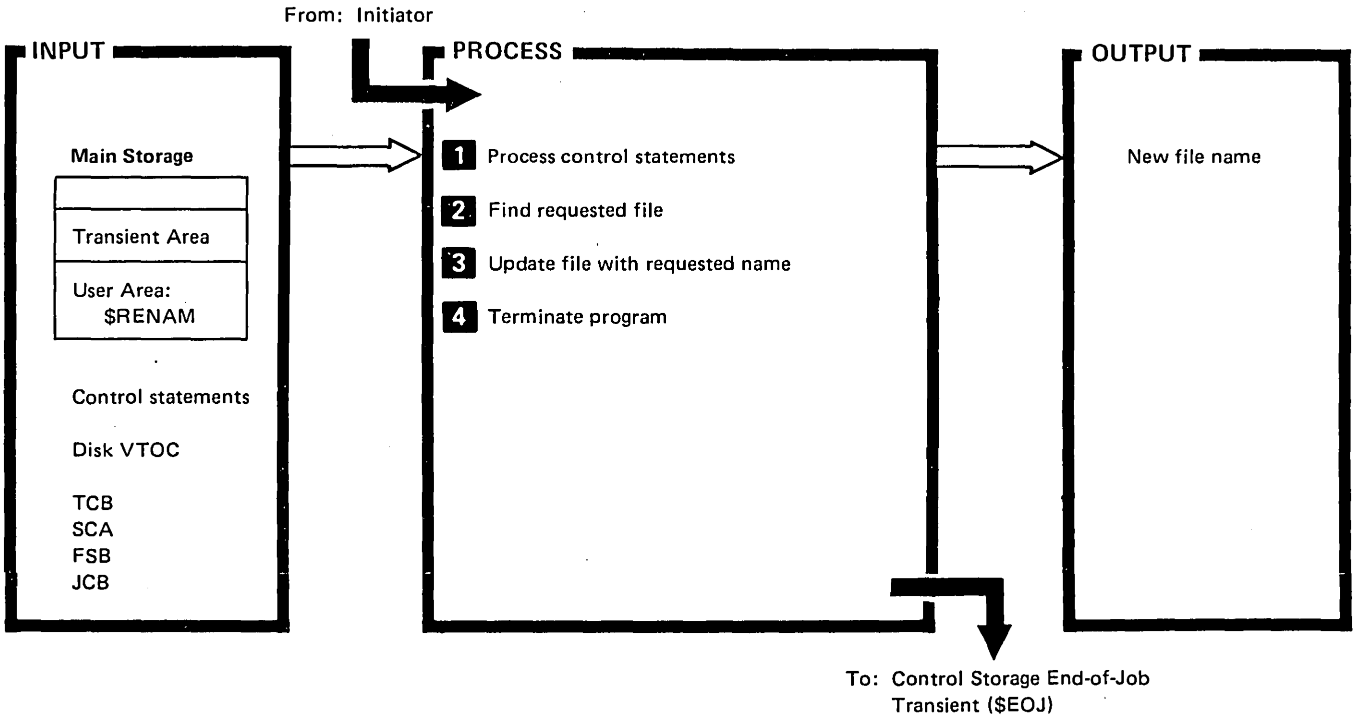
Before the file label is changed, the disk VTOC is searched to ensure the procedure does not create two disk files with the same label.

The utility is called by the RENAME procedure or appropriate OCL statements. (See the *System Support Reference Manual* for more information about calling and executing $RENAM.)

The file rename utility consists of a mainline module ($RENAM) and a syntax specification module ($RETAB). The program resides in the system library.

Method of Operation

Diagram 25.1 shows the function of the file rename utility.



From: Initiator

INPUT

Main Storage

Transient Area

User Area: $RENAM

Control statements

Disk VTOC

TCB
SCA
FSB
JCB

PROCESS

1 Process control statements

2 Find requested file

3 Update file with requested name

4 Terminate program

OUTPUT

New file name

To: Control Storage End-of-Job Transient ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Set old file date to zeros.	$RENAM
Find syntax checker (#USYX).	#MASFN
Use main storage relocating loader (SVC52) to load #USYX.	$RENAM
Read and check syntax of control statements.	#USYX
If END, go to **4**.	$RENAM
2 Check for existing new label in disk VTOC.	#CSVF
Return.	$RENAM
If file with requested new name already exists, issue error message (MIC 6405).	#CLXS
Find file with old label and date.	#CSVF
Return.	$RENAM
If old file does not exist, issue error message (MIC 6406).	#CLXS
Lock format 1 to ensure no change to format 1 area during program execution.	$RENAM
If file active, issue error message (MIC 6408) and unlock format 1.	#CLXS
3 Write new file name to disk VTOC.	#CSVF
Write latest date indicator to disk VTOC if old file is latest file with multiple file same label.	

Diagram 25.1 (Part 1 of 2). Perform File Rename Utility Function ($RENAM)

DESCRIPTION	MODULE/ ROUTINE
Unlock format 1.	$RENAM
Return to 1 to read next control statement.	
4 Pass control to control storage end-of-job transient ($EOJ).	

Diagram 25.1 (Part 2 of 2). Perform File Rename Utility Function ($RENAM)

Program Organization

Figure 25-1 shows the control flow of the file rename utility.

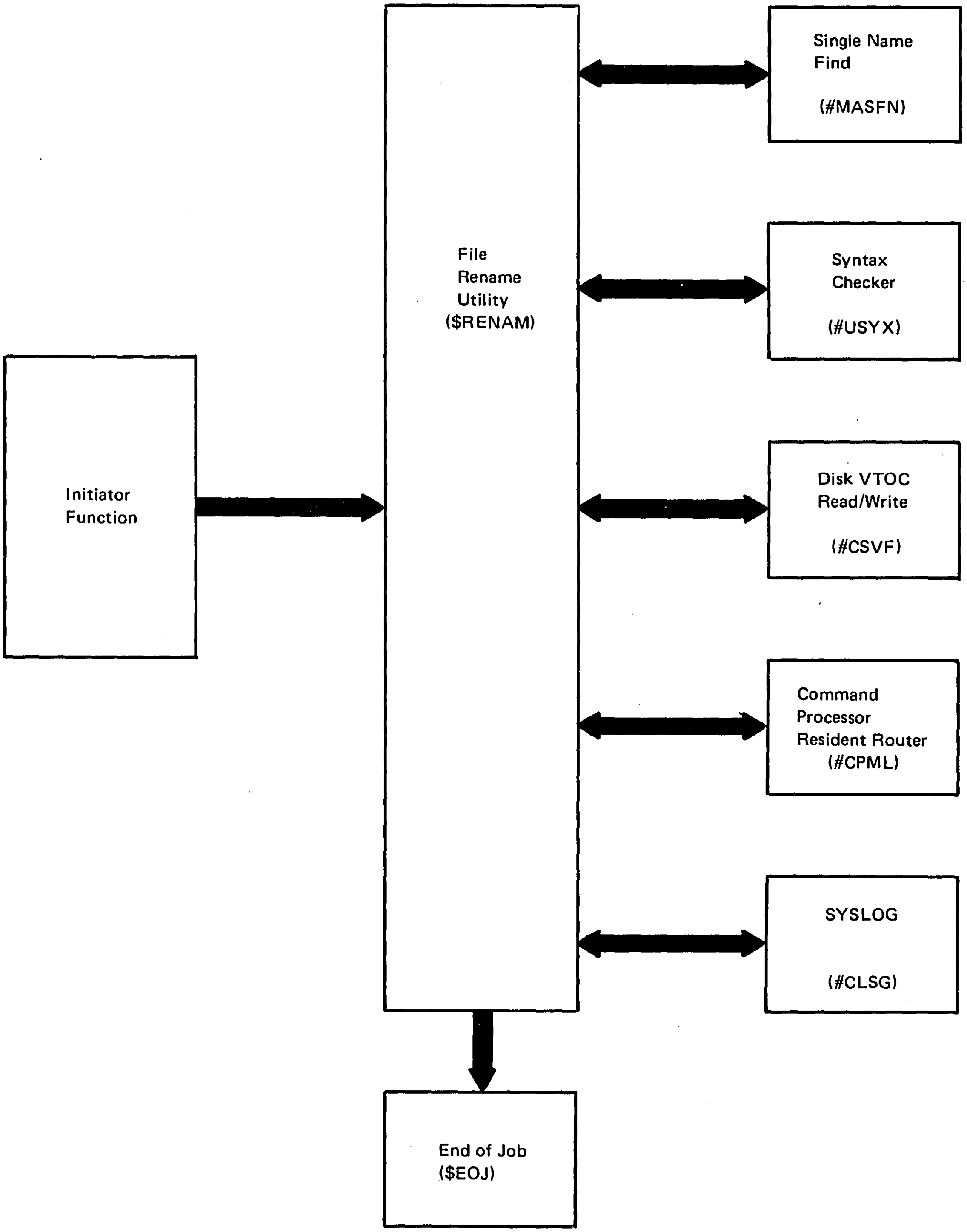


Figure 25-1. File Rename Utility Control Flow ($RENAM)

Introduction

$SETCF is loaded when the current work station configuration record and/or data communications (teleprocessing) configuration records are to be modified.

A system configuration record is built when the system is configured. The first IPL, following a system configuration, builds a work station configuration record and a communications configuration record for each command capable work station by copying the system configuration record to the associated configuration records. $SETCF is then used to modify only the configuration records associated with a work station. The modifications remain in effect until changed by way of $SETCF, or until the system is reconfigured.

$SETCF normally executes as an SRT program, but can be run from the batch job queue. If loaded from the batch job queue, the configurations modified are only in effect for the duration of the job for which $SETCF belongs. If $SETCF is requested as a released program, no diagnostic message is issued, but the net effect is that nothing is modified.

$SETCF is normally loaded by way of the SET, ALTERBSC, or OVERRIDE procedures. (See the *System Support Reference Manual* for information about evoking $SETCF.) The control statement verbs allowed are SETCF, SETB, or SETR.

SETCF allows changing work station configuration items such as:

- Number of lines printed per page
- Print belt image
- Date format
- Designated user library label
- Associated printer ID
- Forms number
- Default job region size
- Date

SETB allows changing data communication configuration items such as:

- Modem speed
- Clocking facility
- Test facility
- Special tone
- Switched or nonswitched line
- Error retries to be attempted

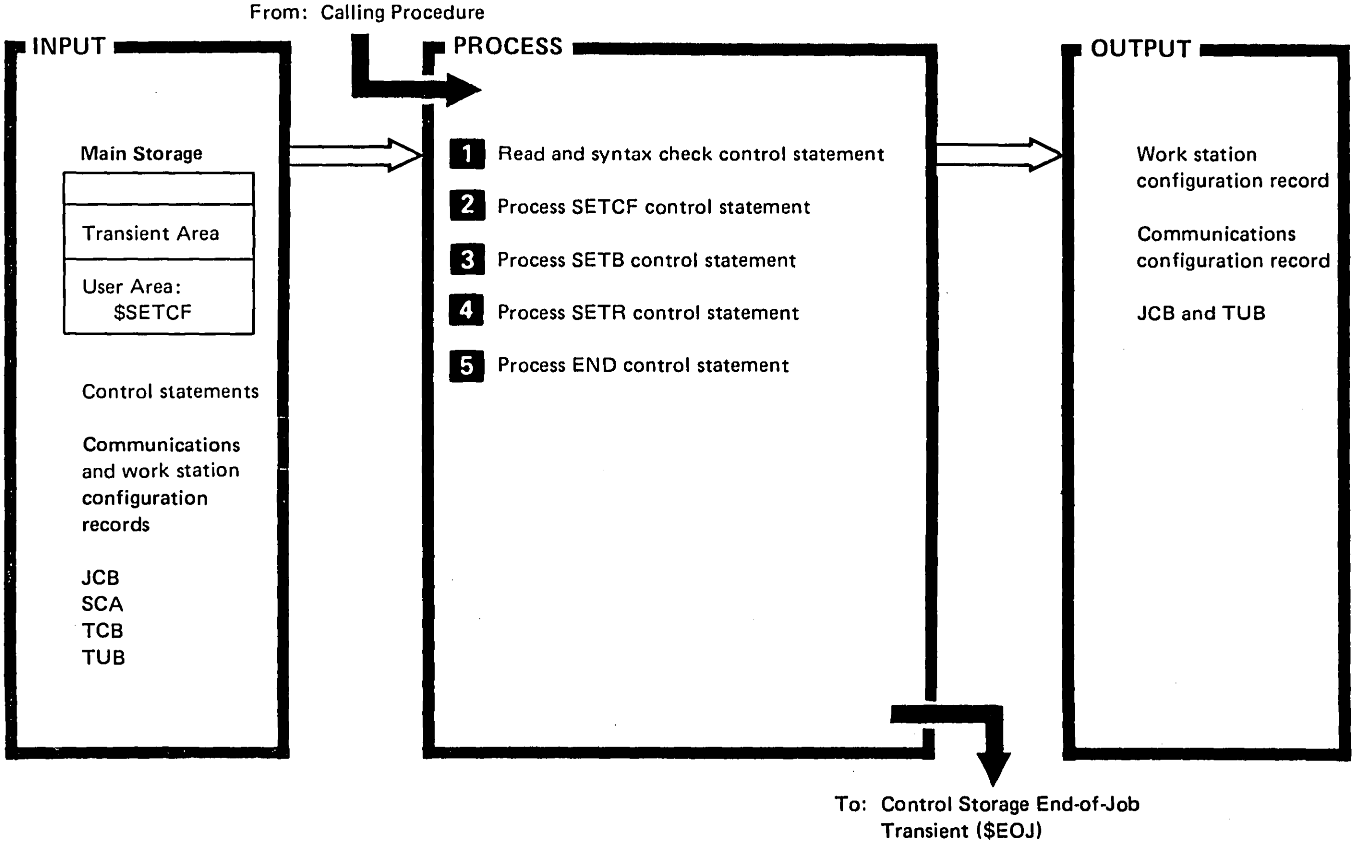
SETR allows changing data communication configuration items such as:

- Line type
  - Point-to-point nonswitched line
  - Line type specified in RPG II program
  - Point-to-point switched line
  - Tributary on multipoint line
- Addressing characters
- Switch type
- Wait time between messages
- Blank compression
- Record separator
- Multiple files
- Local station switched ID
- Remote station switched ID

$SETCF uses the syntax checker (#USYX) to read and check the syntax of the user entered control statements. When the END statement is read, $SETCF calls the end-of-job transient ($EOJ) to terminate the work station configuration procedure.

Method of Operation

Diagram 26.1 shows the function of the work station configuration utility.



From: Calling Procedure

INPUT

Main Storage

Transient Area

User Area:
$SETCF

Control statements

Communications and work station configuration records

JCB
SCA
TCB
TUB

PROCESS

1 Read and syntax check control statement

2 Process SETCF control statement

3 Process SETB control statement

4 Process SETR control statement

5 Process END control statement

OUTPUT

Work station configuration record

Communications configuration record

JCB and TUB

To: Control Storage End-of-Job Transient ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Build system find parameter list to find syntax checker (#USYX).	$SETCF
Find #USYX.	$MASFN
Load #USYX using main storage relocating loader (SVC-52). Branch to #USYX.	$SETCF
Read and syntax check control statement.	#USYX
Check verb from control statement and if: ● SETCF, to to **2**. ● SETB, go to **3**. ● SETR, go to **4**. ● END, go to **5**. **2** Find job control block (JCB). Use task work area (TWA) get function to read work station configuration record from disk. Process control statement parameter given: ● LINES – update JCB and configuration record. ● FORMAT – update JCB and configuration record.	$SETCF

Diagram 26.1 (Part 1 of 3). Perform Work Station Configuration Processing ($SETCF)

DESCRIPTION	MODULE/ ROUTINE
● DATE — update JCB and configuration record. ● PRINTER-SYS: – Set ID to X'FFFF'. – Update JCB and configuration record. ● PRINTER-WSID:	$SETCF
– Issue halt message if invalid ID.	#CLXS
– Move printer ID to terminal unit block (TUB). – Update JCB and configuration record. ● FORMS NO — update JCB and configuration record. ● Default region size:	$SETCF
– Issue halt message if region size invalid.	#CLXS
– Update JCB and configuration record. ● User library:	$SETCF
– Issue halt message if #LIBRARY specified.	#CLXS
– If LIBRARY-0 specified, zero out library name in configuration record. – Move library label to configuration record. Use TWA put function to return updated configuration record to disk. Return to 1 to read next control statement. 3 Use TWA get function to read data communications configuration record from disk. Examine control statement for parameters given: ● BRATE ● CLOCK ● ERC (error retry count) ● SLINE ● TEST ● TONE Update configuration record for parameters given. Use TWA put function to return updated configuration record to disk. Return to 1 to read next control statement. 4 Use TWA get function to read data communications configuration record from disk. Process control statement parameters given: ● MLTFL — update configuration record. ● LINE-S or LINE-C:	$SETCF
– If no SWTYP given, issue halt message.	#CLXS
– If SWTYP given, update configuration record with SWTYP and LINE information. ● WAIT — update configuration record. ● RCSP: – Ensure good value given. – Convert value if necessary.	$SETCF
▶ – Issue halt message if invalid value.	#CLXS ◀
– Update configuration record. ● LOCID: – Ensure good value given. – Convert value if necessary.	$SETCF
– Issue halt message if invalid value.	#CLXS
– Update configuration record. ● REMID: – Ensure good value given. – Convert value if necessary.	$SETCF
– Issue halt message if invalid value.	#CLXS
– Update configuration record.	$SETCF

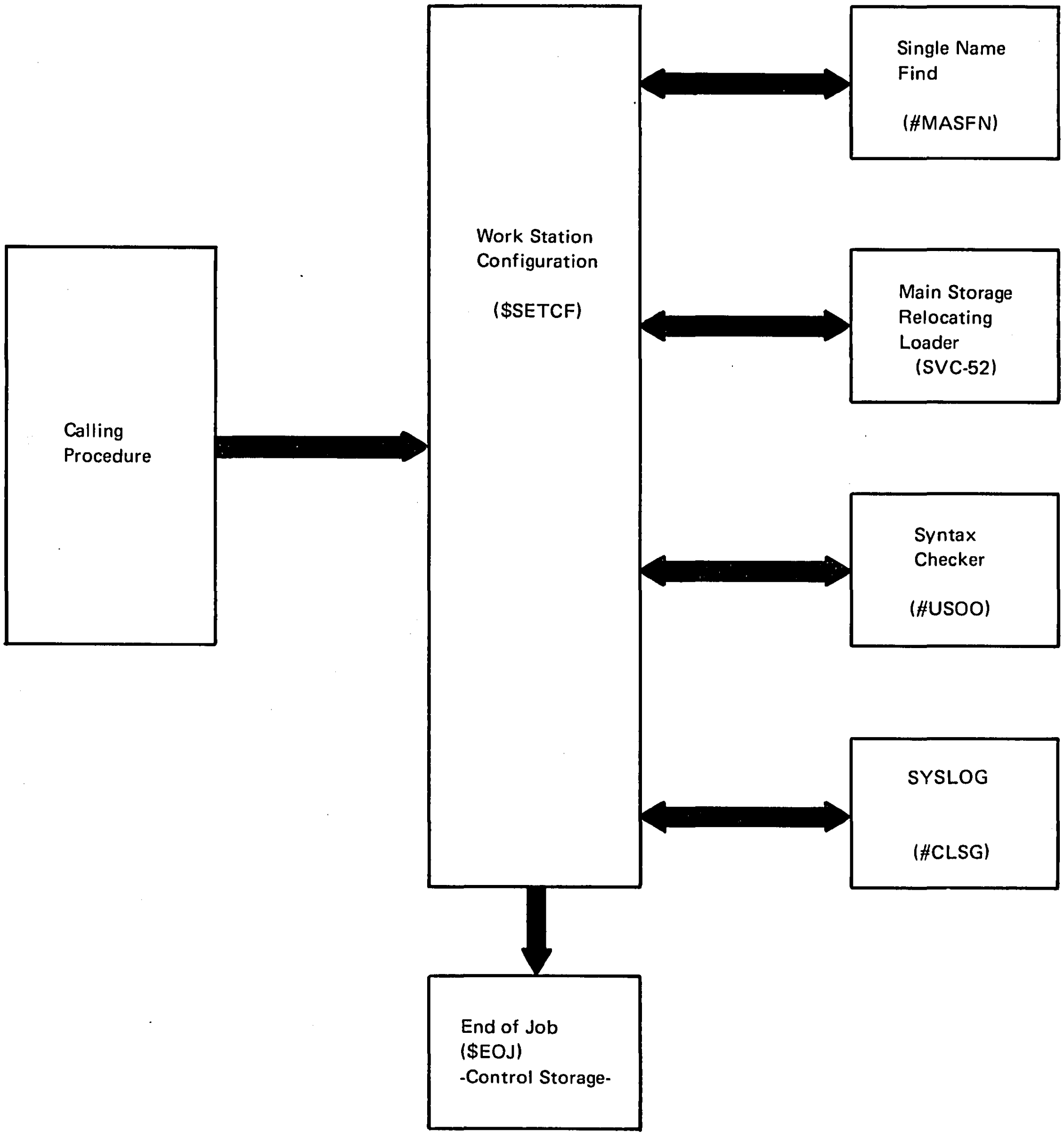
Diagram 26.1 (Part 2 of 3). Perform Work Station Configuration Processing ($SETCF)

DESCRIPTION	MODULE/ ROUTINE
● SWTYP: – Only valid for Line-S or C. Use TWA put function to return updated configuration record to disk. Return to 1 to read next control statement. 5 Pass control to control storage end of job ($EOJ).	$SETCF

Diagram 26.1 (Part 3 of 3). Perform Work Station Configuration Processing ($SETCF)

Program Organization

Figure 26-1 shows the control flow of the work station configuration utility.



Calling Procedure
Work Station Configuration ($SETCF)
Single Name Find (#MASFN)
Main Storage Relocating Loader (SVC-52)
Syntax Checker (#USOO)
SYSLOG (#CLSG)
End of Job ($EOJ) -Control Storage-

Figure 26-1. Work Station Configuration Control Flow ($SETCF)

Chapter 27. Screen-Format Generator Utility ($SFGR)

Introduction

An application programmer who wishes to use the display screen as a formatted input/output device must have a display format in a load member in the library. The screen format generator routine (SFGR) generates these formats from user-created library source members. The generated formats are used by work station data management whenever the application program is interfacing with a terminal.

The screen format generator routine performs the following functions for the application programmer:

- Reads display format specification statements from a library source member.

- Produces a printout of the specification statements, analyzes the specifications for errors, and prints diagnostic error messages.

- Builds the display format as a two part table that includes:
  - A field descriptor table (FDT) consisting of an entry that describes attributes for the entire display and a table of entries that describes attributes of fields in the format. (Only entries needed by WSDM are included in the FDT.)
  - A data stream consisting of orders, commands, and data (if provided) for the screen display (see *IBM System/34 Functions Reference Manual*, SA21-9243).

  The screen format is placed into a load member. The load member may contain one or all of the formats for the application program. When an application program is evoked, the application program must open each load member containing formats used by the program.

- Provides, if data is required during execution by the application program using the display format, a printout of those fields in the order they must appear in the output record area.

- Provides a printout of all fields defined for input and the order in which they will appear in the input record area.

- Calculates and prints the input and output record lengths to be used during execution by the application program.

- Catalogs the display formats provided for a particular application program in an object member in the library. The object module containing the formats for an application program have the load member name specified in the utility control statement.

The screen format generator routine program ($SFGR) is initiated by way of an OCL statement or the FORMAT command. (See *System Support Reference Manual* for information about calling $SFGR.)

When the application programmer calls the screen format generator routine (SFGR), the mainline module ($SFGR) is first to receive control (see Figure 27-2).

$SFGR performs the initial checks and verification for the screen format generator routine. It ensures load member names are valid, utilizes the syntax checker (#USYX) to check control statement validity, and checks that the LOADMBR statement follows immediately after the RUN statement. It calls the library find routine ($MAFND) to ensure that the screen format load member and source member exist; it also calls the find specified library routine (#MAFLB) to determine if the specified library exists when the INOUT verb is given and checks the screen format load member index for the specified format name when DELETE is specified. The screen format communications area along with the format load member index are then moved to the beginning of the task area. Figure   is a storage map of SFGR.

If the only operation requested is DELETE, $SFGR passes control to the screen format load member maintenance routine ($SFLMM). If other operations are requested, $SFGR passes control to the diagnose errors and print routine ($SFDEP).

$SFDEP diagnoses errors in the input source specifications and prints requested information when PRINT-NO is not specified. It calls the source get transient (#MASGT) to fetch the source specifications. $SFPED checks the source specifications for valid values and valid combinations. The print function is performed by the system list transient (#CLST). #CLST prints the source specification records, input and output record area formats, the field's location in the buffer, and the field length. If necessary, #CLST also prints error messages. $SFDEP then passes control to the FDT and data stream build routine ($SFFDB).

$SFFDB passes control to special allocate (#CAS1) to allocate a scratch work file. $SFFDB then builds the data stream and field description table (FDT). It sets up buffers for the data stream and FDT, searches the communications table for CREATE, ADD, or UPDATE entries to determine operation requested, and sets up #SYSG to retrieve screen S entries. $SFFDB places information such as write command, start of header order, screen control character, start of header data, insert cursor order, and repeat to address entity into the data stream buffer if needed. #MASYG then retrieves detail field definition D records. $SFFDB places information from the D records into the data and FDT buffers. Subroutines within $SFFDB are used as required to build screen attribute, move row/column number to the insert cursor order, and build field format words. FDT entries are moved to the FDT buffer and when the FDT buffer is full, $SFFDB calls #DDSM to write the buffer to a temporary work file. Processing continues until the record type changes back to S (only one S record per record group) or end of file is reached. At this point, $SFFDB writes the balance of FDT entries and data stream to the temporary work file.

If data fields are in ascending screen sequence, $SFFDB passes control to $SFLMM. If, however, the data fields require resequencing, $SFFDB passes control to the data stream resequence routine (SFRSQ).

$SFRSQ arranges the input data fields in ascending screen sequence and ensures that trailing screen attributes of output fields do not overlay starting screen attributes of any other field in the display. It also allocates storage space for input and outpub buffers, searches the communications table for the data stream that requires resequencing, assigns pointer values in the data stream, and then resequences the data stream. Once the data stream is resequenced, $SFRSQ writes it from the buffer back to the work file. $SFRSQ then passes control to the screen format load member maintenance routine ($SFLMM).

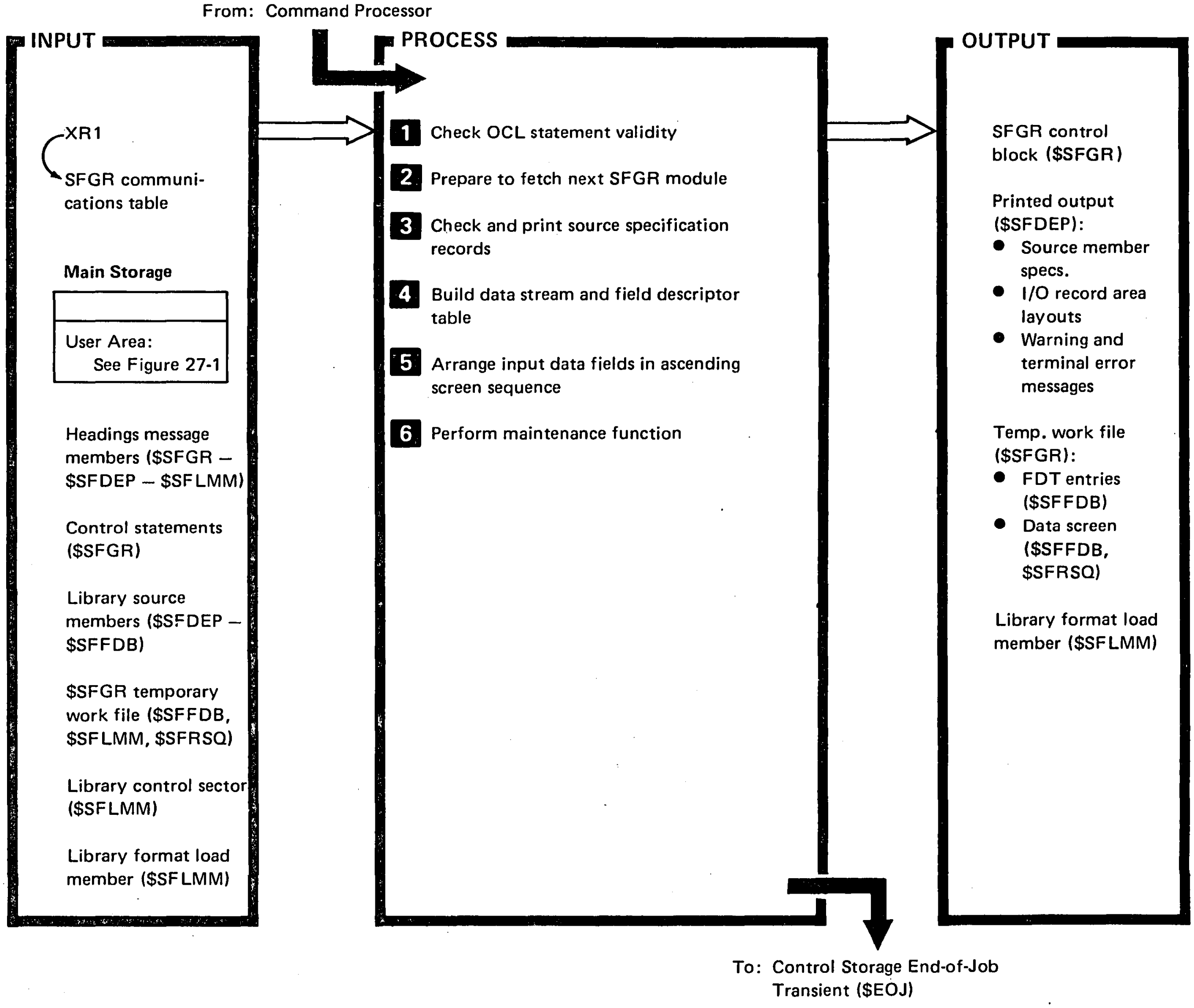
$SFLMM performs the maintenance functions required to build the display screen format in the library format load member. It searches the SFGR communications table and format index of the old format load member for formats that must be copied from the old format load member to the work file. The sector get/put access routine ($MAPGS) is used to read the old formats from the load member. $SFLMM resolves address displacements in the format indexes in the work file. The file is then read into the buffer and $MAPGS is used to write the formats to the library load member.

After the new screen formats are in the library format load member, $SFLMM prints the size of each format in the load member and calls the end-of-job transient to terminate the screen format generator routine.

Figure 27-1 shows a storage map of the screen format generator routine.

Method of Operation

Diagram 27.1 shows the function of the screen format generator utility.


From: Command Processor

INPUT

XR1

SFGR communications table

Main Storage

User Area: See Figure 27-1

Headings message members ($SFGR – $SFDEP – $SFLMM)

Control statements ($SFGR)

Library source members ($SFDEP – $SFFDB)

$SFGR temporary work file ($SFFDB, $SFLMM, $SFRSQ)

Library control sector ($SFLMM)

Library format load member ($SFLMM)

PROCESS

1 Check OCL statement validity

2 Prepare to fetch next SFGR module

3 Check and print source specification records

4 Build data stream and field descriptor table

5 Arrange input data fields in ascending screen sequence

6 Perform maintenance function

OUTPUT

SFGR control block ($SFGR)

Printed output ($SFDEP):
- Source member specs.
- I/O record area layouts
- Warning and terminal error messages

Temp. work file ($SFGR):
- FDT entries ($SFFDB)
- Data screen ($SFFDB, $SFRSQ)

Library format load member ($SFLMM)

To: Control Storage End-of-Job Transient ($EOJ)

DESCRIPTION	MODULE/ ROUTINE
1 Retrieve heading messages 2400, 2406, and 2407 to ensure valid load member names.	#MGRET
Load syntax checker (#USYX) into main storage.	$SFGR
Clearn output table area.	
Branch to #USYX to get and check control statement.	#USYX
Check return code from #USYX for valid control statement and call SYSLOG if statement invalid.	$SFGR
If statement after RUN not LOADMBR, call SYSLOG (#CLXS) to issue error message.	
Check for screen format load member existence and check REPLACE value on LOADMBR statement.	$MAFND

Diagram 27.1 (Part 1 of 4). Build Screen Format (SFGR)

DESCRIPTION	MODULE/ ROUTINE
If INOUT verb given, call the find specified library routine ($MAFLB).	$SFGR
Determine if specified library exists.	$MAFLB
If INOUT verb not given, assign default values.	$SFGR
Ⓐ Check for CREATE, ADD, UPDATE, or DELETE verb.	
If CREATE specified with either UPDATE, ADD, or DELETE, call SYSLOG to issue error message.	
Determine if specified source member library name exists.	$MAFND
If specified member does not exist, call SYSLOG (#CLXS) to issue error message.	$SFGR
If DELETE specified, check screen format load member index for format name.	
If format name not in index, call SYSLOG to issue error message.	
Return to **1** Ⓐ and repeat steps until END verb or 32 statements are read.	
2 Move screen format communications area and format load member index to beginning of task area.	$SFDEP or $SFLMM
If only DELETE verb specified, go to **6**.	
3 Load source get transient (#MASYG).	$SFDEP
Assign required work areas needed for input and output field descriptions.	
Ⓐ Search communications table for CREATE, ADD, or UPDATE entry.	#SFDEP
Set #MASYG parameter to do a get next record.	#MASYG
Call #MASYG and check input source specifications for valid values and valid combinations.	
Build table of input and output field descriptions.	$SFDEP
If PRINT-NO not specified; call system list routine (#CLST) to print:	
● Source specification records. ● Input and output record area formats in order required in user's input or output record area. ● Field's location in buffer. ● Length of field.	#CLST
If necessary, call #CLST to print error messages.	$SFDEP
Return to **3** Ⓐ and repeat steps until all source members are read.	
If terminal errors, call end of job transient.	
4 Load sectorized disk data management routine (#DDSM).	$SFFDB
Call special allocate (#CAS1) to allocate scratch work file.	#CAS1
Set up buffers for source get (#SYSG), field descriptor table (FDT), and data stream.	$SFFDB
Open temporary work file.	#OPEN
Ⓐ Search communications table entries for CREATE, ADD, or UPDATE entry.	$SFFDB
Set up #MASYG input parameter to retrieve screen S entry.	#MASYG

Diagram 27.1 (Part 2 of 4). Build Screen Format (SFGR)

DESCRIPTION	MODULE/ ROUTINE
Check values in S entry for YES, NO, and indicator values. As required, place in data stream buffer: ● Clear unit or clear format table order. ● Write command. ● Screen control character. ● Start of header order and data. ● Insert cursor order. ● Repeat to address entity (if repeat to address order required).	$SFFDB
B Retrieve detail field definition D records.	#MASYG
Process D records placing data stream orders and data into data buffer and build applicable FDT entries: ● Convert field length, row, and column position from EBCDIC to hex. ● Add start line number minus one to detail row number. ● Move field length, row/column address, and displacement into data stream and FDT. ● If field is output: — Branch to subroutine to process output fields. — Branch to subroutine to build screen attribute. ● If field is input; branch to subroutine to process input fields: — If field is not also output, branch to subroutine to build screen attribute. — If first input field, move row/column number to the insert cursor order. — Build required field format words based on input specifications. If required, move FDT entry to FDT buffer.	$SFFDB
When full, write FDT buffer to temporary work file.	#DDSM
Go to 4 B and repeat steps until next S record found or end of file reached.	$SFFDB
If record type changed from D to S or last record processed, write balance of FDT entries and data stream to work file.	#DDSM
Go to 4 A and repeat steps until all input source specification members processed.	$SFFDB
Close output file.	#DMCL
If fields specified in input specifications in ascending screen location sequence, go to 6.	$SFFDB
5 Allocate storage space for necessary input and output buffers. A Search communication table for data stream that requires resequencing.	$SFRSQ
Read data stream from work file into input buffer.	Disk IOS
Move input fields from input buffer to output buffer in ascending screen sequence. Remove trailing attributes from output field if required, and move them to the output buffer. Assign pointer values in data stream to permit data stream being placed into user's input buffer in correct sequence.	$SFRSQ
Write resequenced data stream back to work file.	Disk IOS
Return to 5 A and repeat steps until all data streams not in ascending screen order are resequenced and returned to work file.	$SFRSQ
6 Load sector get/put access routine ($MAPGS) If CREATE not specified:	$SFLMM

Diagram 27.1 (Part 3 of 4). Build Screen Format (SFGR)

DESCRIPTION	MODULE/ ROUTINE
A Search SFGR communications table and format index of old format load member for formats that must be copied from old format load member to work file.	$SFLMM
• Read required formats from library.	$MAPGS
• Return.	$SFLMM
• Write formats to work file.	Disk IOS
• Return to 5 A and repeat steps until all necessary formats are copied to work file.	$SFLMM
Resolve address displacements in indexes of formats in work file.	
Open library member.	$MAPGS
Return.	$SFLMM
Read formats from work file.	Disk IOS
Return.	$SFLMM
Write formats to library member.	$MAPGS
Return.	$SFLMM
Print the size of each format in the load member.	#CLST
Return.	$SFLMM
Call end of job transient.	

Diagram 27.1 (Part 4 of 4). Build Screen Format (SFGR)

Program Organization

Figure 27-1 shows the main storage map of the screen format generator utility.

Figure 27-2 shows the control flow for the utility.

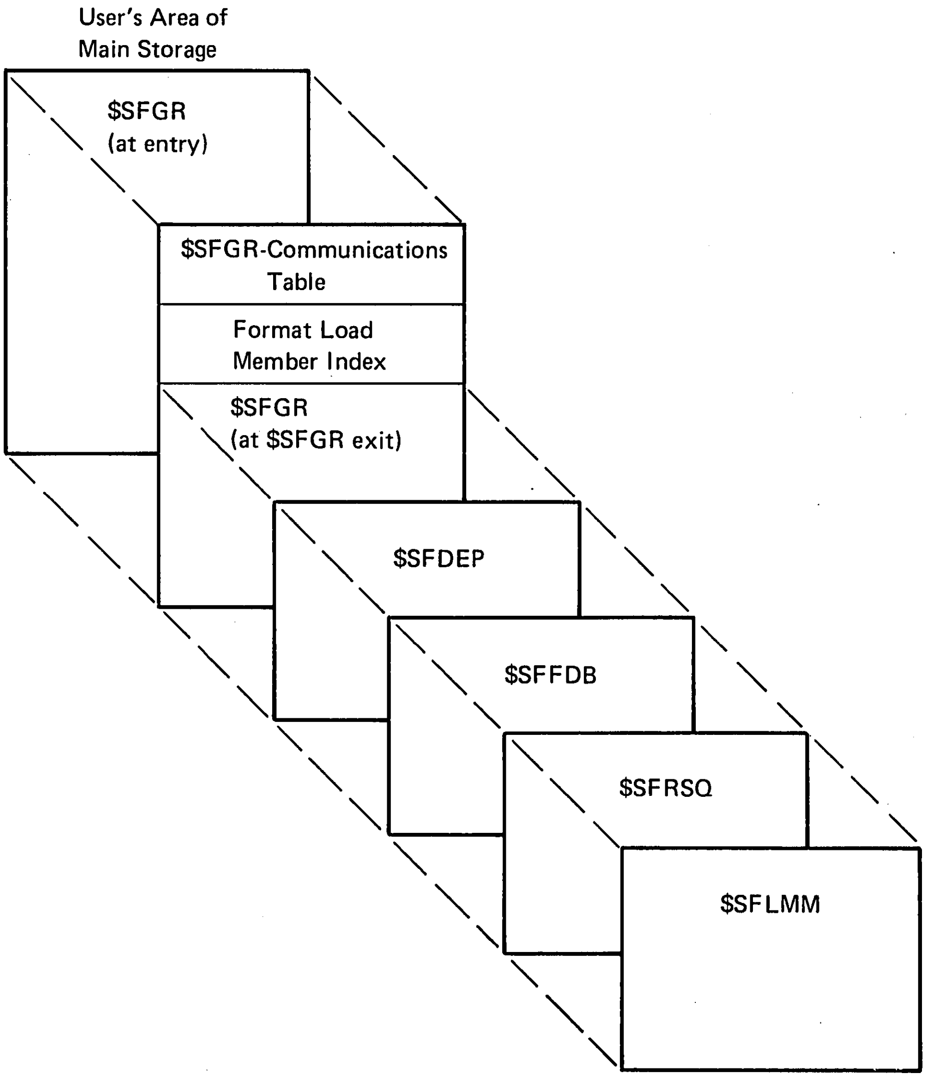


Figure 27-1. Storage Map of Screen Format Generator

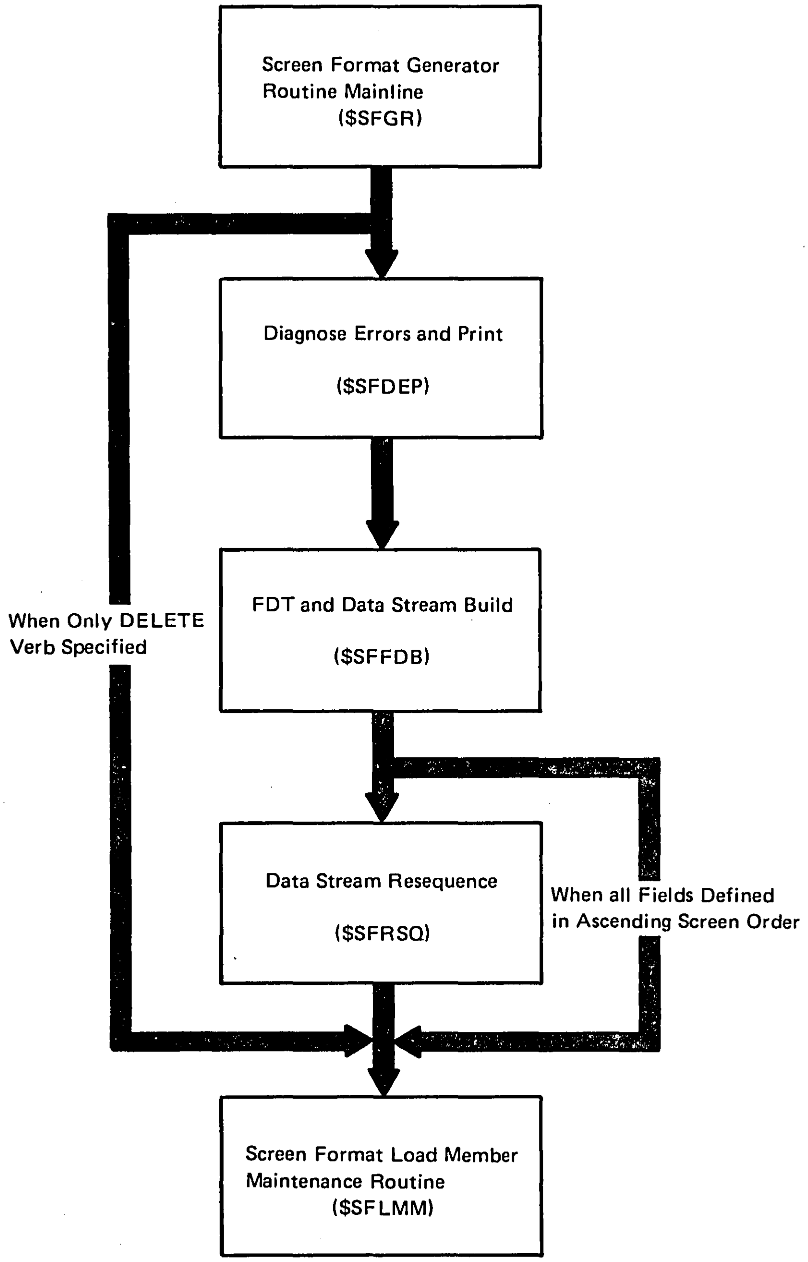

Screen Format Generator Routine Mainline ($SFGR)
Diagnose Errors and Print ($SFDEP)
FDT and Data Stream Build ($SFFDB)
Data Stream Resequence ($SFRSQ)
Screen Format Load Member Maintenance Routine ($SFLMM)
When Only DELETE Verb Specified
When all Fields Defined in Ascending Screen Order

Figure 27-2. Screen Format Generator Control Flow

Chapter 28. Command Processor Procedure Error Utility ($CPPE)

Introduction

This utility provides a means by which SSP and program product procedures can issue error messages. SSP and program product procedures use the #ERR error procedure to call $CPPE.

The $CPPE utility consists of the command processor procedure error utility phase ($CPPE) and syntax checker specification module ($CPTB). This utility resides in the system library.

The #ERR SSP procedure is used to execute $CPPE. It can be used only by SSP and program product procedures:

```
#ERR nnnn,C,alpha code
```

nnnn — Four-digit MIC (message identification code). This parameter must be specified.

C — This parameter can only be C. If omitted, the default is C. It specifies that the job is to be cancelled.

alpha code — Three or four characters specifying the message member to be used. The default is SSP.

Alpha Code	*Message Member*
SSP	##MSG1
SSPU	Active USER1
RPG	Active USER1
AUTO	Active USER1
SEU	Active USER1
SORT	Active USER1
DFU	Active USER1
WSU	Active USER1
ASM	Active USER1
SDA	Active USER1

Note: SSPU is the same as SSP except that the message is retrieved from the active USER1 member.

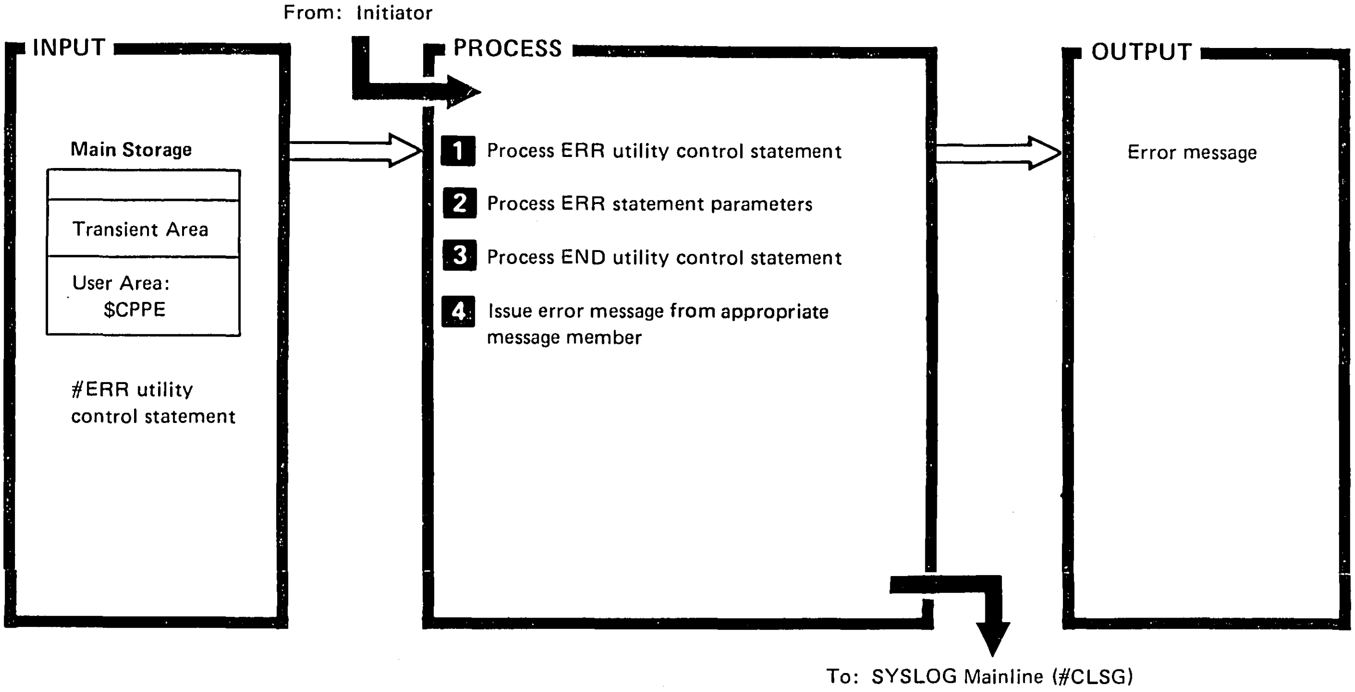
#ERR builds the following OCL statements:

```
// LOAD $CPPE
// RUN
// ERR MIC-nnnn,CONTROL-C,ALPHA-alpha code
// END
```

$CPPE uses the MIC passed in the ERR statement to issue the corresponding message with a 3 option only. The message is retrieved from the message member determined by the ALPHA parameter; the message is issued with the message identifier corresponding to the ALPHA and MIC parameters. The $CPPE utility requires 14K bytes of main storage for execution.

Method of Operation

Diagram 28.1 shows the function of the command processor procedure error utility.


From: Initiator
INPUT
Main Storage
Transient Area
User Area:
$CPPE
#ERR utility control statement
PROCESS
1 Process ERR utility control statement
2 Process ERR statement parameters
3 Process END utility control statement
4 Issue error message from appropriate message member
OUTPUT
Error message
To: SYSLOG Mainline (#CLSG)

DESCRIPTION	MODULE/ ROUTINE
1 Process ERR statement.	#USYX
2 Process parameters.	$CPPE
If errors found in parameters, issue message SYS-5560.	#CLSG
3 Process END statement.	#USYX
If errors found in statement, issue message SYS-5560.	#CLSG
4 Issue error message specified on ERR statement.	#USYX
Issue MIC with 3 option only.	#CLSG
If ALPHA-SSP, retrieve MIC from ##MSG1. Otherwise, retrieve MIC from active USER1 message member. Use ALPHA parameter to determine SYSLOG program ID (SYSLOG uses program ID and MIC for message identifier).	

Diagram 28.1. Command Processor Procedure Error Utility ($CPPE)

Program Organization

Figure 28-1 shows the control flow for the command processor procedure error utility.

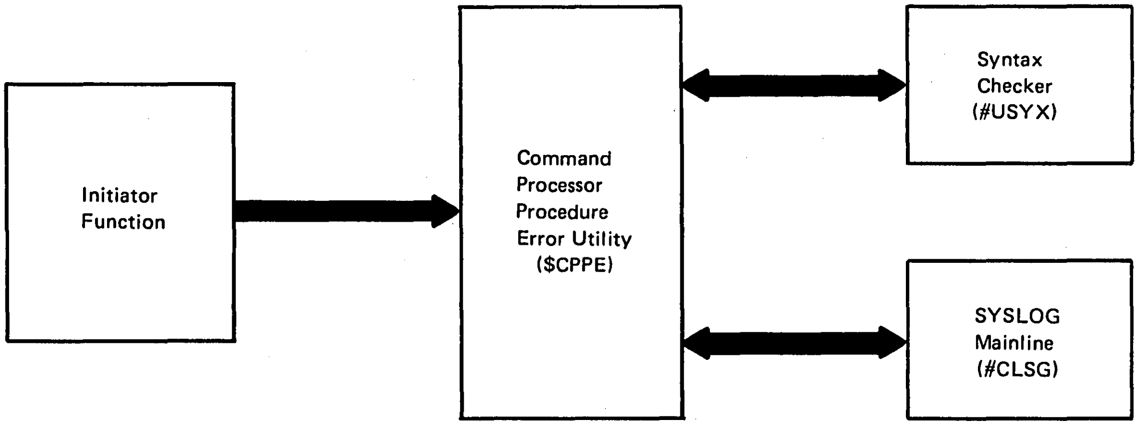


Figure 28-1. Command Processor Procedure Error Utility Control Flow ($CPPE)

Appendixes

Appendix A. Directory

This directory is a reference to the microfiche listings for modules discussed in this manual.

Modules are listed in sequence by module name.

Each directory entry contains the module name, a HIPO reference (if any), the descriptive name, the module's entry point in the listing on microfiche, and a brief description of the module's functions.

Module Name	Diagram	Descriptive Name	Entry Point	Function
$BACK	09.1	Backup library utility	$BACK	Copy the library directory from disk to diskette and update the directory. Create the portion of the backup file on diskette from which IPL can be done. Copy library members from disk to diskette
$BICDI	10.1	Basic interchange, diskette to SYSLIST	$BICDI	Copy files from diskette to the SYSLIST device
$BICFI	10.1	Basic interchange, disk to diskette	$BICFI	Copy files from fixed disk to diskette
$BICIF	10.1	Basic interchange, diskette to disk	$BICIF	Copy files from diskette to fixed disk
$BICR	10.1	Basic interchange control module	$BICR	Read in input parameters, determine which function is to be performed, and load the proper module
$BITAB	–	Syntax specifications for $BICR	None	Input parameter specifications for $BICR
$BMENU	11.1	Build menu utility	$BMENU	Build a menu
$BMTB	–	Build menu specification module	None	Syntax checker specification module used by $BMENU
$BUILD	12.1	Alternate sector rebuild utility	$BUILD	Display the contents of an alternate sector for operator modification
$CNFIG	01.5	System configuration	$CNFIG	Display configuration parameters. Prompt for system configuration parameters. Modify system config record

Module Name	Diagram	Descriptive Name	Entry Point	Function
$COADD	13.1	COPY ADD	$COADD	Add a disk file to an existing diskette file
$COALL	13.2	Copy-all-files preparation routine	BEGIN	Control selection of files from disk/diskette VTOC
$COANT	13.2	Copy-all-files initialization	BEGIN	Control selection of files from disk or diskette; ensure compatibility across diskette boundaries
$COCRT	13.4	Display to the CRT	$COCRT	Interface between $COGET and $FEKEY
$COGET	13.4	Get/put interface with record-mode data management	BEGIN	Interface for record mode data management and/or $COCRT and $COPRT
$COINT	13.3	Single file initialization	BEGIN	Determine the allocation and access methods (including those not resident). Screen mismatched requests
$COPRT	13.4	Display to the printer	$COPRT	Interface between $COGET and SYSLIST
$COPY	13.1	Copy user file utility	$COPY	Mainline module of the $COPY utility. Invoke the syntax checker and route control to (1) $COINT, (2) $COANT, or (3) $COADD according to the function specified
$COSEL	13.4	Record selection and/or deletion	$COSEL	Control selection and/or deletion of records
$COTAB	–	Syntax specifications	None	Define syntax of control statements for $COPY
$COZIP	13.2	Get/put interface with sector mode data management	COZ00Z00	Control movement of data disk-to-disk, disk-to-diskette, diskette-to-disk
$CPPE	28.1	Command processor procedure error utility (#ERR proc)	$CPPE	Issue error messages for #ERR procedure
$CPTB	–	Syntax specifications for #CPPE	None	Define syntax specifications for #ERR procedure statements
$DDST	–	Keysort utility load module	$DDST	Call allocate to have the keys sorted in the index of the specified file

Module Name	Diagram	Descriptive Name	Entry Point	Function
$DELET	14.1	File delete control module	$DELET	Read input parameters, determine which function is to be performed, and load the proper module
$DELF1	14.1	Delete files from fixed disk	$DELF1	Delete files from fixed disk
$DELI1	14.1	Delete files from diskette	$DELI1	Delete files from diskette
$DETAB	–	Syntax specifications for $DELET	$DETAB	Input parameter specifications for $DELET
$DUPRD	15.1	Utility to duplicate diskette	$DUPRD	Copy diskettes from one to another, or one file from one diskette to another
$DUTAB	–	Syntax specifications for $DUPRD	$DUTAB	Input parameter specifications for $DUPRD
$ERAP	07.6	Error recording analysis procedure mainline	$ERAP	Display or print error history information
$ERA0	07.6	ERAP module for disk	ERAPFD	Format information for disk
$ERCA	07.6	ERAP module for work station controller	ERCAX	Format information for work station controller
$ERC0	07.6	ERAP module for work station devices	ERAPWS	Format information for work station devices
$ERD0	07.6	ERAP module for diskette	ERAPIX	Format information for diskette
$ERE0	07.6	ERAP modules for line printers	ERAPLX	Format information for line printers
$ER01	07.6	ERAP module for control store processor	ERAPCS	Format information for control storage processor
$ER02	07.6	ERAP module for main store processor	ERAPMS	Format information for main storage processor
$ER80	07.6	ERAP module for BSC	ER80X	Format information for BSC

Module Name	Diagram	Descriptive Name	Entry Point	Function
$FBLD	16.1	Build null files utility	$FBLD	Build null files with attributes given by user
$FBTAB	—	Syntax specifications for $FBLD	$FBTAB	Input parameter specifications for $FBLD
$FEAPR	07.1	FE APAR prog.	FEAPRMNL	Create an APAR diskette
$FECRT	07.2	FE display program	FECRTMNL	Format and output storage to work station
$FEDMP	07.2	FE dump utility	FEDMPMNL	Process dump utility control cards and pass control to a subroutine to handle specified options
$FEDSK	07.2	FE disk dump routine	FEDSKMNL	Output disk/diskette storage to specified output device
$FEFIX	07.3	FE PTF utility	FEFIXMNL	Apply a fix to system programs
$FEIOP	07.2	FE I/O processor dump routine	FEIOPMNL	Output I/O processor storage to specified output device
$FEKEY	12.1 13.4	FE CRT window display	FEKEYMNL	Display an 80-byte window of a block of data on the CRT screen
$FEPCH	07.4	FE patch program	FEPCHMNL	Process patch utility control cards and pass control to a subroutine to display disk/diskette data
$FESTR	07.2	FE storage dump routine	FESTRMNL	Write main/control storage to specified output device
$FESYM	—	Specification module	None	Defines syntax of control statements for $FEDMP and $FEPCH
$FESYS	07.2	FE system area dump routine	FESYSMNL	Write PTFLOG, configuration record, or disk trace file to specified output device
$FETRC	07.5	FE trace program	FETRCMNL	Set events to be traced by the system and initialize trace file if required
$FREE	23.1	Reorganize disk initialization module	$FREE	Build a table of the start and end SSS address for each file located within the user area of F/D and sort the table in ascending sequence by beginning SSS address

Module Name	Diagram	Descriptive Name	Entry Point	Function
$FRE1	23.1	Accumulate free space high	$FRE1	Each data file on the specified spindle is successively moved to the lowest available block number and all free space is accumulated at the high end of the spindle
$FRE2	23.1	Accumulate free space low	$FRE2	Each data file on the specified spindle is successively moved to the highest available block number and all free space is accumulated at the low end of the spindle
$FRE3	23.1	Rebuild the format 5s	$FRE3	Rebuild the Format 5s to represent the accumulated free space and write them back to disk VTOC
$FRES	–	Syntax specifications for $FREE	None	Input parameter specifications for $FREE
$HIST	17.1	History file display	$HIST	Display contents of history file to CRT/ Printer
$HITB	–	Specification module	None	Define syntax of control statements
$INIT	18.1	Diskette initialization utility	$INIT	Rename diskettes, delete all files, and format diskettes
$LABEL	19.1	Disk VTOC display	–	Build a table containing the start and end SSS address of each file represented in the system and user VTOC on F/D. This table is then sorted in ascending sequence by beginning SS address and a catalog listing of pertinent information is printed via syslist. This module also calls and passes control to $LABI1 if the request was for a display of I1.
$LABI1	19.1	Diskette VTOC display	$LABI1	Part of $LABEL, called by $LABEL. Display VTOC entries from diskette
$LABLS	–	Syntax specifications for $LABEL	None	Input parameter specifications for $LABEL
$LOADI	20.1	Reload library utility	LOADIMNL	Allocate system areas. Copy system library from diskette to disk
$MACMP	21.1	Directory compactor	$MACMP	Remove deleted library directory entries
$MACOM	06.5 06.6	Library open/close	$MACOM	Transient version. Open and close processing for output to a library

Module Name	Diagram	Descriptive Name	Entry Point	Function
$MADLT	21.3	Library member delete	$MADLT	Driver for removing library members
$MADSP	21.4	Library display	$MADSP	Driver to display library status, directory entries, or library members
$MADXP	–	WTG table for $MADSP	None	Include for a WTG table for $MADSP
$MAFIR	21.6	Library directory fast insert	$MAFIR	Put a stack of directory entries into the library directory
$MAFND	06.3	Librarian find	$MAFND	Transient version. Partial name, multiple type, and update find function
$MAF32	21.6	Copy from single program sector mode file to library	$MAF32	Copy single program sector mode file to library
$MAILD	–	Library directory insert	$MAILD	Transient version. Put directory entry in library directory
$MAINT	21.1	Library maintenance mainline	$MAINT	Process utility control statements, set up control bucket, and load and pass control to appropriate driver
$MALCO	21.6 06.5 06.6	Library open/close	$MALCO	Load version. Open and close processing for output to a library
$MALFN	06.3 21.4 21.5	Librarian find	$MALFN	Load version. Find function with capabilities of partial names, multiple types and update
$MALIL	–	Library directory insert	$MALIL	Load version. Add or replace directory entry
$MALOC	21.1	Library allocate	$MALOC	Driver to allocate user library, change library member size, or change user library directory size
$MALTL	21.5	Library to library copy	$MALTL	Driver to copy library members between libraries or within a library; optionally renames members
$MALXC	–	WTG table for $MALOC	None	Include WTG table for $MALOC

Module Name	Diagram	Descriptive Name	Entry Point	Function
$MAPGS	06.6 21.5 21.6	Sector get/put	$MAPGS	Get sector mode modules from a library. Put sector mode modules into a library
$MAPTF	21.3 06.6	PTF handler	$MAPTF	Get, Put, Update and delete entries in PTFLOG
$MAPUR	06.5 21.6	Record put	$MAPUR	Put source or procedure members into a library in compressed format
$MARCK	21.2	Library compress	$MARCK	Driver to compress library member space
$MARDR	21.6	Reader to library copy	$MARDR	Driver to copy S or P from the reader (system input device) to a library
$MARFF	21.6	Copy from file to library in record mode	$MARFF	Copy source or procedure members in record mode from a disk or diskette file into a library
$MARPF	21.4	Record mode file to display	$MARPF	Driver to display type and name of library members in a record mode file
$MARTF	21.5	Copy from library to file in record mode	$MARTF	Copy record mode members from a library to either a disk or diskette file. Copy control statement is put at the beginning of member and CEND record is put at the end
$MARXF	–	Where-to-go table	None.	This module is included by $MARTF
$MAR2K	21.2	Library compress	$MAR2K	Second phase of library compress – update library member disk addresses kept in control blocks
$MAR3K	21.2	Library compress	$MAR3K	Suspend command processor when #LIBRARY is being compressed
$MASDF	21.4	Sector mode file to display	$MASDF	Driver to display type and name of library members in sector mode librarian file (S/32 or S/34)
$MASPC	–	Specification module	None	Define utility control statements for $MAINT
$MATFS	21.5	Copy from library to file in sector mode	$MATFS	Copy sector mode members from a library to either a disk or diskette file
$MATLS	21.6	Copy from file to library in sector mode	$MATLS	Copy librarian sector mode members from a disk or diskette file into a library

Module Name	Diagram	Descriptive Name	Entry Point	Function
$MAXNT	–	Where-to-go table of system transient table entries	None	This module is loaded by $MATLS and is a where-to-go table of all members in the system transient table
$MGBLD	22.1	Message build utility	$MGBLD	Create load message members
$MGTAB	–	Syntax specifications for $MGBLD	$MGTAB	Input parameter specifications for $MGBLD
$PACK	23.1	Reorganize disk initialization module	PAK000	Build a table of the start and end SSS address for each file located within the user area of disk and sorts the table in ascending sequence by beginning SSS address
$PROF	24.1	Security file utility	$PROF	Maintain system security file
$PRST	24.2	Security file restore utility	$PRST	Restore security file from diskette
$PRSV	24.3	Security file save utility	$PRSV	Save security file on diskette
$RENAM	25.1	Rename utility	$RENAM	Rename user files
$RETAB	–	Syntax specifications for $RENAME	None	Input parameter specifications for $RENAME
$SETCF	26.1	Set configuration utility	$SETCF	Modify and copy to the associated work station configuration record updated session values: that is, date, region site, library, communications configuration values, etc.
$SETSM	–	Set configuration utility syntax specification	$SETSM	Describe via tables all allowable verbs and keywords for set configuration utility
$SFDEP	27.1	Screen format diagnose and print routine	$SFDEP	Print and diagnose SFGR source input specifications. Print input and execution output buffer configurations. Print size of each format
$SFDXP	–	Where-to-go table for $SFDEP	SFFDB DEPXEN	Link edited with $MASYG
$SFFDB	27.1	Screen format FDT and data stream build routine	$SFFDB	Build the data stream and FDT entries (if required) for each format
$SFGR	27.1	Screen format generation routine mainline	$SFCT	Read utility control statements and build SFGR communications table. Diagnose utility control statement errors

Module Name	Diagram	Descriptive Name	Entry Point	Function
$SFLMM	27.1	Screen format load member maintenance routine	$SFLMM	Add, replace, or delete screen formats in a load member; or create or replace an existing load member. Uses $MAPGS to copy the load member data to the library
$SFRSQ	27.1	Screen format resequence routine	$SFRSQ	Sort the data stream input fields into ascending screen sequence. Truncate trailing screen attributes of output fields if necessary. Update FDT pointer
$SFSPC	–	Screen format specification routine	$SFSPC	Specification module for $SFGR and syntax checker
##FCFG	–	Second half of command processor displays	–	Display member
##FCPF	–	First half of command processor displays	–	Display member
##FER1	–	Display screens for ERAP utility	–	Display member
##FFE1	–	Display screens for DUMP/PATCH utility	–	Display member
##FFE2	–	Display screens for TRACE utility	–	Display member
##FFE3	–	Display to allow disk/diskette storage to be modified	–	Display member
##FIPL	–	Display for IPL overrides	–	Display member
##FLOD	–	Display for RELOAD prompts	–	Display member
##FMC1	–	Screen formats for concurrent maintenance	–	Display member
##FPRS	–	Display screens for $PROF security utility	–	Display member

Module Name	Diagram	Descriptive Name	Entry Point	Function
##FST1	–	Menus to allow CE to run I/O during installation tests	–	Display member
##FST2	–	Menus to allow CE to run I/O during installation tests	–	Display member
##FST3	–	Menus to allow CE to run I/O during installation tests	–	Display member
##FST4	–	Menus to allow CE to run I/O during installation tests	–	Display member
##FST5	–	Menus to allow CE to run system test to verify installation	–	Display member
#CACM	03.3	Communicator allocate	#CACM	Attach the data management task necessary for a communications DTF
#CAD1	03.5	Deallocate premainline	#CAD1	Check user's DTF for a valid request, check for keysorting of files, and deallocate devices
#CAD2	03.5	Deallocate mainline	#CAD2	Deallocate disk files, delete disk files, and free unused space in disk files
#CAD3	03.5	Update the VTOC F5 for deallocate	#CAD3	Update the disk VTOC F5 from the info in the DTFs passed by #CAD2
#CAF1	03.3	New file setup	#CAF1	Set up the DTF with the information necessary for #CAF2 to get space. Reads the F5 for #CAF2
#CAF2	03.3	Get space in F5 area	#CAF2	Find space in the F5 area for the new files in the DTF chain
#CAF3	03.3	Format file, update VTOC	#CAF3	Format the file and update to VTOC F1 for new and load to old files
#CAKS	03.3	Allocate keysort	#CAKS	Perform keysorting of files during allocate and deallocate processing
#CAMG	03.3	Allocate messages	#CAMG	Set up SYSLOG parameter list for allocate messages
#CAML	03.3	Allocate mainline	#CAML	Allocate old disk files and devices; determine if further processing to be done

Module Name	Diagram	Descriptive Name	Entry Point	Function
#CAPS	03.3	Push/pull for diskette allocate and keysort	#CAPS	Make available user storage for the modules #CAR1 and #CAKS
#CAPT	03.3	Printer allocate	#CAPT	Allocate printers
#CAR1	03.3	Diskette allocate	#CAR1	Do diskette file allocate processing
#CAS1	03.4	Special allocate	#CAS1	Build and queue an FSB and F1 for each special allocate DTF
#CAS2	03.4	Special old disk allocate	#CAS2	Do old disk file allocate for special DTFs. Determine further processing to be done
#CCAS	02.4	Assign and vary	#CCAS	Process assign and vary commands
#CCCM	02.5	CANCEL command	#CCCM	Process CANCEL command and INQUIRY CANCEL
#CCCO	02.7	CONSOLE command	#CCCO	Reassign system console when an I/O error occurs at the console
#CCCP	02.5	Spool command	#CCCP	Spool command — CANCEL
#CCGP	02.6	Spool command	#CCGP	Spool command — CHANGE
#CCHO	02.8 02.6	Spool command	#CCHO	Spool commands — HOLD, RELEASE
#CCID	02.9	IDELETE command	#CCID	Process the information message delete command
#CCJQ	02.5 02.6	JOBQ command (input)	#CCJQ	Process JOBQ, CANCEL, CHANGE and STOP JOBQ commands
#CCJS	02.15	JOBQ command (input)	#CCJS	Process START JOBQ command
#CCMG	02.11	MSG command	#CCMG	Message command diagnostic and disk queuing
#CCMU	02.10	MENU command	#GCMU	Process MENU command
#CCMX	02.11	MSG command	#CCMX	Display messages at a work station
#CCOF	02.12	OFF command	#CCOF	Process OFF and MODE command
#CCPY	02.13	PRTY command	#CCPY	Change and/or set a job's priority
#CCRE	02.14	REPLY command	#CCRE	Reply command to system console messages

Module Name	Diagram	Descriptive Name	Entry Point	Function
#CCRR	02.15	Spool command	#CCRR	Spool commands – START, STOP, RESTART
#CCRS	02.14	REPLY command	#CCRS	Handle second level messages
#CCRT	02.15	STOP/START commands	#CCRT	Stop/start functions
#CCSJ	02.16	JOBQ command (status)	#CCSJ	Process status JOBQ command
#CCSM	02.16	STATUS mainline	#CCSM	Route control to proper status module
#CCSP	02.16	STATUS PRT	#CCSP	Process status PRT command
#CCSS	02.16	STATUS session-1	#CCSS	Process session status page 1
#CCSU	02.16	STATUS users	#CCSU	Process users status
#CCSW	02.16	STATUS workstn	#CCSW	Process work station status
#CCS2	02.16	STATUS session-2	#CCS2	Process session status page 2
#CCS3	02.16	STATUS session-3	#CCS3	Process session status page 3
#CCS4	02.16	STATUS session-4	#CCS4	Process session status page 4
#CCTD	02.17	TIME command	#CCTD	Return time-of-day and system date
#CCU2	02.16	Command processor status user's routine	#CCU2	Display active user information on display screen
#CIAT	03.1	ATTR statement processor	IAT00100	Process an ATTR statement
#CICM	03.1	COMPILE statement processor	ICM00100	Process a COMPILE statement
#CICO	03.1	COMM statement processor	ICO00100	Process a COMM statement
#CICX	–	Initiator (#CICM) OXREF table	MAFL@ CIER@	Cross reference table
#CIDT	03.1	DATE statement processor	IDT00100	Process a DATE statement
#CIER	03.1	Initiator error routine	IER00100	Issue errors detected by initiator routines
#CIFM	03.1	FORMS statement processor	IFM00010	Process a FORMS statement

Module Name	Diagram	Descriptive Name	Entry Point	Function
#CIIC	03.1	INCLUDE statement processor	IIC03900	Process an INCLUDE statement
#CIIM	03.1	IMAGE statement processor	IIM00100	Process an IMAGE statement
#CIIX	–	Initiator (#CIIC) OXREF table	CIMT@ CIER@	Cross reference table
#CILB	03.1	LIBRARY statement processor	ILB00100	Process a LIBRARY statement
#CILC	03.1	LOCAL statement processor	ILC00100	Process a LOCAL statement
#CILD	03.1	LOAD statement processor	ILD00001	Process a LOAD statement
#CILG	03.1	LOG statement processor	OLG00100	Process a LOG statement
#CILM	03.1	Load MRT routine	ILM00100	Process LOAD statement for a MRT program
#CILX	–	Initiator (#CILD) OXREF table	CILM@ CIER@	Cross reference table
#CIML	03.1	Initiator mainline	IML00010	Read OCL statements; pass control to STMT processors; perform file processing
#CIMM	03.1	MEMBER statement processor	IMM00010	Process a MEMBER statement
#CIMS	03.1	* statement processor	IMS00100	Process an * statement
#CIMT	03.1	MRT procedure	IMT03900	Set up a MRT for initial call and attach to existing MRT
#CIM2	03.1	** statement processor	IM200100	Process a ** statement
#CIPR	03.1	PRINTER statement processor	IPR00010	Process PRINTER statement
#CIPS	03.1	PHASE statement processor	IPS00100	Process a PHASE statement
#CIRG	03.1	REGION statement processor	IRG00010	Process a REGION statement
#CIRN	03.1	RUN statement processor	IRN00100	Final initiator phase; load user program

Module Name	Diagram	Descriptive Name	Entry Point	Function
#CIRX	–	Initiator (#CIRN) OXREF table	SVAU@ CIER@	Cross reference table
#CISL	03.1	SYSLIST statement processor	ISL00100	Process a SYSLIST statement
#CISW	03.1	SWITCH statement processor	ISW00010	Process a SWITCH statement
#CIVT	03.1	VTOC read/write interface	IVT08500	Interface with VTOC read/write routine
#CIVX	–	Initiator (#CIVT, #CIMM, and #CIM2) OXREF table	CIER@	Cross reference table
#CIWK	03.1	WORKSTN statement processor	IWK00100	Process a WORKSTN statement
#CLAC	06.15	Active existence test	#CLAC	IF active existence test
#CLBL	06.15	Blocks existence test	#CLBL	IF blocks existence test
#CLFX	06.15	File existence test	#CLFX	IF file existence test
#CLOX	–	OXRF table	None	OXRF table for initiator and SYSIN
#CLPR	06.15	Prompt routine	#CLPR	Prompt and substitute parameters
#CLSB	06.15	Substitution routine	#CLSB	Perform nonparameter substitution
#CLSC	06.16	SYSLIST CRT (transient)	–	Display system programs output or record (for those programs using SYSLIST) when the SYSLIST device is the CRT
#CLSG	06.17	SYSLOG mainline	–	Display messages to a work station or system console. Print these messages if the printer is the SYSLOG device and the system is in single program mode. Also record the message in the history file
#CLSM	06.15	Switch/member existence tests	#CLSM	Switch/member existence tests
#CLSN	06.15	SYSIN (transient)	#CLSN	Push/pull for SYSIN mainline
#CLSP	06.16	SYSLIST printer loadable module	#CLSP	If printer is SYSLIST device, display SYSLIST records
#CLSS	06.15	SYSIN mainline	#CLSS	Procedure handling mainline

Module Name	Diagram	Descriptive Name	Entry Point	Function
#CLST	06.16	SYSLIST printer transient	–	Print system programs output or record (system programs that use SYSLIST) when the SYSLIST device is the printer
#CLSW	06.16	SYSLIST CRT loadable module	#CLSW	If display screen is SYSLIST device, display SYSLIST records
#CLXS	06.17	SYSLOG push/pull (transient)	#CLXS	Push/pull main storage to load the SYSLOG mainline (#CLSG)
#CMCI	02.28	Console management	#CMCI	Accept messages from a user and queue them to the system console
#CMCS	02.28	Console management	#CMCS	Display messages on the system console queue
#CMCU	02.31	CP clean up	#CMCU	Command processor routine which does log printer, history file put, msg retrieve and message displaying
#CMEJ	02.28	Console EOJ	#CMEJ	Marks all messages responded to by EOJ with '**' on console
#CMLS	02.29	SYSLIST CRT	#CMLS	The data management for SYSLIST to a work station
#CMWI	02.29	Work station SYSIN data management	#CMWI	The data management for retrieving SYSIN records from a work station
#CMWO	02.29	SYSLOG/SYSLIST data management	#CMWO	The data management for SYSLOG to a work station
#CPER	02.31	Console management error recovery	#CPER	Process I/O errors at work stations and printers under the command processor
#CPIO	02.26 02.21 02.23	Command processor input/output processor	#CPIO	Create WSDM parameter list and call WSDM
#CPIQ	02.23 02.24	CP inquiry menu processor	#CPIQ	Process inquiry options, rename and other exception conditions
#CPKS	02.15	KEYSORT all files	#CPKS	Keysort any files in the VTOC that have the sort or merge bit on
#CPML	02.1	Command processor	#CPML	Wait for events to occur
#CPON	02.2 01.4	Sign-on	#CPON	Sign-on module
#CPRT	02.1 02.19	CP input processor/ router	#CPRT	Initially process input routine commands to appropriate transient

Module Name	Diagram	Descriptive Name	Entry Point	Function
#CPSP	02.25	Special key processor	#CPSP	Initiate processing required by special function keys on keyboard
#CPTC	02.1 02.20 02.21 02.30	CP task post processor	#CPTC	Process sys req, inquiry, and release functions
#CPTS	02.30	Sign-on display writer	#CPTS	Write sign-on display
#CSAF	06.8	AFA access	#CSAF	Read the AFA format 1's into a callers I/O buffer
#CSDK	06.10	Duplicate key halt routine	#CSDK	Issue SYSLOG halts if a duplicate key is found during keysort
#CSIM	06.11	Printer set-up halt routine	#CSIM	Issue SYSLOG halts if printer setup required. Image, forms, spool separator pages
#CSVF	06.12	Disk VTOC read/write	#CSVF	Disk VTOC read/write transient version
#CSVI	06.13	Diskette VTOC read/ write main (transient)	#SCVI	Read, write diskette VTOC and prepare function
#CSVJ	06.13	Diskette VTOC read/ write main (transient)	#CSVJ	Build the diskette VTOC work area on disk
#CSVK	06.13	Diskette VTOC read/ write (transient)	#CSVK	Build the diskette VTOC on diskette from the diskette VTOC work area on disk
#CTECM	05.2	Termination communications interface	#CTECM	Process communication specification block (CSB) for a task
#CTEEX	–	Termination exit interface	#CTEEX	Set or modify the termination exit address for communications program
#CTEGU	05.2	Termination interface to get the user into main storage	#CTEGU	Pull user program back into main storage from disk
#CTEIF	05.2	Termination interface	#CTEIF	Get control initially when terminating a step/job. Provide the interface to the main terminator
#CTEKS	05.2	Termination keysort interface	#CTEKS	Pass control to keysort if required
#CTEPR	05.2	Termination processor	#CTEPR	Perform most of the termination functions necessary to terminate a step/job (keysort, files processing, work station logic, etc.)

Module Name	Diagram	Descriptive Name	Entry Point	Function
#CTES	05.2	Spool-JOBQ termination	#CTES	Update spool file and JOBQ file after compress has run
#DDCL	05.1	Disk close (transient)	#DDCL	Issue completion of all I/O operations, return DTF to pre-open status, update format 1 to current status of the file
#DDDM	04.1	Nucleus resident disk data management	X'0000'	Provide general disk data management support
#DDKAA	04.7	Keysort-control	X'0000'	Keysort control
#DDKEP	04.7	Keysort-end of phase	X'0000'	Perform phase to phase communications
#DDKLL	04.7	Keysort-assignment and 3-phase sort control	X'0000'	Design sort, allocate workfile, examine format 1
#DDKSS	04.7	Keysort-sort in place	X'0000'	Sort without a work file
#DDK1A	04.7	Keysort-3-phase internal sort	X'0000'	Create initial strings on work file
#DDK2A	04.7	Keysort-3-phase intermediate merge	X'0000'	Merge intermediate strings from and to work file
#DDK3A	04.7	Keysort-3-phase final merge	X'0000'	Merge all intermediate strings back to index
#DDLM	–	Disk limits open	#DDLM	Establish user limits for index sequential input or update files
#DDSM	04.2	Disk sector data management	#DDSM	Handle gets and puts in sector mode to disk
#DDSR	–	Index sequential/random input to disk	#DDSR	Handle random, forward, or backward gets from the disk
#DD1OP	03.6	Disk open transient one	#DD1OP	Perform diagnostic checking of fixed disk DTFs. Also initialize all DTF fields for dummy and ZPAM open
#DD2OP	03.6	Disk open transient two	#DD2OP	Format the DTF, IOBs and data areas (of fixed disk DTF's) as required
#DMCL	05.1	Common data management close transient	#DMCL	Serve as a routing routine for DTFs that are to be closed. Call appropriate device-dependent close modules to perform re-initialization of DTF's
#DMOF	03.6	Second common data management open transient	#DMOF	Act as routing for preopen DTFs. Pass control to appropriate device dependent open modules

Module Name	Diagram	Descriptive Name	Entry Point	Function
#DMOP	03.6	Common data management open transient	#DMOP	Act as routing for preopen DTFs. Pass control to appropriate device dependent open modules
#DPAL	04.5	Printer alignment (transient)	#DPAL	Give user capability to align printer forms
#DPCL	05.1	Printer close (transient)	#DPCL	Ensure completion of all issued I/O events to printer and restore printer DTF to pre-open state
#DPDM	04.5	Printer data management	#DPDM	Issue IOBs to printer IOS to perform requested I/O operations
#DPOP	03.6	Printer open (transient)	#DPOP	Prepare printer and printer control blocks to handle I/O requests
#DRCL	05.1	Diskette close	#DRCL	Close diskette files
#DRDM	04.3 04.4	Diskette record data management	#DRDM	Handle gets and puts in record mode to diskette
#DRNV	04.4	Diskette end of volume	#DRNV	Handle volume transition of diskette files
#DROP	03.6	Diskette open	#DROP	Open diskette files
#DRSM	04.3 04.4	Diskette sector data management	#DRSM	Handle gets and puts in sector mode to diskette
#DWDM	04.6	Work station data management router	#DWDM	Call work station data management routines to perform management functions (link-edited with #DDDM)
#HFPUT	06.18	History file put (transient)	#HFPUT	Place entries into history file along with user ID < terminal ID, job name, time-stamp, etc.
#MAFLB	06.1	Find a library	#MAFLB	Find and set up for use of a library by name
#MAMPM	06.7	Member protection module	#MAMPM	Prevent SEU tasks from updating a library member if that member is currently being updated by another SEU task. Prevent initiation of an SEU task when an SEU user has made an inquiry back into SEU. Create a new element on the SEU member chain
#MANOP	06.9	No-op module	NOP0000 NOP0003 NOP0007	Match entry points of modules called via WTG tables. This module is put in a WTG table when the desired module is missing. Issue a halt and cancel job when given control

Module Name	Diagram	Descriptive Name	Entry Point	Function
#MASFN	06.2	System find	#MASFN	Single name find routine transient version
#MASGT	06.4	Source get	#MASGT	Transient — get and expand source or procedure records
#MASYL	06.4	Source get	#MASYL	Include and load version expands S or P records
#MAXRF	06.9	Cross reference resolver	XRF0000 XRF0003	Call EOJ when done. Return to caller when done. Fill in WTG and format index tables
#MGRET	06.14	Message retrieve (transient)	#MGRET	Retrieve load message members created by $MGBLD
#MSBFL	01.6	Rebuild VTOC format 1's — phase 2	#MSBFL	Run keysort
#MSBGL	01.6	Rebuild VTOC format 1's — phase 3	#MSBGL	Check F1's for proper latest date indicator
#MSBLD	01.6	Rebuild VTOC format 1's — phase 1	#MSBLD	Rebuild VTOC F1's if requested by system operator
#MSIPL	01.4	IPL — 3rd phase	MSIPLMNT	Initiate SIGN-ON. Process CONFIG. options
#MSJQ	01.4	JOBQ IPL (input)	#MSJQ	Format disk and main store for input JOBQ
#MSNIP	01.2	IPL — 1st phase	MSNIPMNT	Initiate main storage IPL
#MSOER	01.4	IPL — override transient 2	MSOERMNT	Prompt for IPL overrides for JOBQ and spool
#MSRID	01.4	IPL — override transient 1	MSRIDMNT	Prompt for system configuration override options
#MSSP	01.4	Spool IPL	#MSSP	Format disk and main store for spool
#MSSQS	01.4	IPL — initialize storage	MSSQSMNT	Initialize user storage and assign/free area
#MSTWA	01.3	IPL — 2nd phase	MSTWAMNT	Load resident routines. Initialize TWA
#OLAF	08.4	Autolink segment list build	AFA000	Read $WORK object modules and build autolink segment list entries on $SOURCE
#OLAH	08.5	Cross-reference segment list build	OLAH00	Read records from autolink segment list on $SOURCE and build the cross-reference segment list
#OLAJ	08.6	Sort autolink segment list	AJA000	Sort autolink segment list into sublists

Module Name	Diagram	Descriptive Name	Entry Point	Function
#OLAP	08.7	Overlay design	OLAP00	Calculate storage requirements and determine overlay structures
#OLAR	08.8	Overlay segment list build	OLAR00	Build an overlay segment list and writes object code to $WORK
#OLAT	08.9	Storage map phase	AT010	Print storage map and error messages
#OLBE	08.10	Relocate, resolve EXTRNs, and build load module phase	START	Relocate object modules within overlays, resolve external references, and combine relocated object modules into load modules
#OLBO	08.11	Library control phase	OLBO00	Catalog object and load modules in the library directory
#OLER	08.12	Error routine	ER000	Call SYSLOG to issue messages
#OLINK	08.14	User entry phase 1	INK000	Open work files, initialize the common area (LOMMON), and read control statements
#OLISP	–	OLE syntax specification module	None	Define the user control statement verbs, keywords, and parameters
#OLI1	08.15	User entry phase 2	INK105	Check control statement parameters and transfer data to common area (LOMMON)
#OLI2	08.16	User entry phase 3	INK105	Check control statement parameters and copy object modules to $WORK
#OLI3	08.17	User entry phase 4	INK000	Check control statement parameters and build a segment list entry on $SOURCE
#OLMSG	08.13	Error message print phase	OLMSG0	Print error messages
#OLYNX	08.3	Compiler entry phase	YNX000	Initialize common area (LOMMON), move OPTIONS data and name ESL to LOMMON
#PRSD	02.32	Password security	#PRSD	Retrieve user's security file and make sign-on security check
#PTFLOG	–	PTFLOG load module	None	Contain PTF log entries for library in which it resides
#SA@CU	–	Display screens to create, add, update, total formats	–	Display member
#SA@DS	–	Display screens to delete entire formats or to manipulate SFGR statements directly	–	Display member

Module Name	Diagram	Descriptive Name	Entry Point	Function
#SA@DP	–	Display screens to display formats in an SFGR load module	–	Display member
#SA@HP	–	Display screen for help text	–	Display member
#SA@ME	–	Display screens to create or update menu source statements	–	Display member
#SA@RC	–	Display screens to build a skeleton RPG display station program	–	Display member
#SA@RP	–	Display screens for user recovery	–	Display member
#SPALC	04.8	Spool allocate	#SPALC	Allocate spool control blocks and disk extents
#SPCLO	04.8	Spool close	#SPCLO	Close the spool file entry
#SPDPQ	04.8	Spool data and print queue	#SPDPQ	Create the data entries and add the index to the print queue
#SPINT	04.8	Spool intercept	#SPINT	Intercept the print line
#SPQMG	04.9	Queue manager	#SPQMG	Get entries from spool file
#SPWRT	04.9	Spool writer	#SPWRT	Print entries from the spool file
#SVAT	06.19	New task attach	#SVAT	Allocate necessary resources to attach a new task to the system
#SVAU	06.20	Change task's ORG point or detach task	#SVAU	Perform necessary ATR manipulation whenever a task's ORG point changes, or deallocate resources and detach task from system
#SVDMP	06.24	Snap dump (transient)	SVDMPANT	Nonterminating main storage dump
#SVERJ	05.2	EOJ purge of console queue for I/O message	#SVERJ	Check console SYSLOG queue and console matrix for messages associated with the task that is being cancelled
#SVERP	02.33	Error processing (transient)	#SVERP	Allows I/O devices to issue error messages to the system console. Routes control for I/O error recovery

Module Name	Diagram	Descriptive Name	Entry Point	Function
#SVINF	06.22	Information retrieval (transient)	#SVINF	Retrieve required information for non-privileged programs requiring access to fields in privileged JCBs
#SVNRY	02.35	Display station error recovery for device not ready	#SVNRY	Attempt recovery from operator error mode and system request mode command reject exceptions
#SVPRE	02.33	Printer prepare	SVPREMNT	Set printer image, forms length and width, and graphic error procedure to be taken
#SVRD	02.35	Command reject ready routine	#SVRD	Retrieve command rejected records from a work station command reject file
#SVTTC	06.32	Data management task transfer control	SVTNTY	Control communication, between user programs and system data management tasks
#SVUR	02.35	Command reject ready transient	#SVUR	Push out user task and load #SVRD
#SVWER	02.33	Work station error message transient	#SVWER	Allow work stations to issue error messages to system console
#SVWSR	02.34	Display station error recovery	#SVWSR	Attempt recovery from: hardware malfunctions, software errors, command reject exceptions, and process not ready to ready ready-post
#USYX	06.21	Syntax checker	#USYX	Verify the syntax for control statements as defined by the specification module
#WDAF	04.6	Assign failure (transient)	#WDAF	Processed WSQS assign failure for work station data management
#WDDA	04.6	Work station data management	#WDDA	Resident version of work station data management
#WDDB	04.6	Work station data management	#WDDB	Transient version of work station data management
#WDDC	04.6	Work station data management	#WDDC	Transient that handles miscellaneous work station data management functions
#WDDG	04.6	Work station data management GET routine	#WDDG	Handle status inquiry, acquire terminal, get terminal attributes, invite terminal input, accept terminal input, get terminal input, and release terminal from program functions

Module Name	Diagram	Descriptive Name	Entry Point	Function
#WDDH	04.6	Work station data management transient routines	#WDDH	Handle process requests for help and print keys, key masking functions, and issue message for print key request functions
#WDDK	04.6	Work station data management transient routines	#WDDK	Process requests for the print key
#WDDO	04.6	Work station data management PUT override routine	#WDDO	Handle put override operation
#WDDQ	04.6	Work station data management GET routine	#WDDQ	Handle acquire terminal and get terminal attribute functions
#WDOPN	03.6	Work station open transient	#WDOPN	Open work station DTFs for the user
@CSVF	06.12	Disk VTOC read/write	@CSVF	Disk VTOC read/write (link-edit version)
@HFPTR	06.18	History file put (read main storage include)	@HFPTR	Place entries into history file along with user ID, work station ID, job name, timestamp, etc.
@HFPUT	06.18	History file put (transient include)	@HFPUT	Place entries into history file along with user ID, work station ID, job name, timestamp, etc.
@MASFN	06.2	System find	@MASFN	Include version of the single-name-find routine
@MGRET	06.14	Message retrieve load include	@MGRET	Object include version of message retrieve, to be included at compile time
@MASYL	22.1	Get source member	@MASYL	Get source member from library and process

Appendix B. List of Acronyms and Abbreviations

@	Address
ACE	Action control element
AFA	Active format 1 area
APAR	Authorized program analysis report
AQE	Allocation queue element
ARR	Address recall register
ASCII	American national standard code for information interchange
ATR	Address translation register
BSC	Binary synchronous communications
CAM	Compiler access method
CIB	Compiler information block
CP	Command processor
CRT	Cathode ray tube (display screen)
CS	Control storage
CSB	Communication specification block
DTF	Define the file
EBCDIC	Extended binary coded decimal interchange code
ENQ	Enquiry
EOJ	End of job
ERAP	Error recording analysis procedure
ERB	Error recovery block
ERP	Error recovery procedure
ESL	External symbol list table
FDT	Field descriptor table
FQE	Free queue element
FSB	File specification block
HDR	Header
I/O	Input/output
IAR	Instruction address register
IMPL	Initial microprogram load
IOB	Input/output block
IOCH	Input/output control handler
IOS	Input/output supervisor
IPL	Initial program load
IWA	Initiator work area
JCB	Job control block
LCB	Library control block
LCS	Library control sector
LSR	Local storage register
MEB	Member enqueue block
MIC	Message identification code
MRT	Multiple requesting terminal
MS	Main storage
MVF	Multivolume file
NEP	Never ending program
OCL	Operation control language
OXRF	Cross-reference table

PIQ	Place in queue
PMR	Program mode register
PPSA	Procedure parameter save area
PSB	Printer specification block
PSR	Program status register
PTF	Program temporary fix
QFD	Queue file description
RB	Request block
RIB	Request indicator byte
RIWA	Reader interpreter work area
RLD	Relocation dictionary
RRN	Relative record number
SCA	System communication area
SFD	Spool file description
SFGR	Screen format generator routine
SIO	Start input/output
SQB	Sector queue block
SQE	Status queue element
SRT	Single requesting terminal
SS	Sector address
SSP	System support program
SVC	Supervisor call
TCB	Task control block
TQE	Timer queue element
TUB	Terminal unit block
TWA	Task work area
UPSI	User program status indicator
VTOC	Volume table of contents
WSB	Work station specification block
WSDM	Work station data management
WSWA	Work station work area
WTG	Where-to-go (table)
XR1	Index register one
XR2	Index register two
ZPAM	Sector data management

Index

IBM System/34
System Support Program
Logic Manual: System

LY21-0050-0

Please use this form only to identify publication errors or request changes to publications. Technical questions about IBM systems, changes in IBM programming support, requests for additional publications, etc, should be directed to your IBM representative or to the IBM branch office nearest your location.

Error in publication (typographical, illustration, and so on). **No reply.**

Page Number *Error*

Inaccurate or misleading information in this publication. Please tell us about it by using this postage-paid form. We will correct or clarify the publication, or tell you why a change is not being made, provided you include your name and address.

Page Number *Comment*

Note: All comments and suggestions become the property of IBM.

● No postage necessary if mailed in the U.S.A.

Name ______________________________

Address ______________________________

____

— Cut Along Line —

Fold

Fold

FIRST CLASS
PERMIT NO. 40
ARMONK, N. Y.

IBM Çorporation
General Systems Division
Development Laboratory
Publications, Dept. 245
Rochester, Minnesota 55901

Fold

Fold

IBM®

International Business Machines Corporation

General Systems Division
4111 Northside Parkway N.W.
P.O. Box 2150
Atlanta, Georgia 30301
(U.S.A. only)

General Business Group/International
44 South Broadway
White Plains, New York 10601
U.S.A.
(International)

International Business Machines Corporation

General Systems Division
4111 Northside Parkway N.W.
P.O. Box 2150
Atlanta, Georgia 30301
(U.S.A. only)

General Business Group/International
44 South Broadway
White Plains, New York 10601
U.S.A.
(International)

IBM System/34 System Support Program Logic Manual: System (File No. S34-36) Printed in U.S.A. LY21-0050-0

LY21-0050-0